Spiele entwickeln mit dem Game Maker

Originalversion: 5.3 (4. April 2004)

Übersetzungsversion: 5.3 (14. Juni 2004)

Geschrieben von Mark Overmars

ins deutsche Übersetzt von www.gm-d.de

Inhaltsverzeichnis

1. Du willst dein eigenes Computerspiel erstellen

Computerspielen macht Spass. Aber es macht noch mehr Spass ein Computerspiel selbst zu erstellen und es anderen zum Spielen zu geben. Leider ist das Erstellen solch eines Spieles nicht so einfach. Kommerzielle Computerspiele (solche die du kaufst) brauchen oft sogar 1 - 3 Jahre Entwicklungszeit, obwohl in einem Team von meist 10 - 50 Leuten gearbeitet wird. Die Budgets steigen oft bis in die Millionen-Höhe. Und all diese Leute sind bestens ausgebildet: Programmierer, Designer , Klangtechniker , usw. Soll das jetzt heißen, dass es unmöglich ist ein eigenes Computerspiel zu erstellen? Sicher nicht. Natürlich kannst du nicht davon ausgehen, dass du dein eigenes Quake oder Age of Empires in ein paar Wochen erstellst. Dies ist aber auch nicht nötig. Ein paar einfachere Spiele wie Tetris, Pacman, Space Invaders, usw. reichen auch schon aus. Auch solche Spiele machen Spass beim Spielen und sind viel einfacher zu erstellen. Natürlich braucht man auch da für gute Programmierkenntnisse um Sounds, Grafiken usw. einzubinden.

Aber hier kommt der Game Maker ins Spiel. Der Game Maker wurde programmiert um das Erstellen solcher Spiele einfacher zu machen. Du brauchst nicht einmal eine Programmiersprache beherrschen. Eine intuitive und einfach handzuhabende Drag & Drop Technik erlaubt es dir dein Spiel schnell zu erstellen. Du kannst Bilder importieren oder sogar selbst erstellen, Sprites (animierte Bilder), Sounds und vieles Andere nutzen. Du bestimmst ganz einfach die Eigenschaften der Objekte in deinem Spiel und zeigst ihnen, wie sie sich verhalten sollen. Du kannst sogar Bilder auf einer Karte bewegen lassen. Wenn du die volle Kontrolle über alles haben willst, gibt es auch noch eine einfach handzuhabende Programmiersprache mit der du genauestens bestimmen kannst was in deinem Spiel geschehen soll.

Der Game Maker beschäftigt sich mit 2 dimensionalen Spielen. Also keine 3D-Welten wie in Quake. Aber lass dich davon nicht abhalten. Viele großartige Spiele wie Age of Empires, die Command & Conquer Serie, und Diablo benutzen 2 dimensionale Sprite-Technologien und sehen manchmal trotzdem 3 dimensional aus. Ausserdem ist das Design von 2D (2 dimensionalen) Spielen nicht so zeitaufwendig wie das von 3D-Spielen.

Du kannst den Game Maker völlig kostenlos nutzen. Und du kannst deine selbst erstellten Spiele gratis weitergeben, oder auch verkaufen. Schau dir die beiliegenden Lizenzbestimmungen für nähere Details an. Du solltest aber den Game Maker registrieren. Mit deiner Registrierung förderst du die Entwicklung des Game Makers, einige erweiterte Funktionen des Game Makers werden freigeschaltet und das kleine Logo beim Starten des Spieles verschwindet.

Dieses Dokument wird dir alles über den Game Maker berichten, was du wissen musst, sogar wie du dein eigenes Spiel erstellst. Bitte denke dran, dass sogar mit Programmen wie dem Game Maker das Erstellen von Computerspielen nicht einfach ist. Es gibt so viele wichtige Aspekte die bei der Programmierung solcher Spiele wichtig sind, z.B.: Spielablauf, Grafik, Sound, das Benutzerinterface, usw. Fange mit einfachen Spielen an und du wirst merken, dass das Programmieren von Spielen sehr viel Spass macht. Komme auch mal auf unsere Internetseite unter http://www.gamemaker.nl und besuche dort das Forum (englischsprachig!). Dort warten eine Menge Beispiele, Tutorials, Ideen und Hilfen. Und bald wirst du ein Meister im Spiele erstellen sein.

Also viel Spass.

2. Installation

Vermutlich hast du den Game Maker schon installiert, falls dem nicht so ist, hier kommt eine Erklärung zum Installationsprozess des Game Makers. Führe einfach die Datei gmaker.exe aus. Folge den Installationsanweisungen des Setups. Du kannst das Programm installieren wohin du möchtest, aber am einfachsten ist es den vorgegebenen Pfad zu verwenden. Ist die Installation beendet, wirst du eine neue Programmgruppe im Startmenü finden. In dieser Programmgruppe kannst du den Game Maker starten oder dich über ihn informieren.
Ausserdem Game Maker wird auch eine Hilfe Datei installiert.

Wenn du den Game Maker zum ersten Mal startest, wirst du gefragt, ob du das Programm im normalen oder im erweiterten Modus starten möchtest. Wenn du vorher noch nie mit solch einem Programm gearbeitet hast, also noch keine Erfahrungen damit hast, solltest du lieber den normalen Modus wählen (also wähle nein). Im normalen Modus werden mehrere Optionen versteckt. Du kannst ganz einfach zum erweiterten Modus wechseln indem du im File-Menü auf Advance Mode klickst.

Im Installationsordner (normalerweise C:/Programme/Game_Maker5/) gibt es ein paar Ordner:
Examples: dieser Ordner beinhaltet Beispiel Spiele zur Anschauung
Lib: beinhaltet die Aktionsbibliotheken, wenn du zusätzliche Aktionen installieren willst, musst du sie in diesen Ordner kopieren
Sprites: in diesem Ordner gibt es ein paar Sprites die du benutzen kannst. Durch die normale Installation werden ein paar Standardsprites installiert, aber von der Game Maker Website http://www.gamemaker.nl kannst du weitere Ressourcen wie Hintergründe (Backgrounds), Sounds, Sprites etc. herunterladen
Backgrounds, Sounds: das sind ganz normale Ordner in denen die Hintergrundbilder, Sounds etc beinhaltet sind

Der Game Maker benötigt einen Pentium kompatiblen PC mit Windows 98,NT,2000,Me,XP oder einer späteren Windows Version (die Spiele selber laufen auch unter Windows 95). Es wird eine Bildschirmauflösung von mind. 800x600 und 65000 Farben (16bit) Farben benötigt. Ausserdem ist mindestens DirectX 5 erforderlich. Wenn du ein Spiel erstellst oder Testen möchtest, brauchst du mindestens 32MB freien Speicher auf deiner Festplatte. Der benötigte Speicher hängt von dem Spiel ab, welches du spielst.

3. Die Registrierung

Wie oben genannt, kann Game Maker gebührenfrei benutzt werden. Es gibt aber Einschränkungen bei den von dir erstellten Spielen. Im Spiel werden dann kleine "nag-screens" (nervige Hinweisfenster) gezeigt. Für weitere Details lies dir bitte die enthaltende "license agreement" (Lizenzbestimmung) aufmerksam durch.

Die folgenden Funktionen werden durch die Registrierung freigeschaltet:

- Kein Game Maker Logo beim Spielstart
- Keine Erinnerung zur Registrierung
- Zwei erweiterte Aktionssets
- Die Möglichkeit den Game Maker durch Dlls zu erweitern
- Ein Partikel-System um Feuerwerk, Flammen, Regen und andere Effekte zu produzieren
- Die Möglichkeit Mehrspieler-Spiele über Netzwerke zu spielen
- Funktionen um Ressourcen (Sprites, Objekte, Räume) zu modifizieren, während das Spiel läuft
- Eine Anzahl erweiterter Zeichenfunktionen
- Eine Anzahl von Funktionen um Datenstrukturen zu erstellen und zu benutzen
- Funktionen zur Bewegungsplanung (Wegfindung)

Die Registrierungsgebühr für den Game Maker beträgt 15 Euro oder diesen Gegenwert in einer anderen Währung, z.B. 18 Dollar. Es gibt verschiedene Wege sich anzumelden. Die einfachste ist die Online-Anmeldung mithilfe eines sicheren Kreditkartenzahlungsverfahren oder über einen PayPal Account. Alternativ kann die Summe auch an unser Bankkonto überwiesen werden, per Geldbrief/Zahlungsanweisung (money order) oder Bar zugesandt werden. Weitere Informationen können auf der Game Maker Registrierungsseite gefunden werden:
http://www.gamemaker.nl/registration.html

Um deine Kopie vom Game Maker zu registrieren benutze die oben angegebene Website oder verwende die Option "Registration" aus dem "Help"-Menü. Im unteren Teil des Fensters befindet sich ein Knopf "Registration" - betätige ihn. Er bringt dich zu unserer Internetseite, wo alle möglichen Anmeldeverfahren aufgezeigt werden - auch die Online-Registrierung.
Wenn deine Anmeldung empfangen wurde, wird eine e-mail an dich gesendet, welche den einzutragenden Namen und Registrierungsschlüssel enthält und wie er eingegeben werden soll. Um den Schlüssel einzugeben wähle "Registration" aus dem "Help"-Menü. Drücke den "Enter"-Knopf. Gib den Namen und den Zeichencode ein, drücke auf "OK". Wenn du keinen Fehler gemacht hast, ist das Programm jetzt registriert.

4. Der Grundgedanke

Bevor du dich eingehender mit den Möglichkeiten des Game Makers befasst, ist es von Vorteil, wenn du erstmal den Grundgedanken des Programmes erfasst. Spiele, die mit dem Game Maker erstellt wurden, finden in einem oder mehreren "rooms" (Räumen) statt (Räume sind flach, nicht dreidimensional, aber sie können dreidimensional aussehende Grafiken enthalten). In diese Räume platzierst du objects (Objekte), die du innerhalb der Software definieren kannst. Typische Objekte sind Wände (walls), sich bewegende Bälle, die Spielerfigur, Monster, usw.. Manche Objekte, wie beispielsweise Wände, sind einfach nur da und machen gar nichts. Andere Objekte, wie die Spielerfigur, bewegen sich umher und reagieren auf die Eingaben des Spielers (Tastatur, Maus, und Steuerknüppel) und aufeinander. Beispielsweise, wenn die Spielerfigur auf ein Monster trifft, "stirbt" sie möglicherweise. Objekte sind der wichtigste Bestandteil der mit Game Maker erstellten Spiele - beschäftigen wir uns eingehender damit.

Zu allererst benötigen die meisten Objekte ein Bild, um sie auf dem Bildschirm darzustellen. Solche Bilder werden "sprites" genannt. Ein sprite ist oftmals kein einzelnes Bild, sondern ein Reihe von Bildern, welche nacheinander angezeigt werden, um eine Animation zu erzeugen. Auf diese Art wird erreicht, dass es so scheint, als laufe die Spielerfigur, als rotiere ein Ball, als explodiere ein Raumschiff, usw.. Während des Spiels kann das "sprite" für ein bestimmtes Objekt wechseln. (so sieht die Spielerfigur unterschiedlich aus, wenn sie nach links bzw. rechts läuft.) Du kannst deine eigenen Sprites im Game Maker erstellen oder sie aus Dateien laden (z. B. animierte GIF's).

Gewisse Dinge geschehen mit den Objekten. Solche Ereignisse nennt man "events". Objekte reagieren mit bestimmten "actions" (Aktionen) auf eintretende "events" (Ereignisse). Es gibt eine Vielzahl von unterschiedlichen Ereignissen, die auftreten können und eine große Zahl von diversen "actions" (Aktionen), mit denen die Objekte darauf reagieren können. Beispielsweise wird ein "creation event" ausgelöst, wenn ein Objekt Erschaffen wird. (genauer: Wenn eine Instanz eines Objektes erschaffen wird; es können mehrere Instanzen von einem Objektes erzeugt werden.) Wenn zum Beispiel ein "ball object" erzeugt wird, kannst du diesem eine Bewegungsaktion zuweisen, so dass er sich fortbewegt.
Wenn zwei Objekte aufeinander treffen, wird ein "collision event" (Kollisionsereignis) ausgelöst. In solch einem Fall kannst du den Ball anhalten oder in die entgegengesetzte Richtung bewegen. Du kannst auch einen Soundeffekt (Klang-) abspielen. Zu diesem Zweck kannst du im Game Maker "Sounds" (Klänge) definieren.
Wenn der Spieler eine Taste auf der Tastatur drückt, wird ein "keyboard event" (Tastaturereignis) ausgelöst und das Objekt kann darauf mit einer entsprechenden "action" (Aktion) reagieren - zum Beispiel sich in die jeweilige Richtung bewegen. Ich hoffe, du hast das Prinzip verstanden. Für jedes Objekt, das du erstellst, kannst du "actions" festlegen für unterschiedliche "events" und somit das Verhalten des Objektes definieren.

Wenn du deine Objekte definiert hast, ist es an der Zeit einen Raum zu beschreiben, in dem die Objekte agieren werden. Räume können als Level in deinem Spiel verwendet werden oder als unterschiedliche Orte/Gebiete. Es gibt bestimmte "actions", um von einem Raum zu einem anderen zu gelangen.
Räume besitzen einen Hintergrund. Das kann einfach eine Farbe sein oder ein Bild. Solche Hintergrundbilder können im Game Maker erstellt werden oder aus Dateien geladen werden. (Der Hintergrund kann eine Menge Sachen - aber vorläufig genügt es, wenn du ihn als etwas erachtest, was den Raum schöner aussehen lässt.) Als nächstes platziere die Objekte im Raum. Du kannst mehrere Instanzen desselben Objektes in einem Raum arrangieren. So kannst du beispielsweise nur einmal ein Wandobjekt definieren und es an verschiedenen Stellen im Raum einsetzen.

Auch kannst du mehrere Instanzen desselben Monsterobjektes einsetzen - solange sie alle das gleiche Verhalten haben sollen.

Nun ist es soweit, dein Spiel zu starten. Der erste "room" wird angezeigt und die Objekte darin erwachen zum Leben, wegen der "actions" in ihren "creation events". Sie reagieren aufeinander aufgrund ihrer "actions" in den "collision events und sie reagieren auf den Spieler durch die "actions" im "keyboard-" oder "mouse-event".

Zusammenfassend spielen folgende Dinge (oft als "resources" (~Spielkomponenten) bezeichnet) eine entscheidende Rolle:

· objects: sind die wahren/echten Dinge (entities) des Spiels
· rooms: die Orte/Gebiete (levels) in denen die Objekte existieren
· sprites: (animierte) Bilder, die verwendet werden, um ein Objekt darzustellen
· sounds: sie werden in Spielen als Hintergrundmusik oder Geräuscheffekt verwendet
· backgrounds: Die Bilder, welche als Hintergrund für die "rooms" genutzt werden

Es gibt noch weitere Spielkomponenten (resources): paths, scripts, data files, und time lines. Sie sind wichtig, für kompliziertere Spiele. Du kannst sie nur erreichen, wenn du den Game Maker im "advanced mode" betreibst. Mit ihnen befassen wir uns später in den fortgeschrittenen Kapiteln dieses Dokumentes.

5. Ein einfaches Beispiel

Sehen wir uns erst ein einfaches Beispiel an. Ich nehme an, dass du den Game Maker im "simple mode" benutzt.
Der erste Schritt ist es, das Spiel, welches wir machen wollen, zu beschreiben. (Das solltest du immer als Erstes machen, es erspart dir später viel Arbeit). Das Spiel wird sehr einfach: Ein Ball prallt zwischen ein paar Wänden ab. Der Spieler soll mit der Maus auf den Ball klicken. Hat er Erfolg, bekommt er einen Punkt.

Wie man sieht, benötigen wir hier zwei verschiedene Objekte: den Ball und die Wand. Wir brauchen auch zwei verschiedene Sprites: eben eines für den Ball und eines für die Wand. Dann brauchen wir noch einen Sound-Effekt, der abgespielt wird, wenn der Spieler einen Punkt macht. Wir werden nur einen Raum brauchen, in dem das Spiel statt findet. (Wenn du das Spiel nicht selbst machen willst, kannst du es aus dem "Examples"-Ordner unter dem Namen "touch the ball.gmd")

Machen wir zuerst die Sprites. Wähle "Add Sprite" aus dem "Add"-Menü (du kannst auch auf den Button auf der Symbolleiste klicken). Ein Fenster öffnet sich. In das "Name"-Feld schreibe "wall". Klicke auf den "Load Sprite"-Button und wähle ein passendes Bild. Das war's schon, und du kannst das Fenster mit einem Klick auf "OK" schliessen. Erstelle nun ein Ball-Sprite auf die selbe Art.

Als nächstes ist der Sound dran. Wähle "Add Sound" aus dem "Add"-Menü. Ein anderes Fenster öffnet sich. Gib dem Sound einen Name (z.B. "cought") und wähle "Load Sound". Suche dir einen Sound aus (du kannst ihn dir mit der Play-Taste anhören). Wenn du zufrieden bist, klicke "OK".

Der nächste Schritt ist, die zwei Objekte zu erstellen. Machen wir zuerst das Wand-Objekt. Wähle "Add Object" aus dem "Add"-Menü. Ein Fenster öffnet sich, das ein bisschen komplizierter als die zwei zuvor aussieht. Auf der linken Seite befinden sich allgemeine Informationen über das Objekt. Gib dem Objekt einen passenden Namen und wähle das "wall"-Sprite aus dem Drop-down aus. Weil eine Wand solide ist (der Ball kann nicht durch), setze einen Haken bei "Solid". Das war alles.

Erstelle ein weiteres neues Objekt, benenne es "ball" und gib ihm das Ball-Sprite. Der Ball soll nicht solide sein. Für ihn müssen wir nun das Verhalten definieren. In der Mitte siehst du eine leere Liste von "Events" (Ereignissen). Darunter befindet sich ein Button "Add Event". Betätige ihn und du siehst alle möglichen "Events". Wähle das "Creation"-Event. Es wird nun zur Liste hinzugefügt. Ganz rechts siehst du alle möglichen "Actions" (Aktionen, Befehle) in Gruppen zusammengefasst. Von der "Move" (bewegen) - Gruppe wähle die Aktion mit den acht roten Pfeilen und ziehe sie in die Aktionsliste in der Mitte. Nun erscheint ein Dialog, in dem du die Richtung der Bewegung wählen kannst. Klicke alle acht Pfeile an (jetzt wird eine zufällige Richtung gewählt). Den "speed" (Geschwindigkeit) kannst du bei 8 lassen. Schliesse nun den Dialog. Jetzt beginnt sich der Ball zu bewegen, wenn er erstellt wird. Als zweites müssen wir definieren, was im Fall einer Kollision mit der Wand passieren soll. Drücke auf "Add Event". Klicke auf den Button für "Collision"-Events und wähle im Drop-down das Wand-Objekt. Für dieses Event brauchen wir die "bounce" (abprallen) - Aktion (Du kannst dir ansehen, was eine Aktion macht, indem du den Cursor darüber hältst). Zum Schluss müssen wir noch einstellen, was passieren soll, wenn der Benutzer über dem Ball die linke Maustaste drückt. Füge das entsprechende Event hinzu, und wähle "left mouse button" aus dem Drop-down. Für dieses Event brauchen wir mehrere Aktionen: -eine, die einen Sound abspielt (in der "main1"-Gruppe), -eine, die die Punktezahl ("score") verändert (in der "score"-Gruppe) und zwei, die den Ball zu einer neuen, zufälligen Position bewegen und ihn wieder bewegen lassen (so wie im creation"-Event).

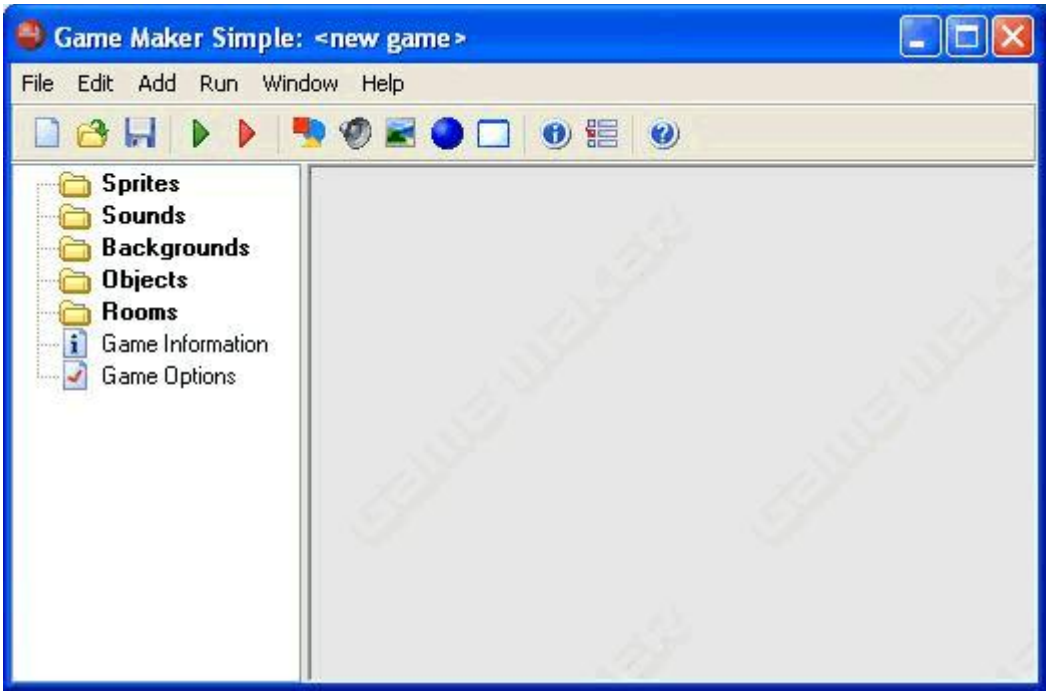
Für die "score"-Aktion, tippe den Wert 1 ein und setze einen Haken bei "Relative". Das bedeutet, dass 1 zur Punktezahl addiert werden soll.
Übrigens: Wenn du einen Fehler machst, kannst du die Einstellungen ändern, indem du auf die Aktion in der Liste doppelklickst.

Unsere Objekte sind fertig, jetzt bleibt noch der Raum übrig. Füge (über das "Add"-Menü) einen Raum zum Spiel hinzu. Rechts im Fenster siehst du nun den leeren Raum. Links sind einige Register:
für Hintergründe (background), für allgemeine Einstellungen (settings), Objekte (objects), ...
Die voreingestellte Raumgrösse (im Register "settings") ist etwas zu gross. Stelle sie herunter. Jetzt ist es Zeit, unsere Objekte zu platzieren: Wähle den "objects"-Register aus und wähle im Drop-down unten das Wand-Objekt. Umrahme den Raum mit Wänden. Zum Schluss platzierst du ein oder zwei Bälle im Raum. So, unser Spiel ist fertig! Klicke auf den "Run"-Button (der grüne Pfeil) um es zu testen. Wenn du alles richtig gemacht hast, beginnt sich der Ball zu bewegen. Klicke ihn an und achte darauf, was passiert. Du kannst das Spiel mit der <Esc>-Taste beenden.

Glückwunsch!
Du hast dein erstes kleines Spiel gemacht! Aber ich denke, es ist Zeit, etwas mehr über den Game Maker zu lernen...

6. Die Benutzerführung

Nach dem Start vom Game Maker erscheint folgendes Fenster:



(Hier siehst du den Anfänger-Modus vom Game Maker. Im Expertenmodus (Advanced) gibt es einige zusätzliche Einträge, Siehe Kapitel 14). Links siehst du verschiedene Ressourcen, wie Sprites, Soundeffekte, Hintergründe, Objekte, Räume und noch zwei andere: Spielinformation und Spieloptionen. Oben gibt es ein für Windows typisches Menü und eine Symbolleiste. In diesem Kapitel werde ich die einzelnen Menüeinträge, Buttons usw. beschreiben. In den folgenden Kapitel werde ich einige davon genauer beschreiben. Es gibt immer verschiedene Wege für das gleiche Ziel: Benutzen eines Befehls des Menüs, Klicken eines Button oder Rechtsklicken auf eine Ressource.

6.1 File menu - Das Datei Menü

Im File menu sind einige nützliche Befehle für das Laden und Speichern von Dateien, plus einige spezielle Befehle:

-New: Benutze diesen Befehl um ein neues Spiel zu erstellen. Wenn das aktuelle Spiel geändert wurde, wirst du gefragt ob die Änderungen gespeichert werden sollen. Es gibt dafür auch einen Button in der Symbolleiste.

- Open: Öffnet eine Spieldatei. Game Maker-Dateien haben die Dateierweiterung .gmd. Hierfür existiert auch ein Button in der Symbolleiste. Wenn man eine Datei in Windows markiert und dann mit der Maus in das Hauptfenster vom Game Maker zieht, wird diese ebenfalls geöffnet.
- Recent Files: Benutze dieses Untermenü um zuletzt geöffneten Dateien zu öffnen.

- Save: Speichert die Datei unter dem aktuellen Namen. Wenn kein Name angegeben wurde, wird nach einem Neuen gefragt. Du kannst diesen Punkt nur benutzen wenn die Datei geändert wurde. Es gibt dafür auch einen Button in der Symbolleiste.

- Save As: Speichert das Spiel unter einem anderen Namen. Du wirst danach gefragt.

- Create Executable: Wenn dein Spiel fertigt ist willst du es wahrscheinlich anderen zum Spielen geben. Mit diesem Befehl kannst du eine Version deines Spieles machen, welche nicht den Game Maker zum Spielen benötigt. Das ist einfach eine ausführbare Datei die du anderen geben kannst. Du findest mehr über das Verteilen von Spielen im dazugehörigen Kapitel.

- Advanced Mode: Dieser Befehl wechselt zwischen dem Anfänger- und Expertenmodus des Game Makers. Im Expertenmodus gibt es einige zusätzliche Befehle und Ressourcen.

- Exit: Benutze diesen Befehl um, den Game Maker zu beenden. Wenn du das aktuelle Spiel geändert hast, wirst du gefragt ob die Änderungen gespeichert werden sollen.

6.2 Edit menu - Das Bearbeiten Menü

Das Edit Menu beinhaltet einige Befehle die sich auf die aktuell gewählte Ressource (Objekt, Sprite, Sound, usw.) bezieht. Abhängig von der Art der Ressource sind manche Befehle nicht verfügbar.

- Insert resource: Fügt eine neue Instanz der aktuell gewählten Art der Ressource vor dem aktuellen hinzu. Ein Fenster öffnet sich wo du die Eigenschaften der Ressource ändern kannst. Darauf wird in späteren Kapiteln genauer eingegangen.
- Duplicate: Dupliziert die aktuelle Ressource. Ein Fenster öffnet sich worin das Duplikat verändert werden kann.

- Delete: Löscht die gewählte Ressource (oder Gruppe von Ressourcen). Sei vorsichtig, da dies nicht rückgängig gemacht werden kann.

- Rename: Gibt der Ressource einen neuen Namen. Das kann auch im Eigenschaftsfenster der Ressource geschehen oder man kann die Ressource auswählen und dann auf den Namen klicken um ihn zu ändern.

- Properties: Öffnet ein Fenster um die Eigenschaften der Ressource zu ändern. Bedenke alle Änderung werden im Hauptfenster übernommen. Du kannst viele gleichzeitig bearbeiten. Außerdem kannst du die Eigenschaften durch Doppelklicken auf die Ressource ändern.

- Alle diese Befehle können auf verschieden Arten gewählt werden. Rechts klicken auf eine Ressource oder Ressourcengruppen und das gewünschte Pop-up Menu erscheint.

6.3 Run menu - Das Ausführen Menü

Dieses Menü dient dem Ausführen des Spieles. Es gibt zwei Möglichkeiten ein Spiel auszuführen.

- **Run normally**. Führt das Spiel so wie es ist ganz normal aus.
- **Run in Debug mode**. Führt das Spiel im Debug Modus aus. In diesem Menü kannst du einige Aspekte des Spieles, wie Variablen, ansehen, es pausieren lassen oder schrittweise ausführen. Dies ist nützlich wenn etwas fortgeschritteneres nicht richtig läuft.

Wenn das Spiel fertig ist kannst du mit dem "Create Executable" Kommando im Dateimenü eine Exe Datei erzeugen.

6.4 Add menu - Das Hinzufügen Menü

In diesem Menü können neue Ressourcen jeder Art hinzugefügt werden. Für jede dieser Ressourcen gibt es auch einen Button in der Symbolleiste und ein Tastenkürzel.

6.5 Window menu - Das Fenster Menü

In diesem Menü kannst du nützliche Befehle für die verschiedenen Fenster im Hauptfenster finden.

- Cascade: Ordnet alle Fenster kaskadiert hintereinander an.

Arrange Icons: Ordnet die verkleinerten Eigenschaftsfenster. (Nützlich wenn der Hauptbildschirm vergrössert wird.)

- Close All: Schliesst alle Eigenschaftsfenster und fragt den Benutzer ob die Änderungen gespeichert werden sollen.

6.6 Help menu - Das Hilfe Menü

Einige Befehle über die Hilfe.

- Contents: Hier kann die Game Maker Hilfe angesehen werden, welche größtenteils diesem Dokument gleicht.

- Registration: Obwohl Game Maker gratis benutzt werden kann, bist du dazu eingeladen das Programm zu registrieren.

Es zeigt den Nag-Bildschirm, der manchmal erscheint, nicht mehr an und hilft für die weitere Entwicklung des Programms. Im Kapitel 3 kannst du Informationen über die Registrierung des Programms finden.
Wenn du dich registriert hast kannst du den erhaltenen Registrierungsschlüssel eingeben.

- Website: Verbindet dich zur Webseite des Game Maker, wo du Informationen über die aktuelle Version des Game Maker und eine Sammlung von Spielen und Ressourcen finden kannst. Es wird empfohlen mindestens einmal im Monat nach neuen Informationen Ausschau zu halten.
- Forum: Verbindet dich zum Forum des Game Maker, wo dir viele Leute Fragen beantworten und Hilfe zu Problemen geben.

- About Game Maker: Zeigt eine kurze Informationen über die Version des Game Maker an.

6.7 The resource explorer - Der Ressourcen Explorer

Links im Hauptfenster findest du den Ressourcen Explorer. Hier siehst du in einer Baumstruktur alle Ressourcen deines Spieles. Es arbeitet genauso wie der Windows Explorer und ist fast genauso zu bedienen. Wenn ein Eintrage ein + Zeichen vorne hat, kannst du darauf klicken und die Ressourcen darin werden gezeigt. Durch Klicken auf das - Zeichen verschwinden diese wieder. Du kannst den Namen der Ressource (ausser das Oberste) durch Auswählen (durch einen einfachen Klick) ändern. Doppelklicken auf eine Ressource ändert deren Eigenschaften. Benutze die rechte Maustaste, um die gleichen Befehle wie im Edit Menü zu erhalten. Nun kannst du die Reihenfolge der Ressourcen mit der Maus durch Klick und Verschieben ändern. (Der neue Platz muss richtig sein. Also du kannst keine Soundressourcen in andere, z.B. Sprite verschieben).

7. Sprites Definieren

Sprites sind die visuellen Darstellungen aller Objekte im Spiel. Ein Sprite ist entweder ein Einzelbild, mit irgendeinem Bildbearbeitungsprogramm gezeichnet, oder ein Set von Bildern, welche nacheinander abgespielt, wie eine Animationsbewegung aussehen.
Die folgenden vier Bilder zeigen ein Sprite für einen Pacman der sich nach rechts bewegt.

Wenn du dein Spiel machst, beginnst du eine Auswahl von schönen Sprites für die Objekte in deinem Spiel zu sammeln. Viele Sammlungen von interessanten Sprites können auch auf der Internetseite vom Game Maker gefunden werden. Sprites können auch im Internet gefunden werden, normalerweise in der Form von animierten Gif Dateien. Um einen Sprite hinzuzufügen, wähle den Eintrag Add Sprite vom Add Menü, oder benütze den entsprechenden Button in der Symbolleiste. Das folgende Dialogfenster öffnet sich:

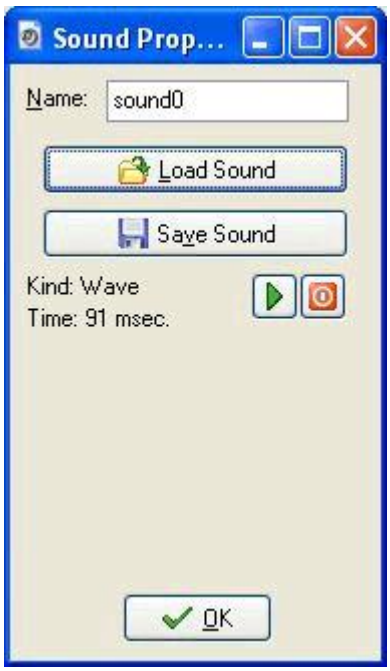
Ganz oben kann der Name des Sprites angegeben werden. Alle Sprites (und alle anderen Ressourcen) haben einen Namen. Am besten gibst du jedem Sprite einen beschreibenden Namen. Gehe sicher das alle Ressourcen verschiedene Namen bekommen. Obwohl dass nicht unbedingt notwendig ist, ist es sehr empfehlenswert nur Buchstaben und Ziffern und das Underscore Symbol ("_") im Namen des Sprites (und allen anderen Ressourcen) zu verwenden und ihn mit einem Buchstaben beginnen lassen. Besonders sollten keine Leerzeichen enthalten sein. Das wird sehr wichtig sein, wenn du einen Code erstellst. Um ein Sprite zu laden, drücke den Button "Load Sprite". Eine Dialogfenster öffnet sich, wo du das Sprite angeben kannst. Game Maker kann viele verschiedene Grafikdaten laden. Wenn du eine animierte Gif Datei lädst, bilden die verschiedenen Einzelbilder das Sprite mit Animation. Wenn das Sprite geladen ist, wird das erste Einzelbild rechts gezeigt. Wenn es mehrere Einzelbilder gibt, kannst du mit den Pfeiltasten das nächste Bild ansehen. Die Auswahlbox Transparent zeigt, ob der Hintergrund transparent sein soll. Die meisten Sprites sind transparent. Der Hintergrund wird von der linken, unteren Farbe des Pixels im Bild bestimmt. Gehe sicher dass sonst kein Pixel im aktuellen Bild diese Farbe hat. (Bedenke, dass Gif-Dateien öfters ihre eigene Transparenzfarbe haben. Diese Funktion wird nicht im Game Maker unterstützt).

Mit dem Button Edit Sprite kannst du Sprites bearbeiten, oder sogar eine komplett neuen Sprite erstellen. Für nähere Informationen für das Erstellen und Verändern von Sprites, siehe in Kapitel 16.

8. Sound und Musik

Viele Spiele benutzen bestimmte Soundeffekte und manche Hintergrundmusik. Viele nützliche Soundeffekte können auf der Homepage vom Game Maker gefunden werden. Noch mehr können auf anderen Seiten im Internet gefunden werden.

Um deinem Spiel eine Soundressource hinzuzufügen, benutze den Add Sound Eintrag im Add Menu oder den entsprechenden Button in der Symbolleiste. Das folgende Dialogfenster öffnet sich:



Drücke den Button "Load Sound" um einen Sound zu laden. Eine Dateiauswahlbox öffnet sich, wo die Sounddatei gewählt werden kann. Es gibt zwei Arten von Sounddateien, Wave-Dateien und MIDI-Dateien (für Informationen über MP3-Dateien lese Kapitel 17) Wave-Dateien werden für kurze Soundeffekte genutzt. Sie benötigen eine Menge Speicher aber werden sofort abgespielt. Benutze sie für alle Soundeffekte im Spiel. MIDI-Dateien gehen einen anderen Weg. Sie benötigen viel weniger Speicher, aber sie sind auf Hintergrundmusik beschränkt. Ausserdem kann nur eine MIDI-Datei gleichzeitig abgespielt werden. Wenn die Musikdatei geladen ist wird die Art und Länge angezeigt. Drücke den Play Button um die Datei abzuspielen. Es gibt auch einen Button Save Sound, damit wird der aktuelle Sound gespeichert. Dieser Button wird zwar selten benötigt, aber er könnte benötigt werden, wenn die Originaldatei verloren gegangen ist.

9. Hintergründe

Die dritte Art von einfachen Ressourcen sind Hintergründe. Hintergründe sind üblicherweise große Bilder die als Hintergründe (oder Vordergründe) für einen Raum, wo das Spiel stattfindet, benutzt werden. Hintergrundbilder werden oft so erstellt, dass sie ein Feld ohne visuelle Beschränkungen teilen. Einige solcher Hintergründe gibt es auf der Internetseite vom Game Maker. Noch mehr davon gibt es auf vielen anderen Internetseiten.

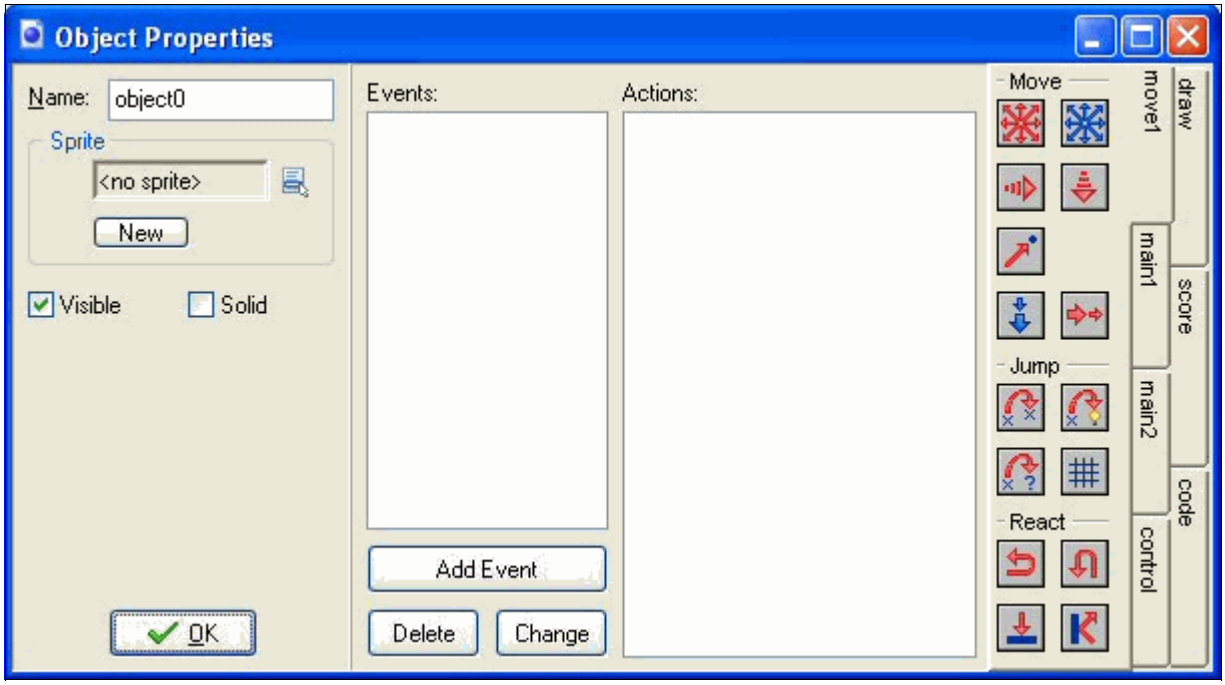
Um deinem Spiel einen Hintergrund hinzuzufügen, benutze den "Add Background" Eintrag im "Add Menu" oder den entsprechenden Button in der Symbolleiste. Das folgende Dialogfenster öffnet sich:

Drücke den Button "Load Background" um eine Hintergrunddatei zu laden. Der Game Maker unterstützt viele Bildformate. Bildschirmhintergründe können nicht animiert sein!
Die Kontrollbox Transparent zeigt an, ob der Hintergrund teilweise transparent ist. Meistens sind Hintergründe nicht transparent, daher ist nicht vorgegeben. Als Transparenzfarbe wird der linke, unterste Pixel verwendet.
Du kannst den Hintergrund verändern oder einen neuen erstellen durch Drücken des Button "Edit Background".
Für mehr Informationen siehe Kapitel 18.

10. Objekte Definieren

Mit Ressourcen kannst du einige nette Bilder und Soundeffekte deinem Spiel hinzufügen, aber die machen gar nichts. Wir kommen jetzt zur wichtigsten Ressource vom Game Maker, den Objekten. Objekte sind Einträge im Spiel die Dinge machen. Die meisten von ihnen haben ein Sprite als grafische Darstellung. Sie haben ein Verhalten da sie auf bestimme Ereignisse reagieren. Alle Dinge die du im Spiel siehst (ausser Bildschirmhintergründe) sind Objekte. (Oder um genauer zu sein, sie sind Instanzen der Objekte). Der Charakter, die Monster, die Bälle, die Wände etc. sind alles Objekte. Es ist möglich, dass bestimme Objekte nicht sichtbar sind, sie steuern aber bestimmte Aspekte im Spielablauf. Wichtig ist der Unterschied zwischen Sprites und Objekten. Sprites sind nur (animierte) Bilder die kein Verhalten haben. Objekte haben normalerweise auch ein Sprite aber Objekte besitzen ein Verhalten. Ohne Objekte gibt es kein Spiel.

Den Unterschied zwischen Objekten und Instanzen solltest du auch kennen. Ein Objekt beschreibt eine bestimmte Existenz, z.B. ein Monster. Es kann mehrere Instanzen eines Objektes im Spiel geben. Wenn wir über eine Instanz sprechen, meinen wir eine besondere Instanz eines Objektes. Wenn wir über ein Objekt sprechen meinen wir alle Instanzen eines Objektes.
Um ein Objekt deinem Spiel hinzuzufügen, wähle "Add Object" vom "Add Menu". Das folgende Fenster öffnet sich:



Das ist ziemlich komplex. Links gibt es allgemeine Informationen über das Objekt. In der Mitte ist eine Liste von Ereignissen welches dem Objekt passieren kann. Im nächsten Kapitel stehen die Details. Rechts gibt es verschieden Aktionen, die das Objekt ausführen kann. Das wird in Kapitel 12 gezeigt.
Wie immer kannst (und sollst) du deinem Objekt einen Namen geben. Dann kannst du das Sprite für das Objekt angeben. Klicke mit der linken Maustaste auf die Sprite Box oder dem Menü Button neben ihm. Ein Menu wird aufgeschlagen mit allen verfügbaren Sprites. Wähle das passende Sprite für das Objekt.
Falls du noch kein Sprite haben solltest kannst du auf den Button **New** klicken um eine neue Sprite Ressource hinzuzufügen und zu bearbeiten. Auch der Knopf **Edit** kann genutzt werden um das Sprite zu verändern.

Dies geht schneller als es erst in der Ressourcenliste zu suchen und dann zu bearbeiten.
Darunter sind zwei Kontrollboxen.

Visible gibt an ob die Instanzen dieses Objektes sichtbar sind. Klar, die meisten Objekte sind sichtbar, aber manchmal ist es nützlich unsichtbare Objekte zu nutzen. Beispielsweise kannst du sie als Wegpunkte für ein sich bewegendes Monster einsetzen. Unsichtbare Objekte reagieren auf Events und Kollisionen genauso wie sichtbare.
Die **Solid** gibt an ob das Objekt ein solides Objekt (wie eine Wand) ist. Kollisionen von soliden Objekten werden anders behandelt als die von nicht-soliden Objekten.
Das nächste Kapitel gibt darüber mehr Auskunft.

11. Ereignisse

Der Game Maker ist ereignisgessteuert. Das funktioniert so: Wann immer etwas im Spiel geschieht , erhalten die Instanzen der Objekte "events" (das sind Nachrichten/Meldungen, dass etwas passiert ist). Die Instanzen können dann auf diese Nachrichten reagieren, indem sie bestimmte "actions" (Aktionen/Reaktionen) auslösen. Für jedes Objekt muss festgelegt werden, auf welche Ereignisse ("events") es reagiert und welche "actions" ausgeführt werden sollen, wenn dieses Ereignis eintritt. Das hört sich kompliziert an, ist tatsächlich aber ganz einfach. Zuerst einmal müssen die Objekte für die meisten Ereignisse nichts machen. Für die "events", wo Reaktionen gefordert sind, steht ein simples "Drag&Drop-System" zur Verfügung, worin die entsprechenden "actions" zugeordnet werden können.

In der Mitte der "object property form" (Eingabemaske der Objekteigenschaften) befindet sich eine Liste der "events", auf die das Objekt reagieren muss. Anfangs ist sie leer. Du kannst Ereignisse hinzufügen, indem du auf den "Add event" Knopf klickst. Ein kleines Auswahlmenü mit den diversen Ereignissen klappt auf. Hier kannst du das "event" auswählen, welches du hinzufügen möchtest. Manchmal klappt noch ein zusätzliches Menü auf, mit weiteren Auswahlmöglichkeiten. Beispielsweise musst du für ein "keyboard event" die Taste (key) festlegen. Weiter unten wird aufgelistet welche Ereignisse es gibt plus Beschreibung. Ein Ereignis wird aus der Liste ausgewählt. Dieses ist es, welches wir momentan bearbeiten. Um ein anderes anzuwählen, klicke es einmal mit der Maus an. Rechts befinden sich die ganzen "actions", welche durch kleine "icons" dargestellt werden. Sie sind in mehrere Gruppen unterteilt. Im nächsten Kapitel werden sie beschrieben. Das Feld zwischen der Ereignisliste und den Aktionen ist die "action list". Hier werden die Aktionen für das angewählte Ereignis festgelegt. Um eine Aktion hinzuzufügen, ziehe sie (das icon) mit der Maus einfach rüber in diese Liste. Sie werden mit einer Kurzbeschreibung untereinander einsortiert. Für jede "action" werden einige Parameter erwartet. Dies wird auch im nächsten Kapitel erläutert. So, wenn nun einige Aktionen hinzugefügt wurden, sieht die Situation wie folgt aus:

Jetzt kannst du anfangen "actions" anderen Ereignissen zuzuordnen. Klicke mit der linken Maustaste auf das entsprechende Ereignis, um es auszuwählen und ziehe die "actions" in die jeweilige Liste. Du kannst die Sortierung der Liste per "Drag-and-Drop" ändern. Wenn du die "<Ctrl>"- Taste (Strg) während des Ziehens gedrückt hälst, machst du eine Kopie der "action". Du kannst sogar zwischen den "action"-Listen verschiedener Objekte "Drag-and-Drop" verwenden.

Wenn du mit der rechten Maustaste auf eine "action" klickst, erscheint ein Menü worin du die "action" löschen kannst (kann man auch mit der <Del>-Taste machen) oder Kopier- und Einfügevorgänge wählen kannst.
Wenn der Mauszeiger längere Zeit über einer "action" verharrt, erscheint eine längere Beschreibung der Aktion. Schau im nächsten Kapitel nach weiteren Informationen über "actions".

Um das momentan markierte "event" zusammen mit all seinen "actions" zu löschen, drücke den "Delete"-Knopf. (Ereignisse ohne irgendwelche Aktionen in ihrer Liste, werden automatisch beim Schliessen der Eingabemaske entfernt, so dass man sie nicht extra von Hand aussortieren muss.)
Wenn du die Aktionen einer Liste einem anderem Ereignis zuweisen willst (weil du Dich vielleicht für eine andere Taste entschieden hast), drücke einfach den Knopf "Change" und wähle ein anderes Ereignis. (Das neue "event" sollte nicht schon definiert sein!)

Wie oben erwähnt erscheint folgendes Auswahlmenü, wenn du auf den "Add event"-Knopf drückst:

Hier wählst du das hinzuzufügende Ereignis aus. Bei manchen erscheint ein weiteres Menü, mit zusätzlichen Auswahlmöglichkeiten. Hier folgt eine Beschreibung der unterschiedlichen Ereignisse(Erinnere Dich daran, dass man üblicherweise nur einige wenige verwendet).

Create Event
Dieses Ereignis wird ausgelöst, wenn eine Instanz eines Objektes erzeugt wird. Gewöhnlich verwendet man es, um die Instanz in Bewegung zu setzen oder bestimmte Variablen der Instanz zu festzulegen.

Destroy Event
Es wird ausgelöst, wenn die Instanz zerstört wird. Um genau zu sein - kurz vorher. Was bedeutet, dass die Instanz noch existiert, wenn das Ereignis ausgeführt wird! Die meiste Zeit über wird dieses Ereignis nicht verwendet aber man kann es beispielsweise nutzen, um den "score" (Punktestand) zu ändern oder ein anderes Objekt zu erschaffen.

Alarm Events
Jede Instanz besitzt 8 Alarmuhren. Du kannst diese Alarmuhren durch verschiedene "actions" einstellen (siehe nächstes Kapitel). Die Alarmuhr tickt dann runter bis sie 0 erreicht und in diesem Moment das "alarm event" auslöst. Um die "actions" für eine Alarmuhr anzuzeigen, musst du sie zuerst im Menü anwählen. "alarm clocks" sind sehr nützlich. Du kannst sie verwenden, um gewisse Dinge von Zeit zu Zeit geschehen zu lassen. Beispielsweise kann ein Monster seine Bewegungsrichtung alle 20 "steps" ändern (in so einem Fall muss eine Aktion innerhalb des Ereignisses die Alarmuhr wieder neu stellen).

Step Events
Das "step event" wird bei jedem "step" (Schritt) des Spiels ausgelöst. Hier kannst du "actions" platzieren, die kontinuierlich ausgeführt werden müssen. Beispielsweise wenn ein Objekt einem anderen folgen soll, kannst du hier die Bewegungsrichtung des Verfolgers angleichen. Sei trotzdem vorsichtig mit diesem Ereignis.
Packe nicht zu viele komplizierte "actions" in die "step events" von Objekten, von welchen viele Instanzen vorhanden sind. Das könnte das Spiel verlangsamen.
Um präzise zu sein - es gibt drei verschiedene "step events". Normalerweise braucht man nur das voreingestellte. Aber mittels des Menüs kann man auch die
Anderen verwenden ("begin step" (Schrittanfang) und "end step" (Schrittende)).
Das "begin step"-Ereignis wird am Anfang eines Schrittes ausgelöst, bevor andere Ereignisse auftreten. Das voreingestellte/normale "step event" wird ausgelöst, kurz bevor die Instanzen auf ihre neuen Positionen gesetzt werden. Das "end step"-Ereignis wird am Endes eines Schrittes ausgelöst, kurz bevor gezeichnet wird. Es wird üblicherweise benutzt, um beispielsweise das "sprite" in Abhängigkeit von der Bewegungsrichtung zu ändern.

Collision Events
Wann immer zwei Instanzen kollidieren (das geschieht, wenn ihre "sprites" überlappen), wird ein "collision event" ausgelöst. Genauer zwei - eins pro Instanz. Die Instanz kann auf dieses Ereignis reagieren. Wähle im Menü aus, bei welchem Objekt du ein Kollisionsereignis definieren willst. Danach platziere hier weitere
"actions".
Es gibt Unterschiede was passiert, wenn Instanzen mit soliden bzw. nicht-"soliden" Objekten kollidieren. Zuerst einmal, wenn keine "actions" im Kollisionsereignis definiert sind, passiert nichts. Die Instanz bewegt sich einfach weiter, auch wenn das andere Objekt "solid" ist. Wenn das Kollisionsereignis "actions" enthält
geschieht folgendes:
Wenn das andere Objekt "solid" ist, wird die an ihre vorherige Position Instanz (vor der Kollision) zurückgesetzt. Dann erst wird das Ereignis ausgelöst. Anschliessend wird die Instanz an die neue Position bewegt. Wenn das Ereignis beispielsweise die Bewegungsrichtung umkehrt, prallt die Instanz gegen die Wand, ohne anzuhalten. Wenn dann immer noch eine Kollision vorliegt, wird die Instanz an der vorherigen Stelle gehalten. Sie hält effektiv an.
Wenn das andere Objekt nicht "solid" ist, wird die Instanz nicht zurückgesetzt. Das Ereignis wird einfach ausgelöst mit der Instanz an der aktuellen Position. Auch gibt es keine 2. Kollisionsabfrage. Wenn du darüber nachdenkst, ist es die logische Konsequenz, die eintreten sollte. Weil das andere Objekt ja nicht "solid" ist,
kann es einfach durchquert werden. Das Ereignis informiert dich also darüber, dass dieses gerade geschehen ist.
Es gibt viele Anwendungsmöglichkeiten für das "collision event". Instanzen können es verwenden, um von Wänden abzuprallen. Du kannst es einsetzen, um Objekte zu vernichten, wenn sie z. B. von einer Kugel getroffen werden - und so weiter.

Keyboard Events
Wenn der Spieler eine Taste drückt, wird ein Tastaturereignis für alle Instanzen aller Objekte ausgelöst. Für jede Taste ein anderes Ereignis. Im Menü kannst du die
Taste bestimmen, für die du ein Tastaturereignis festlegen willst und anschliessend die "actions" hinüberziehen. Klar ausgedrückt brauchen nur wenige Objekte solche Ereignisse für wenige Tasten. Ein Ereignis wird in jedem Schritt (step) erzeugt, solange der Spieler die Taste drückt.
Es gibt zwei besondere Tastaturereignisse. Eines wird <No key> (keine Taste) genannt. Es wird in jedem Schritt erzeugt, wo keine Taste gedrückt ist. Das andere ist das <Any key> (irgendeine Taste)

was bei beliebigem Tastendruck ausgelöst wird. Wenn der Spieler mehrere Tasten drückt, wird für jede Taste ein Ereignis ausgelöst. Beachte, dass die Ereignisse des Ziffernblocks nur die korrespondierenden sind, wenn <Num lock> aktiviert ist.

Mouse Events

Ein "mouse event" wird immer dann für eine Instanz ausgelöst, wenn der Mauszeiger innerhalb dessen "sprite" positioniert ist. Abhängig von der Maustaste erhälst du ein "no button"(keine Taste), ein "left button"(linke Maustaste), ein "right button"(rechte Maustaste) oder ein "middle button"(mittlere Maustaste) Ereignis. Die Ereignisse werden in jedem Schritt erzeugt, solange der Spieler die Taste drückt. Die "press events" (Taste drücken) werden nur einmal beim Herunterdrücken der Taste erzeugt. Die "release events" (Taste loslassen) nur beim Loslassen der Taste. Beachte, dass diese Ereignisse nur auftreten, wenn der Mauszeiger über der Instanz ist. Wenn diese Ereignisse für beliebige Mauspostionen gelten sollen, verwende die globalen "press/release" Ereignisse stattdessen. Es gibt zwei besondere Mausereignisse. Das "mouse enter" (Beginn der Überdeckung) Ereignis und das "mouse leave" (Verlassen der Überdeckung) Ereignis.
Ersteres wird beim "Berühren" der Instanz ausgelöst, letzteres, wenn der Mauszeiger die Zone über der Instanz verlässt. Diese Ereigisse werden üblicherweise genutzt, um das "sprite" zu ändern oder einen SFX zu erzeugen. Schliesslich gibts noch Ereignisse für Steuerknüppel (Joysticks). Du kannst Aktionen für die 4 Richtungen angeben (diagonal werden beide Ereignisse ausgelöst). Bis zu 8 Knöpfe können verwaltet werden.
Du kannst all das für 2 Spielgeräte verwenden (1. und 2. Steuerknüppel).

Other Events

Es gibt eine Anzahl anderer Ereignisse, welche in bestimmten Spielen nützlich sind. Man findet sie in diesem Menü. Folgende Ereignisse können hier gefunden werden:

• Outside: Dieses Ereignis wird ausgelöst, wenn die Instanz komplett ausserhalb des Raumes liegt. Typischerweise ein guter Moment, sie zu vernichten.
• Boundary: Dieses Ereignis wird ausgelöst, wenn die Instanz die Grenze des Raumes berührt.
• Game start: Dieses Ereignis wird für alle Instanzen des ersten Raumes ausgelöst, wenn das Spiel beginnt. Es wird vor dem "room start event" (siehe unten) aber nach dem "creation event" für die Instanzen innerhalb des Raumes. Es wird typischerweise nur in einem "Steuerungsobjekt" verwendet, um Hintergrundmusik zu starten oder Variablen zu initialisieren oder ein paar Daten zu laden.
• Game end: Dieses Ereignis wird für alle Instanzen am Ende des Spiels ausgelöst. Auch dieses wird in einem Objekt verwendet. Es wird verwendet, um beispielsweise Spieldaten zu speichern.
• Room start: Dieses Ereignis wird am Anfang eines Raumes für alle Instanzen ausgelöst. Es geschieht nach den "creation events".
• Room end: Wird für alle existierenden Instanzen am Ende des Raumes ausgelöst.
• No more lives: Game Maker hat ein eingebautes Lebenssystem. Es gibt eine "action" um die Anzahl der "lives" ("Leben"), zu setzen und zu verändern. Wann immer dieser Wert auf 0 sinkt (oder tiefer), wird dieses Ereignis ausgelöst. Es wird benutzt, um das Spiel zu beenden oder neu zu starten.
• No more health: Game Maker hat ein eingebautes Lebensenergie. Es gibt eine "action", um diesen Wert zu setzen und zu verändern. Wann immer dieser Wert auf 0 oder weniger sinkt, wird dieses Ereignis ausgelöst. Es wird normalerweise verwendet. um die Anzahl der "lives " zu reduzieren oder das Spiel zu neu zu starten.
• End of animation: Wie oben erwähnt, besteht eine Animation aus einer Anzahl Einzelbilder, welche nacheinander angezeigt werden. Wenn das letzte angezeigt wurde beginnt es wieder von vorne. Das Ereignis wird genau hier ausgelöst.

Es kann verwendet werden, um die Animation zu ändern oder die Instanz zu vernichten.
• End of path: Dieses Ereignis wird ausgelöst, wenn die Instanz einem Pfad (path) folgt und an dessen Ende angelangt. Siehe Kapitel 22 für mehr Information über Pfade.
• User defined: Es gibt acht dieser Ereignisse. Sie werden normalerweise nie ausgelöst, solange du sie nicht via Programmcode auslöst.

 **Drawing Event**
Instanzen, wenn sie sichtbar sind, zeichnen ihr "sprite" in jedem "step" auf den Bildschirm. Wenn du "actions" im "drawing event" definierst, wird das sprite nicht gezeichnet aber die "actions" stattdessen ausgeführt. Das kann verwendet werden, um etwas anderes, als das "sprite" zu zeichnen oder erst die Parameter des "sprite" zu ändern. Es gibt eine Anzahl von "drawing actions" welche extra für das "drawing event" bestimmt sind. Beachte, dass das "drawing event" nur ausgelöst wird, wenn das Objekt sichtbar ist.
Beachte auch, das unabhängig von dem was du darstellst, Kollisionsereignisse auf dem zugehörigen "sprite" basieren.

 **Key press events**
Dieses Ereignis ist dem Tastaturereignis ähnlich. Es wird ausgelöst beim Runterdrücken der Taste anstatt permanent. Es ist nützlich wenn du nur einmal etwas bei Tastendruck passieren lassen willst.

 **Key release**
Wie oben, nur beim Loslassen der Taste.

In manchen Situationen ist es wichtig zu wissen, wie der Game Maker Ereignisse verarbeitet. Dies läuft so ab:
· Begin step events
· Alarm events
· Keyboard, Key press, and Key release events
· Mouse events
· Normal step events
· (nun werden alle Instanzen an ihre Positionen gesetzt)
· Collision events
· End step events
· Drawing events
Die creation, destroy und anderen Ereignisse werden ausgelöst, wenn die korrespondierenden Umstände eintreten.

12. Aktionen

Aktionen beschreiben Dinge, die in einem Game Maker - Spiel passieren. Sie werden in Ereignissen von Objekten platziert. Immer, wenn das Ereignis stattfindet, werden sie ausgeführt, was ein bestimmtes Verhalten für Instanzen des Objekts hervorruft. Es gibt viele verschiedene Aktionen und es ist wichtig, dass du verstehst was sie tun. In diesem Kapitel werde ich die Standard-Aktionen beschreiben. Es wird möglicherweise zusätzliche Aktionsbibliotheken geben. Schau dafür auf der Website nach. Alle Aktionen kannst du in den Registern rechts im "object property"-Fenster finden. Es gibt sieben Sets. Wenn du den Cursor über eine Aktion hältst, erscheint eine kurze Beschreibung.

Ich wiederhole kurz: Um eine Aktion in ein Ereignis einzufügen, ziehe es einfach von der rechten Seite in die Aktionsliste. Du kannst die Reihenfolge der Aktionen in der Liste verändern - einfach wieder ziehen. Hältst du dabei die <Strg>-Taste, wird die Aktion kopiert. Du kannst auch Aktionen zwischen Listen in verschiedenen Objektfenster ziehen. Verwende die rechte Maustaste um Aktionen zu löschen, kopieren oder einfügen.
Wenn du eine Aktion in die Liste ziehst, öffnet sich ein Fenster (jedenfalls meistens), wo du bestimmte Parameter für die Aktion einstellen kannst. Die Parameter werden weiter unten, in den Aktionsbeschreibungen erklärt.

Zwei Typen von Parametern kommen in vielen Aktionen vor, deswegen werde ich sie hier beschreiben: Oben kannst du angeben, zu welcher Instanz die Aktion gehören soll. Der Standard ist "self", was die eigene Instanz ist. Meistens ist es das was du willst. Im Fall eines "collision event" kannst du hier auch die andere ("Other") Instanz die an der Kollision beteiligt ist angeben. So kannst du z.B. die andere Instanz zerstören. Oder du kannst die Aktion für alle Instanzen eines bestimmten Objekts ausführen lassen. So kannst du z.B. rote Bälle in blaue Bälle verwandeln. Der zweite Typ ist eine Box "Relative". Wenn du diese Box anwählst, sind die Werte, die du angibst, relativ zu den aktuellen Werten. Zum Beispiel kannst du so zur Punktezahl etwas dazu zählen, statt sie direkt zu verändern. Andere Parameter werden unten beschreiben. Du kannst Parameter später ändern, wenn du auf die Aktion doppelklickst.

12.1 Bewegungsaktionen

Diese Aktionen haben etwas mit der Bewegung von Objekten zu tun:

 **Start moving in a direction (Beginne Bewegung in eine Richtung)**
Verwende diese Aktion, um die Instanz in eine bestimmte Richtung zu bewegen. Du kannst diese Richtung mit den Pfeilbuttons bestimmen. Verwende den mittleren Button, um die Bewegung zu stoppen. Du musst auch die Geschwindigkeit (in Pixel pro Schritt) angeben. Der Standardwert ist 8. Verwende lieber keine negativen Geschwindigkeiten. Wenn du mehrere Richtungen angibst, wird die Richtung zufällig gewählt. So kannst du z.B. ein Monster entweder nach rechts oder links starten lassen.

 **Set direction and speed of motion (Richtung u. Geschwindigkeit der Bewegung setzen)**
Das ist der zweite Weg, eine Bewegung festzulegen. Hier kannst du eine genaue Richtung angeben. Das ist ein Winkel zwischen 0° und 360°. 0 ist nach rechts, die Richtung ist gegen den Uhrzeigersinn (90 ist daher nach oben). Wenn du eine zufällige Richtung willst, kannst du random(360) eingeben. Wie du noch sehen wirst, gibt die Funktion random eine zufällige Nummer kleiner als die angegebene Zahl zurück.

Wie du vielleicht bemerkt hast, gibt es auch noch eine Box "Relative". Wenn du einen Haken hinein machst, wird die neue Bewegung zur alten dazu addiert. Zum Beispiel, wenn sich die Instanz nach oben bewegst und du addierst eine Bewegung nach links, bewegt sich die Instanz nun nach oben links.

 **Set the horizontal speed (lege die horizontale Geschwindigkeit fest)**
Die Geschwindigkeit einer Instanz besteht aus einem horizontalen und einem vertikalen Teil. Mit dieser Aktion kannst du die horizontale Geschwindigkeit ändern. (Die vertikale bleibt gleich.)
Eine positive horizontale Geschwindigkeit ist eine Bewegung nach rechts, eine negative nach links. Verwende "Relative" um die horizontale Geschwindigkeit zu erhöhen (oder mit einer negativen Zahl zu vermindern).

 **Set the vertical speed (Lege die vertikale Geschwindigkeit fest)**
Wie oben, nur mit der vertikalen Geschwindigkeit.

 **Move towards a point (Auf einen Punkt zubewegen)**
Diese Aktion ist noch ein anderer Weg, um eine Bewegung festzulegen. Du gibst die Position und eine Geschwindigkeit an und die Instanz bewegt sich auf den Punkt zu (Sie stoppt nicht auf dem Punkt!) Wenn du z.B. ein Geschoss auf das Raumschiff zubewegen lassen willst, kannst du als Position raumschiff.x, raumschiff.y verwenden (Du wirst später noch über solche Variablen lernen). Wenn du "Relative" verwendest, wird die Position relativ zu der der aktuellen Instanz genommen (die Geschwindigkeit nicht!)

 **Set the gravity (Setze die Anziehungskraft)**
Mit dieser Aktion kannst du die Anziehungskraft eines bestimmten Objektes erstellen. Du gibst die Richtung (Winkel zwischen 0 und 360 Grad) und die Geschwindigkeit ein, und in jedem Schritt wird die Anzahl der Geschwindigkeit in der gegebenen Richtung zu Bewegung der Objektinstanz hinzugefügt. Normal benötigst du eine kleine Geschwindigkeitssteigung (wie 0.01). Eine Bewegung nach unten ist typisch (270 Grad). Wenn du die Relative Box aktivierst, erhöhst
du die Gewschindigkeit der Anziehungskraft und die Richtung. Bedenke, dass entgegen dem wirklichen Leben, verschiedene Objekte verschiedene Richtungen der Anziehungskräfte haben können.

 **Set the friction (Setze die Reibung)**
Reibung verlangsamt die Instanz bei der Bewegung. Du gibst die Menge der Reibung ein. In jedem Schritt wird diese Menge von der Geschwindigkeit abgezogen, bis die Geschwindigkeit bei 0 ist. Normal brauchst du hier eine kleine Anzahl (wie 0.01).

 **Jump to a given position (Springe zu der angegebenen Position)**
Mit dieser Aktion kannst du Instanzen an eine bestimmte Position platzieren. Du gibst einfach die x- und y-Koordinate an, und die Instanz wird mit seinem Referenzpunkt an die Position gesetzt. Wenn du die Relativ Box aktivierst, ist die Position relativ zur aktuellen Position der Instanz. Diese Aktion wird oft benutzt um eine Instanz kontinuierlich zu bewegen. In jedem Schritt wird die Position ein bisschen gesteigert.

 **Jump to the start position (Springe zur Startposition)**
Diese Aktion platziert die Instanz zurück zur Position wo sie erstellt wurde.

 **Jump to a random position (Springe zu einer zufälligen Position)**
Diese Aktion bewegt die Instanz auf eine zufällige Position im Raum. Es werden nur Positionen gewählt, an denen die Instanz keine solide Instanz überschneidet. Du kannst angeben, ob das Einrasten genützt wird. Wenn du positive Zahlen wählst, werden die gewählten Koordinaten mit ganzen Zahlen des gewählten Wertes multipliziert. Das kann benutzt werden, um Instanzen an einem Raster auszurichten. Du kannst den horizontalen und vertikalen Gitterabstand getrennt eingeben.

 **Snap to grid (Im Netz einrasten)**
Mit dieser Aktion kannst du die Position der Instanz an einem Raster ausrichten. Damit wird die Instanz in der nähesten Zelle eingerastet. Du kannst den horizontalen und vertikalen Gitterabstand getrennt eingeben. Das kann sehr nützlich sein.

 **Reverse horizontal direction (Horizontale Richtung umkehren)**
Mit dieser Aktion kannst du die horizontale Bewegung der Instanz umkehren. Verwende diese Aktion um z. B. ein Objekt welches mit einer Wand kollidiert umzukehren.

 **Reverse vertical direction (Vertikale Richtung umkehren)**
Mit dieser Aktion kannst du die vertikale Bewegung der Instanz umkehren. Verwende diese Aktion um z. B. ein Objekt welches mit einer Wand kollidiert umzukehren.

 **Move to contact position (Zur Position des Kontaktes bewegen)**
Mit dieser Aktion kannst du die Instanz in eine gegebene Richtung bewegen, bis das Objekt eine Kontaktposition erreicht. Wenn bereits eine Kollision an der aktuellen Position, wird die Instanz nicht bewegt. Sonst wird die Instanz kurz bevor die Kollision passiert gesetzt. Du kannst die Richtung angeben, aber auch eine maximale Distanz bis sie mit einem Objekt zusammenstösst. Beispielsweise, wenn die Instanz fällt, kannst du die maximale Distanz nach unten bewegen, bis das Objekt erreicht ist. Du kannst auch angeben ob nur solide Objekte oder alle Objekte berücksichtigt werden. Typischerweise wird diese Aktion in collisions event eingegeben um sicherzugehen dass das Objekt hält wenn es mit einer anderen Instanz in einer Kollision verwickelt ist.

 **Bounce against objects (Gegen Objekte prallen)**
Wenn du diese Aktion im collision event mit irgendeinem Objekt gibst, prallt die Instanz von diesem Objekt in einem natürlichen Weg zurück. Wenn du den Parameter precise auf false setzt werden nur horizontale und vertikale Wände werden korrekt behandelt. Wenn du presice auf true setzt werden auch schräge (und auch kurvige) Wände zufriedenstellend behandelt. Das ist aber langsamer. Ausserdem kannst du angeben, ob nur gegen solide oder gegen alle Objekte abgeprallt wird. Bitte bedenke dass das Abprallen nicht ganz korrekt ist, da es von vielen Eigenschaften abhängt. Aber in den meisten Situationen ist der Effekt gut genug.

12.2 Main actions, set 1 (Hauptaktionen, Set 1)

Die folgenden Aktionen behandeln das Erstellen, Ändern und Zerstören von Instanzen der Objekte und benutzen von Sounds und Räumen.

 **Create an instance of an object (Erstelle eine Instanz eines Objektes)**
Mit dieser Aktion kannst du eine Instanz eines Objektes erstellen. Du gibst an welches Objekt erstellt werden soll und die Position der neuen Instanz. Wenn du die Relative Box aktivierst wird die Position relativ zur Position der aktuellen Instanz gesetzt. Das Erstellen von Instanzen während eines Spieles ist extrem nützlich. Ein Raumschiff kann Schüsse erzeugen; eine Bombe kann Explosionen erstellen, usw. In vielen Spielen hast du ein Kontrollerobjekt, dass von Zeit zu Zeit Monster oder andere Objekte erstellt. Für die neu erstelle Instanz wird der creation event ausgeführt.

 **Create an instance of an object with a speed and direction (Erstelle eine Instanz eines Objektes mit einer bestimmten Geschwindigkeit und Richtung)**
Diese Aktion funktioniert wie die Obere, hat aber noch 2 zusätzliche Felder. Du kannst auch die Geschwindigkeit und Richtung der neu erstellten Instanz angeben. Beachte das bei markiertem „relative“ Kästchen nur die Position, nicht die Richtung und Geschwindigkeit relativ ist. Um beispielsweise eine Kugel in die Schussrichtung der Person fliegen zu lassen muss ein kleiner Trick angewandt werden. Als Position gebe 0,0 an und markiere das „Relative“ Kästchen. Als Richtung benötigen wir die aktuelle Richtung der Instanz. Dies kann durch das Wort `direction` erreicht werden. (Dies ist eine Variable die immer die Richtung der aktuellen Instanz angibt.)

 **Change the instance (Ändere die Instanz)**
Mit dieser Aktion kannst du die aktuelle Instanz in eine Instanz eines anderen Objektes ändern. Du kannst z. B. die Instanz einer Bombe in eine Explosion ändern. Alle Einstellungen, wie die Bewegung und die Werte der Variablen, bleiben unverändert. Du kannst angeben ob der Destroy Event für das aktuelle Objekte und der creation Event für das neue Objekt ausgeführt wird.

 **Change the sprite (Ändere das Sprite)**
Benutze diese Aktion um das Sprite einer Instanz zu ändern. Du gibst das neue Sprite an. Du kannst auch einen Skalierungsfaktor angeben. Ein Faktor von 1 bedeutet, dass das Sprite nicht skaliert wird. Der Skalierungsfaktor muss grösser als 0 sein. Bitte bedenke dass das Skalieren des Sprites das Zeichnen verlangsamt. Das Ändern eines Sprites ist ein wichtiges Feature. Öfters willst du den Sprite des Charakters ändern, abhäging in welche Richtung der Charakter geht.
Erstelle verschiedene Sprites für alle (vier) Richtungen. In den Keyboard Events der Richtungstasten setzt du die Richtung der Bewegung und das Sprite.

 **Destroy the instance (Zerstöre die Instanz)**
Mit dieser Aktion zerstörst du die aktuelle Instanz. Der Destroy Event wird für die Instanz ausgeführt.

 **Destroy instances at a position (Zerstöre Instanzen an einer Position)**
Mit dieser Aktion vernichtest du alle Instanzen, deren "bounding box" die angegebene Position enthält. Das ist z. b. nützlich wenn eine Bombe explodiert. Wenn du die Relative Box aktivierst wird die Position relativ zur aktuellen Position genommen.

 **Play a sound (Spiele einen Sound)**
Mit dieser Aktion kannst du eine Soundressource, die du deinem Spiel hinzugefügt hast, abspielen. Du kannst den gewünschten Sound angeben und ob er einmal (voreingestellt) oder in einer Schleife abgespielt wird. Mehrere Wave-Sounds können gleichzeitig abgespielt werden, aber es kann nur eine Midi-Datei zur gleichen Zeit gespielt werden. Also wenn du einen Midi-Sound startest, wird der aktuelle Midi-Sound gestoppt. Sofern der Sound keine mehrfachen Puffer hat (siehe Kapitel 17) kann nur eine Instanz jedes Sounds gespielt werden. Also wenn der gleiche Sound bereits gespielt wird, wird er gestoppt und neu gestartet.

 **Stop a sound (Einen Sound stoppen)**
Diese Aktion stoppt den angegebenen Sound. Wenn mehrere Instanzen diesen Sound spielen, werden alle gestoppt.

 **If a sound is playing (Wenn der Sound gespielt wird)**
Wenn der angegebene Sound gespielt wird, wird die nächste Aktion ausgeführt. Anderenfalls wird sie übersprungen. Du kannst "Not" auswählen um anzugeben dass die nächste Aktion ausgeführt wird, wenn der Sound nicht abgespielt wird. Beispielsweise kannst du prüfen, ob Hintergrundmusik abgespielt wird, wenn nicht starte eine Hintergrundmusik. Für mehr Information über Aktionen die bestimmte Bedingungen testen.

 **Go to previous room (Gehe zum vorherigen Raum)**
Geht zum vorherigen Raum. Du kannst den Typ des Übergangseffektes zwischen den Räumen auswählen. Am besten probierst du herum, um den schönsten für dich zu finden. Wenn du im ersten Raum bist, tritt ein Fehler auf.

 **Go to next room (Gehe zum nächsten Raum)**
Geht zum nächsten Raum. Du kannst den Typ Übergangseffekt angeben.

 **Restart the current room (Starte den aktuellen Raum neu)**
Der aktuelle Raum wird neu gestartet. Du kannst den Typ des Übergangseffektes angeben.

 **Go to a different room (Gehe zu einem verschiedenen Raum)**
Mit dieser Aktion kannst du zu einem bestimmten Raum gehen. Gibt den Raum und den Typ des Übergangseffektes an.

 **If previous room exists (Wenn der vorherige Raum existiert)**
Diese Aktion testet ob der vorherige Raum vorhanden ist. Wenn vorhanden, wird die nächste Aktion ausgeführt. Normal wird dies benutzt, bevor du zum vorherigen Raum wechselst.

 **If next room exists (Wenn der nächste Raum existiert)**
Diese Aktion testet ob der nächste Raum vorhanden ist. Wenn vorhanden, wird die nächste Aktion ausgeführt. Normal wird dies benutzt, bevor du zum nächsten Raum wechselst.

12.3 Main actions, set 2 (Hauptaktionen, Set 2)

Hier sind mehr Hauptaktionen. Sie behandeln Timing, Nachrichten an den Benutzer und den Umgang mit dem Spiel als Ganzen.

 **Set an alarm clock (Setze die Alarm-Uhr)**
Mit dieser Aktion kannst du acht Alarmuhren für die Instanz setzen. Du gibst die Anzahl der Schritte und die Alarmuhr an. Wenn die angegeben Anzahl der Schritte erreicht ist, bekommt die Instanz das entsprechende Alarm Event. Du kannst auch durch ankreuzen der Relative Box den Wert erhöhen oder vermindern. Wenn du eine Alarmuhr auf einen Wert kleiner als 0 setzt, schaltet du sie ab und es wird kein Event wird erstellt.

 **Sleep for a while (Schlafe für eine Zeit)**
Mit dieser Aktion kannst du das Spiel eine gewisse Zeit einfrieren. Das wird typischerweise beim Start oder Ende eines Levels verwendet oder wenn du dem Spieler eine Nachricht gibst. Du gibst die Millisekunden zum Schlafen an. Ausserdem kannst du angeben, ob der Bildschirm vorher neu gezeichnet werden soll, um die neuste Situation wiederzugeben.

 **Set a time line (Setze eine Zeitleiste)**
(Nur verfügbar im Expertenmodus!)
Mit dieser Aktion kannst du eine bestimmte Zeitleiste für eine Instanz eines Objektes setzen. Du gibst die Zeitleiste und die Startposition in der Zeitleiste
(0 ist der Anfang) an. Du kannst diese Aktion auch verwenden, um eine Zeitleiste zu beenden, in diesem Fall wähle No Time Line als Wert.

 **Set the time line position (Setze die Zeitleisten-Position)**
(Nur verfügbar im Expertenmodus!)
Mit dieser Aktion kannst du die Position in der aktuellen Zeitleiste ändern (entweder absolut oder relativ). Dies kann benutzt werden um bestimmte Teile
einer Zeitleiste zu überspringen oder gewisse Teile zu wiederholen. Beispielsweise wenn du eine Zeitleiste in einer Schleife laufen lassen willst, musst du im letzten Moment diese Aktion einfügen um die Position auf 0 zurückzusetzen. Du kannst sie auch einsetzen um auf etwas zu warten. Füge einfach eine Testaktion ein und wenn nicht "wahr" setze die Zeitleisten-Position relativ auf - 1.

 **Display a message (Zeige eine Nachricht)**
Mit dieser Aktion kannst du eine Nachricht in einer Dialogbox anzeigen lassen. Du gibst einfach die Nachricht ein. Wenn du ein # Symbol im Nachrichtentext eingibst, wird es als neue Zeile interpretiert. (Benutze \# um das # Symbol zu erhalten.) Wenn der Nachrichtentext mit einem Anführungszeichen beginnt, wird es als Ausdruck interpretiert. Siehe Unten für nähere Informationen über Ausdrücke. (Bedenke, dass diese Aktion nicht im Exklusiven Modus arbeitet, siehe Kapitel 27.)

 **Show the game information (Zeige die Spielinformation)**
Mit dieser Aktion öffnet sich das Spielinformationsfenster. Siehe Kapitel 26 für nähere Informationen wie die Spielinformation erstellt wird. (Diese Aktion arbeitet nicht im Exklusiven Modus.)

 **Show a video (Zeige ein Video)**
Mit dieser Aktion kannst du eine Video-/Filmdatei abspielen. Du musst den Dateinamen angeben und wählen ob es in einem Fenster oder als Vollbild gezeigt werden soll. Stelle sicher das die Datei existiert. Du solltest es entweder mit dem Spiel ausliefern oder als Datafile exportieren.

 **Restart the game (Neustarten des Spieles)**
Mit dieser Aktion startest du das Spiel wieder von Anfang an.

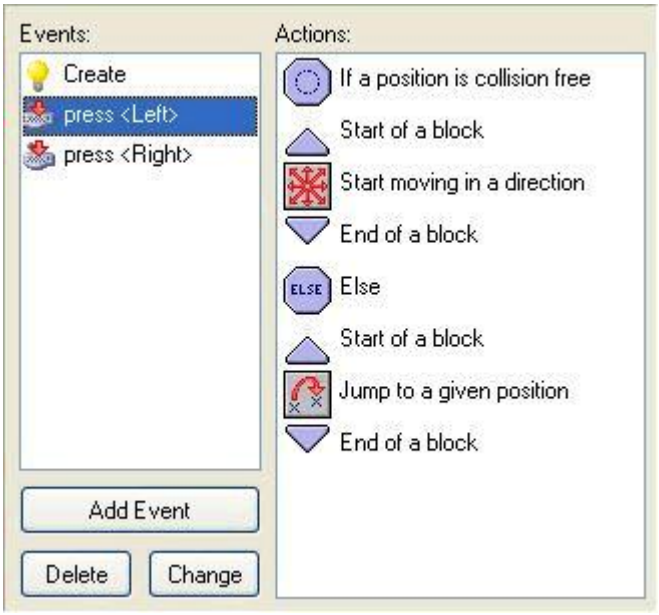
 **End the game (Beende das Spiel)**
Mit dieser Aktion beendest du das Spiel.

 **Save the game (Speichere das Spiel)**
Mit dieser Aktion kannst du den aktuellen Spielstatus speichern. Du gibst den Dateinamen für das Speichern an (die Datei wird im Arbeitsverzeichnis des Spieles erstellt). Später kann das Spiel mit der folgenden Aktion geladen werden.

 **Load the game (Lade ein Spiel)**
Lädt den Spielstatus aus einer Datei. Du gibst den Dateinamen an. Gehe sicher dass dieses gespeicherte Spiel für das gleiche Spiel ist und mit der gleichen Version vom Game Maker erstellt wurde. Sonst gibt es eine Fehlermeldung. (Um genau zu sein, das Spiel wird am Ende des aktuellen Schrittes geladen. So werden einige Aktionen nach diesem noch ausgeführt im aktuellen Spiel, aber nicht das Laden!)

12.4 Control (Kontrolle)

Es gibt eine Anzahl von Aktion mit welchen du kontrollieren kannst welche andere Aktionen ausgeführt wurden. Die meisten dieser Aktionen sind Fragen, z. B. ob eine Position leer ist. Wenn die Antwort ja (true) ist, wird die nächste Aktion ausgeführt, sonst wird sie übersprungen. Wenn du mehrere Aktionen ausführen oder überspringen willst, kannst du sie in einen Start und End Block integrieren. Sie können auch Teil eines Else Teiles sein welcher ausgeführt wird wenn die Antwort no ist. So eine typische Frage sieht so aus:



Hier wird gefragt ob die Position der aktuellen Instanz kollissionsfrei ist. Wenn dies so ist beginnt die Instanz sich in die angegebene Richtung zu bewegen. Wenn nicht springt die Instanz zu einer angegebenen Position. Für alle Frage gibt es ein Feld "Not". Wenn du es aktivierst, wird das Resultat der Frage umgedreht. Also, wenn das Resultat wahr ist wird es falsch und wenn es falsch war wird es wahr. Das erlaubt eine bestimmte Aktionen auszuführen, wenn die Antwort nicht wahr ist.
Für viele Fragen kannst du angeben, ob sie für alle Instanzen eines bestimmten Objektes gelten. In diesem Fall ist das Resultat nur wahr wenn es für alle Instanzen des Objektes wahr ist. Beispielsweise kannst du prüfen, ob alle Bälle rechts neben der Position kollissionsfrei sind. Die folgenden Fragen und verwandten Aktionen sind verfügbar. (Bedenke dass sie alle eine andere Form des Icons und einen andere Hintergrundfarbe haben um sie leicht von anderen Aktionen zu unterscheiden zu lassen.)

If a position is collision free (Wenn die Position kollisionsfrei ist)
Diese Frage gibt true zurück, wenn die aktuelle Instanz an der angegeben Position keine Kollision mit einem Objekt verursacht. Du kannst die Position entweder absolut oder relativ angeben. Du kannst auch angeben, ob nur solide Objekte oder alle Objekte in Betracht gezogen werden.
Diese Aktion wird meistens verwendet, um zu prüfen ob die Instanz an eine bestimmte Position verschoben werden kann.

If there is a collision at a position (Wenn eine Kollision an einer Position ist)
Das ist die Umkehr der vorhergehenden Aktion. Es gibt true zurück wenn eine Kollision stattfindet, wenn die aktuelle Instanz an die gegebene Position gesetzt wird (wieder, entweder nur solide oder alle Objekte).

If there is an object at a position (Wenn ein Objekt an einer Position ist)
Diese Frage gibt true zurück, wenn die Instanz an der angegebenen Position eine Instanz des angegebenen Objektes trifft.

 **If the number of instances is a value (Wenn die Anzahl der Instanzen den Wert hat)**
Du gibst das Objekt und die Zahl an. Wenn die aktuelle Anzahl der Instanzen des Objektes gleich mit den Zahl ist gibt die Frage true zurück, sonst false. Du kannst angeben, ob überprüft werden soll ob die Anzahl der Instanzen größer oder kleiner der angegebenen Zahl ist.
Häufig wird so eine Frage am Ende eines Levels oder Spieles gestellt.

 **If a dice lands on one (Wahrscheinlichkeit mit der eine Zahl kommt)**
Du kannst eine Zahl angeben. Es ist wie bei einem Würfel (Wahrscheinlichkeit beim Würfel 1 : 6, dass die Zahl gewürfelt wird). Wenn die Zahl gewürfelt wird ist das Resultat ist wahr und die nächste Aktion wird ausgeführt. Das kann verwendet werden, um einen Zufallsmechanismus ins Spiel einzubauen. Beispielsweise, kannst du in jedem Schritt eine bestimmte Chance einbauen, dass die Bombe explodiert oder die Richtung ändert. Je grösser die Zahl, desto geringer die Wahrscheinlichkeit. Du kannst auch reele Zahlen verwenden. Beispielsweise wenn du die Zahl auf 1.5 setzt, wird die Aktion in zwei von drei Fällen ausgeführt. Ein Zahl unter 1 macht keinen Sinn.

 **If the user answers yes to a question (Wenn der Benutzer auf eine Frage ja antwortet)**
Du stellst eine Frage. Eine Dialog wird dem Spieler mit einem yes und einem no Button gezeigt. Das Resultat ist true, wenn der Spieler yes antwortet. Diese Aktion kann nicht im Exklusiven Modus benutzt werden, die Antwort wird dann immer ja sein.

 **If an expression is true (Wenn ein Ausdruck wahr ist)**
Das ist die allgemeinste Frage-Aktion. Du kannst einen willkürlichen Ausdruck eingeben. Wenn der Ausdruck wahr ist, wird die nächste Aktion ausgeführt. Siehe unten für nähere Informationen über Ausdrücke.

 **If a mouse button is pressed (Wenn die Maustraste gedrückt ist)**
Gibt true zurück, wenn die angegebene Maustaste gedrückt wird. Wird normalerweise im step event benutzt. Du kannst angeben, ob eine Maustaste gedrückt ist, und so z. B. die Position ändern (benutze den jump to a point Aktion mit den Werten mouse_x und mouse_y).

 **If instance is aligned with grid (Wenn die Instanz im Gitter eingerastet ist)**
Gibt wahr zurück, wenn die Position der Instanz im Gitter liegt. Du kannst die horizontale und vertikale Grösse des Gitters angeben. Das ist nützlich, wenn bestimmte Aktionen, wie eine Drehung machen, nur erlaubt sind wenn die Instanz an einer bestimmten Gitterposition ist.

 **Else (Sonst)**
Nach dieser Aktion folgt der else Teil, der ausgeführt wird wenn das Resultat der Frage false ist.

 **Start of block (Beginn des Blockes)**
Gibt den Beginn des Blockes von Aktionen an.

 **End of block (Ende des Blockes)**
Gibt das Ende des Blockes von Aktionen an.

 **Repeat next action (Wiederhole die nächste Aktion)**

Diese Aktion wird benutzt, um die nächste Aktion (oder Block von Aktionen) einige Male wiederholen zu lassen. Du gibst einfach die Anzahl der Wiederholungen an.

 **Exit the current event (Beenden des aktuellen Event)**
Wenn diese Aktion zutrifft, wird keine Aktion im Event mehr ausgeführt. Typischerweise benutzt du das nach einer Frage. Beispielsweise, wenn eine Position frei ist wird nichts mehr zu tun sein und der Event wird beendet. In diesem Beispiel werden die folgenden Aktion nur ausgeführt, wenn eine Kollision stattfindet.

12.5 Drawing actions (Zeichenaktionen)

Zeichenaktionen machen nur Sinn im Drawing Event. An anderen Stellen werden sie einfach ignoriert. Erinnere dich, dass das Zeichnen andere Dinge als Sprites und Hintergrundbilder sehr langsam ist. Also benutze dies nur wenn es wirklich notwendig ist.

 **Draw a sprite image (Zeichne ein Sprite-Bild)**
Du gibst das Sprite, die Position (entweder absolut oder relativ zur aktuellen Position) und das Einzelbild des Sprites an. (Das Einzelbild ist ein Nummer von 0 aufwärts). Wenn du das aktuelle Einzelbild zeichnen willst, benutze –1.

 **Draw a background image (Zeichne ein Hintergrund-Bild)**
Du gibst das Hintergrund-Bild, die Position (absolut oder relativ) und ob das Bild über den ganzen Raum geteilt werden soll oder nicht.

 **Draw a rectangle (Zeichne ein Rechteck)**
Du gibst die Koordinaten der zwei gegenüberliegenden Ecken des Rechteckes an, absolut oder relativ zur aktuellen Position.

 **Draw an ellipse (Zeichne eine Ellipse)**
Du gibst die Koordinaten der zwei gegenüberliegenden Ecken des umliegenden Rechteckes an, absolut oder relativ zur aktuellen Position.
 **Draw a line (Zeichne eine Linie)**
Du gibst die Koordinaten der zwei Endpunkte der Linie an; absolut oder relativ zur aktuellen Position.

 **Draw a text (Zeichne einen Text)**
Du gibst den Text und die Position an. Ein # Symbol im Text wird als neue Zeile interpretiert. (Benutze \# um das # Symbol selber zu erhalten.) So kannst du mehrzeilige Texte erstellen. Wenn der Text mit einem Anführungszeichen beginnt, wird es als Ausdruck interpretiert. Beispielsweise kannst du

```
'X: ' + string(x)
```

benutzen um den Wert der x-Koordinate der Instanz anzuzeigen. (Die Variable x speichert die aktuelle x-Koordinate. Die Funktion string() wandelt die Zahl in eine Zeichenkette um. + kombiniert die zwei Zeichenketten.)
 **Set the colors (Setze die Farbe)**

Damit wird die Farbe gesetzt, sie wird benutzt um Rechtecke und Ellipsen zu füllen und für die Linien um das Rechteck und die Ellipse und wenn eine Linie gezeichnet wird.

 **Set a font for drawing text (Setze die Schriftart des gezeichneten Textes)**
Du kannst die Schriftart des Textes setzen, diese wird ab diesen Moment verwendet.

 **Change fullscreen mode (Ändere Vollbild-Modus)**
Mit dieser Aktion kannst du den Bildschirm-Modus von Fenster- zu Vollbildmodus und zurück verändern. Du gibst an, ob der Modus umgeschaltet wird, oder ob du zum Fenster- oder Vollbildschirmmodus gehst. (Das funktioniert nicht im Exklusiven Modus.)

12.6 Score actions (Punkteaktionen)

In den meisten Spieler hat der Spieler einen bestimmte Punktestand. Auch geben viele Spiele dem Spieler eine Anzahl von Leben. Zuletzt haben Spieler eine bestimmte Gesundheit. Die folgenden Aktion machen es einfach mit Punkten. Leben und Gesundheit des Spielers zu arbeiten.

 **Set the score (Setze den Punktestand)**
Game Maker hat einen eingebauten Punktemechanismus. Der Punktestand wird normal in der Titelleiste gezeigt. Du kannst diese Aktion verwenden um den Punktestand zu verändern. Oft willst du etwas zum Punktestand hinzufügen, in diesem Fall aktiviere die Relative Box.

 **If score has a value (Wenn der Punktestand einen Wert hat)**
Mit dieser Frage-Aktion kannst do prüfen ob der Punktestand einen gewissen Wert erreicht hat. Du gibst den Wert an und ob der Punktestand gleich der Wert, kleiner dem Wert oder grösser dem Wert sein soll.

 **Draw the value of score (Zeichne den Punktestand)**
Mit dieser Aktion kannst du den Wert des Punktestandes an einer bestimmten Position am Bildschirm zeichnen. Du gibst die Position und die Überschrift an, welche vor dem Punktestand angegeben wird. Der Punktestand wird in der aktuellen Schriftart gezeichnet. Diese Aktion kann nur im drawing event des Objektes genutzt werden.

 **Clear the highscore table (Lösche die Höchstpunktetabelle)**
Diese Aktion löscht die Highscore-Tabelle.

 **Display the highscore table (Zeige die Höchtspunktetabelle)**
Für jedes Spiel werden die zehn besten Punktestände aufbewahrt. Diese Aktion zeigt die Highscore-Liste. Wenn der aktuelle Punktstand unter den besten Zehn ist, wird der neue Punktestand eingefügt und der Spieler kann seinen Namen eingeben. Daher solltest du nicht den Punktestand mit der vorherigen Aktion hinzufügen. Du kannst das Hintergrundbild angeben, ob das Fenster einen Rand haben soll, die Farbe des neuen Eintrages und wie die alten Einträge sein sollen und welche Schriftart verwendet wird. (Diese Aktion arbeitet nicht im Exklusiven Modus!)

 **Set the number of lives (Setzte die Anzahl der Leben)**
Game Maker hat auch ein eingebautes Leben-System. Mit dieser Aktion kannst du die Nummer der übriggeblieben Leben ändern. Normal setzt du den Wert beim Start auf eine Wert wie 3 und dann erhöhst oder senkst du die Anzahl der Leben, je nachdem was im Spiel geschieht. Vergiss nicht Relative Box, wenn du Leben hinzufügen oder abziehen willst. Wenn die Anzahl der Leben 0 ist,

(oder kleiner als 0) wird ein "no more lives" event erstellt.

 **If lives is a value (Wenn das Leben einen Wert hat)**
Mit dieser Frage-Aktion kannst du prüfen ob die Anzahl der Leben einen gewissen Wert erreicht hat. Du gibst den Wert an, und ob die Anzahl der Leben gleich dem Wert, kleiner dem Wert oder grösser dem Wert sein soll.

 **Draw the number of lives (Zeichne die Anzahl der Leben)**
Mit dieser Aktion kannst du die Anzahl der Leben an einer bestimmten Position am Bildschirm zeichnen. Du gibst die Position und die Überschrift an, welche vor der Anzahl der Leben angegeben wird. Der Punktestand wird in der aktuellen Schriftart gezeichnet. Diese Aktion kann nur im drawing event des Objektes genutzt werden.

 **Draw the lives as image (Zeichne die Leben als Bild)**
Statt die Anzahl der Leben als Zahlen zu zeichnen, ist es meisten schöner eine Anzahl von kleinen Bilder dafür zu benutzen. Diese Aktion macht genau das. Du gibst die Position und das Bild an und an der angegebenen Position werden die Anzahl der Leben als Bilder gezeichnet. Diese Aktion kann nur im drawing event eines Objektes verwendet werden.

 **Set the health (Setze die Gesundheit)**
Game Maker hat auch ein eingebautes Gesundheits-System. Mit dieser Aktion kannst du die Gesundheit ändern. Ein Wert von 100 bedeutet volle Gesundheit und ein Wert von 0 keine Gesundheit. Du gibst einfach den neuen Wert der Gesundheit ein. Oft willst du einen Wert von der Gesundheit abziehen. In diesem Fall vergiss nicht die Relative Box zu aktivieren. Wenn die Gesundheit kleiner oder gleich 0 ist wird ein „out of health“ Event erstellt.

 **If health is a value (Wenn die Gesundheit einen Wert hat)**
Mit dieser Frage-Aktion kannst du prüfen ob die Gesundheit einen gewissen Wert erreicht hat. Du gibst den Wert an und ob die Gesundheit gleich den Wert, kleiner dem Wert oder grösser dem Wert sein soll.

 **Draw the health bar (Zeichne den Gesundheitsbalken)**
Mit dieser Aktion kannst du die Gesundheit in Form eines Gesundheitsbalken. Wenn die Gesundheit 100 ist, wird ein voller Balken gezeichnet. Du gibst die Position und die Grösse der Gesundheitsbalken an, die Farbe des Balkens und den Hintergrund.

 **Set the window caption information (Setze die Titelleisten-Information)**
Normalerweise wird in der Titelleiste der Name des Raumes und der Punkterstand angezeigt. Mit dieser Aktion kannst du das ändern. Du kannst angeben, ob der Punktestand, die Leben und / oder die Gesundheit gezeigt werden soll oder nicht und wie die Beschriftung für jeden davon lauten soll.

12.7 Code related actions (Codebezogene Aktionen)

Zuletzt gibt es einige Aktionen die primär mit Code arbeiten.

 **Execute a script (Ein Skript ausführen)**
(Nur im Expertenmodus vorhanden.)
Mit dieser Aktion kannst du einen Skript ausführen, dass du deinem Spiel hinzugefügst hast. Du

gibst den Skript und maximal 5 Argumente an. Siehe Kapitel 24 für nähere Informatione über Skripte.

 **Execute a piece of code (Ein Stück Code ausführen)**
Wenn du dieser Aktion hinzufügst, wird ein Fenster geöffnet wo du ein Stück Code ausführen kannst. Dies arbeit im gleichen Weg wie das definieren von Skripten (siehe Kapitel 24). Der einzige Unterschied ist das du angeben kannst, für welche Instanz der Stück Code ausgeführt wird. Benutze die Code Aktion für einen kleinen Teil Code. Für längere Stücke werden Skripte empfohlen.

 **Set the value of a variable (Setzte die Variable)**
Es gibt viele eingebaute Variablen im Spiel. Mit dieser Aktion kannst du diese verändern. Ausserdem kannst du deine eigenen Variablen erstellen und ihnen einen Wert zuteilen. Du gibst den namen der Variablen und den neuen Wert an. Wenn du die Relative Box aktivierst, wird der Wert zum aktuellen Wert der Variable hinzugezählt. Bitte bedenke dass das nur möglich ist, wenn die Variable ein Wert zugeteilt wurde. Siehe unten für nähere Informationen.

 **If a variable has a value (Wenn die Variable einen Wert hat)**
Mit dieser Aktion kannst du überprüfen was der Wert einer bestimmten Variable ist. Wenn der Wert der Variablen gleich der angegeben Nummer ist, gibt die Frage true zurück. Andererseit gibt sie false zurück. Du kannst auch angegeben, ob überprüft werden soll ob die der Wert kleiner als der angegeben Wert oder grösser als der angegebenen Wert ist. Siehe unten für nährere Informationen über Variablen. Du kannst diese Aktion auch für den Vergleich von zwei Ausdrücke verwenden.

 **Draw the value of a variable (Zeichne den Wert einer Variable)**
Mit diese Aktion kannst du den Wert einer Variable an einer bestimmten Position am Bildschirm zeichnen.

 **Call the inherited event (Rufe den erbenden Event auf)**
(Nur im Expertenmodus vorhanden.)
Diese Aktion ist nur nützlich wenn das Objekt ein Elternobjekt besitzt (siehe Kapitel 19). Es ruft den entsprechen Event im Elternobjekt auf.

 **Comment (Kommentar)**
Benutze diese Aktion um eine Kommentarzeile in deiner Aktionliste hinzuzufügen. Die Zeile wird in der Schriftart Italic gezeigt. Es hat keine Funktion. Das Hinzufügen von Kommentaren hilft dir dich zu erinnern was im Event geschieht.

12.8 Ausdrücke und Variablen benutzen

In vielen Aktionen musst du Werte für Parameter angeben. Anstatt einfach eine Zahl anzugeben kannst du auch eine Formel nehmen, z. B. 32*12. Aber du kannst auch kompliziertere Ausdrücke verwenden. Wenn du die horizontale Geschwindigkeit verdoppeln willst, kannst du sie auf 2*hspeed setzen. Hier ist hspeed eine Variable die die aktuelle horizonale Geschwindigkeit der Variable angibt. Es gibt eine Vielzahl von anderen Variablen die benutzt werden können.

Einige der wichtigsten sind:

x - die x-Koordinate der Instanz
y - die y-Koordinate der Instanz
hspeed - die horizontale Geschwindigkeit (in Pixel pro Schritt)
vspeed - die vertikale Geschwindigkeit (in Pixel pro Schritt)
direction - die aktuelle Richtung der Bewegung in Graden (0-360)
speed - die aktuelle Geschwindigkeit in dieser Richtung
visible - ist das Objekt sichtbar (1) oder unsichtbar (0)
image_scale - die Zahl mit der das Bild skaliert ist(1 = nicht skaliert)
image_single - Diese Variable gibt an welches Unterbild im aktuellen Sprite gezeigt werden muss, wenn du –1 setzt (voreingestellt) läuft das Bild in einer Schleife, ansonst wird nur das aktuelle angegebene Einzelbild gezeigt (startet mit Nummer 0)
image_speed - Diese Variable gibt die Geschwindigkeit an mit der die Einzelbilder gezeigt werden sollen. Der eingestellte Wert ist 1. Wenn du diesen Wert grösser als 1 machst, werden manche Einzelbilder überspringen und die Animation wird schneller. Wenn du den Wert kleiner als 1 machst, wird die Animation langsamer.
score - der aktuelle Wert der Punkte
lives - die aktuelle Anzahl der Leben
health - die aktuelle Gesundheit(0-100)
mouse_x - x-Position der Maus
mouse_y - y-Position der Maus

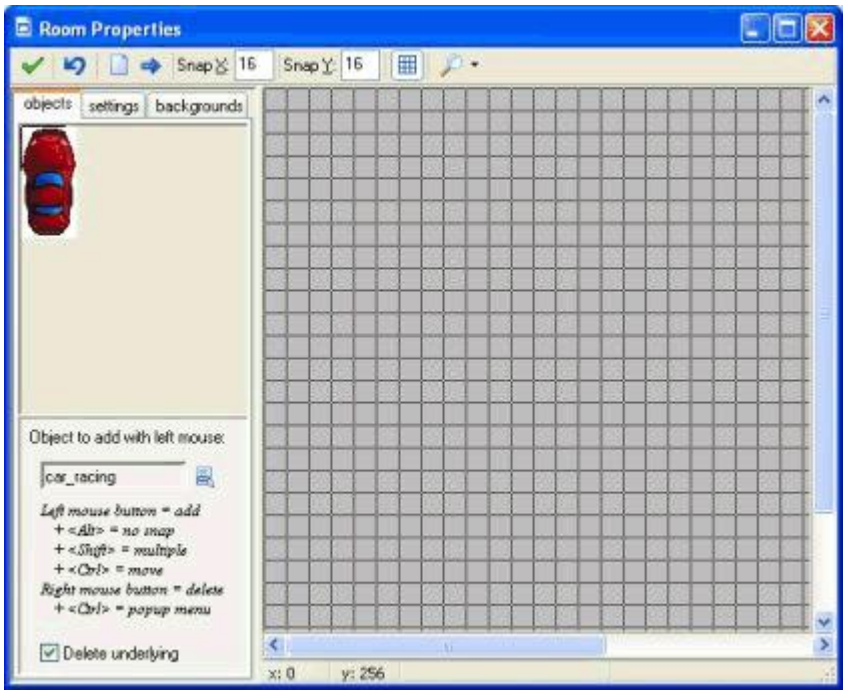
Du kannst die meisten diese Variablen durch Setzen von Aktion verändern. Du kannst auch eigene Variablen durch Setzten der Werte definieren (Benutze nicht relativ, da sie noch nicht existieren). Dann kannst du diese Variablen in Ausdrücken benutzen. Die Variablen die du erstellst sind lokal zu der aktuellen Instanz. Jedes Objekt hat eine Kopie davon. Um eine globale Variable zu erstellen, nimm das Wort global und einen Punkt vor der Variable. Du kannst auch die Wert andere Variablen referenzieren, gibt den Objektname und eine Punkt vor ihn. Beispielsweise wenn du eine Ball zu einem Platz wo die Münze ist bewegen willst, kannst du die Position (coin.x , coin.y) setzten. Im Falle eines collision event kannst du dich auf die x-Koordinate des anderen Objekt beziehen other.x. In bedingten Ausdrücken (expressions) kannst du Vergleiche wie < (grösser als), >, usw. verwenden. In deinen Ausdrücken kannst du auch Funktionen verwenden.

Die Funktion random(10) gibt eine zufällige ganze Nummer unter 10 aus. So kannst du z. B. die Geschwindigkeit oder Richtung der Bewegung einen zufälligen Wert geben. Es existieren noch viel mehr Funktionen. Für nähere Informationen über Ausdrücke und Funktionen siehe Kapitel 29 und folgende.

13. Räume erstellen

Nachdem du die Objekte mit ihren Eigenschaften in den Ereignissen und Aktionen erzeugt hast, ist es Zeit einen Raum oder Level zu erstellen, wo das Spiel stattfindet. Jedes Spiel benötigt mindestens einen Raum. In diesen Räumen platzieren wir die Instanzen der Objekte. Wenn das Spiel gestartet ist wird der erste Raum gezeigt und die Instanzen darin werden lebendig, weil ihre Aktionen im Creation Event ausgeführt werden. Es gibt eine grosse Anzahl von Möglichkeiten beim Erstellen eines Raumes. Neben setzen einiger Eigenschaften und Hinzufügen von Instanzen der Objekte kannst du einen Hintergrund hinzufügen, Views
definieren und Tiles hinzufügen.
Die meisten dieser Optionen werden später im Kapitel 20 erörtert.

In diesem Kapitel zeigen wir nur die einfachen Einstellungen, das Hinzufügen von Instanzen und die Einstellungen von Hintergrundbildern. Zum Erstellen eines Raumes, wähle Add Room vom Add Menu. Das folgende Fenster zeigt sich:



Links siehst du drei Registerkarten (fünf im Experten-Modus). Im Register "objects" erstellst du Instanzen von Objekten im Raum. Im Register settings änderst du einige Einstellungen des Raumes. Im Register backgrounds setzt du die Hintergrundbilder des Raumes.

13.1 Hinzufügen von Instanzen

Rechts im Raumerstellungsfenster siehst du den Raum. Anfangs ist er leer mit einem grauen Hintergrund.

Füge deinem Raum Instanzen hinzu indem du den Register objects wählst. Danach wähle das gewünschte Objekt durch Kicken auf den Button mit dem Menu icon (oder durch Klicken in das Bild links). Das Bild des Objektes erscheint links (Bedenke dass ein Kreuz im Bild ist. Das zeigt wie die Instanz im Raster ausgerichtet wird). Nun klicke mit der linken Maustaste in das Raumfeld rechts. Eine Instanz des Objektes erscheint. Es schnappt im angegebenen Raster ein.

Wenn die <Alt> Taste gedrückt wird, wird die Instanz nicht am Raster ausgerichtet. Mit der rechten Maustaste kannst du die Instanz entfernen. Dadurch gibst du den Inhalt des Raumes an. Wenn die Maustaste während des Verschiebens im Raum gedrückt bleibt, werden mehrere Instanzen hinzugefügt oder gelöscht.

Wie wahrscheinlich schon bemerkt, verschwindet die originale Instanz, wenn eine Instanz über ein andere platziert wird. Normalerweise wird das gewünscht, aber nicht immer.
Das kann vermieden werden durch Deaktivieren der Box Delete underlying. Es gibt drei andere Aktionen die du mit der rechte Maustaste machen kannst. Wenn du die <Strg> Taste und die rechte Maustaste drückst, wird die unterste Instanz an dieser Position nach oben gesendet. Das kann benützt werden um überlappende Instanzen zu ändern. Zuletzt löscht <Shift> Taste alle Instanzen einer Position, nicht nur das oberste.

Es gibt vier nützliche Buttons im linken Register. Wenn du den Clear Button drückst werden alle Instanzen des Raumes entfernt. Wenn du den Shift Button drückst können alle Instanzen verrückt werden. Benutze negative Zahlen um links oder nach oben zu verschieben. Das ist sehr nützlich wenn du entscheidest einen Raum zu vergrössern. (Du kannst das auch benützen um etwas ausserhalb des Raumes zu erstellen, was manchmal nützlich ist). Zuletzt gibt es zwei Buttons um die Instanzen nach der X oder Y Koordinate zu sortieren. Dies ist nützlich wenn Instanzen sich teilweise überlappen.

13.2 Einstellungen der Räume

Jeder Raum besitzt einige Einstellungen die du ändern kannst im Register settings. Wir sehen uns hier nur die wichtigsten an.

Jeder Raum hat einen Namen. Gib ihm einen aussagekräftigen Namen. Es gibt auch eine Überschrift. Die Überschrift wird im Fenster angezeigt wenn das Spiel läuft. Du kannst die Breite und Höhe (in Pixel) des Raumes setzen. Auch die Geschwindigkeit kann gesetzt werden. Das ist die Anzahl der Schritte pro Sekunde.
Je höher die Geschwindigkeit umso flüssiger die Bewegungen. Aber dazu braucht du einen schnelleren Computer.
Unter dem Register settings kannst du die Grösse des Rasters für die Objekte angeben. Durch Klicken auf den Button Show kannst du angeben ob das Netz sichtbar ist oder nicht. (Du kannst auch angeben ob der Hintergrund gezeigt wird etc.). Das ist manchmal sinnvoll um bestimmte Teile eines Raumes zu verstecken.

13.3 Den Hintergrund setzen

In dem Register backgrounds setzt du das Hintergrundbild für den Raum. Es können auch mehrere Hintergründe gesetzt werden. Das Register sieht wie folgt aus:

Oben siehst du die Hintergrundfarbe. Du kannst darauf klicken um sie zu ändern. Die Hintergrundfarbe wird nur benötigt wenn du kein Hintergrundbild, welches den ganzen Raum überdeckt, verwendest. Andererseits wäre es besser die Box Draw background color auszuschalten, weil es sonst eine Verschwendung von Speicher wäre.
Oben siehst du eine Liste von 8 Hintergründen. Du kannst jeden festlegen aber meistens benötigt man nur einen oder zwei. Um den Hintergrund festzulegen, wähle in der Liste. Danach kreuze die Box "Visible when room starts" sonst siehst du nichts. Der Name des Hintergrundes wird jetzt fettgedruckt. Nun gib ein Hintergrundbild im Menu an. Es gibt einige Einstellungsmöglichkeiten.

Zuerst kannst du angeben ob das Hintergrundbild den Raum waagrecht und/oder senkrecht teilen soll. Du kannst die Position des Hintergrundes im Raum angeben (dies beeinflusst auch das Teilen). Zuletzt kannst du den Hintergrund scrollen lassen, gib eine waagrechten oder senkrechte Geschwindigkeit an (Pixel pro Schritt).
Es gibt noch eine Auswahlbox namens Foreground image. Wenn du die Auswahlbox ankreuzt, ist der Hintergrund jetzt ein Vordergrund, welcher vorner gezeichnet wird statt hinter allen anderen. Natürlich sollte solch ein Bild teilweise transparent sein, damit dies einen Sinn ergibt.

14. Weitergeben des Spiels

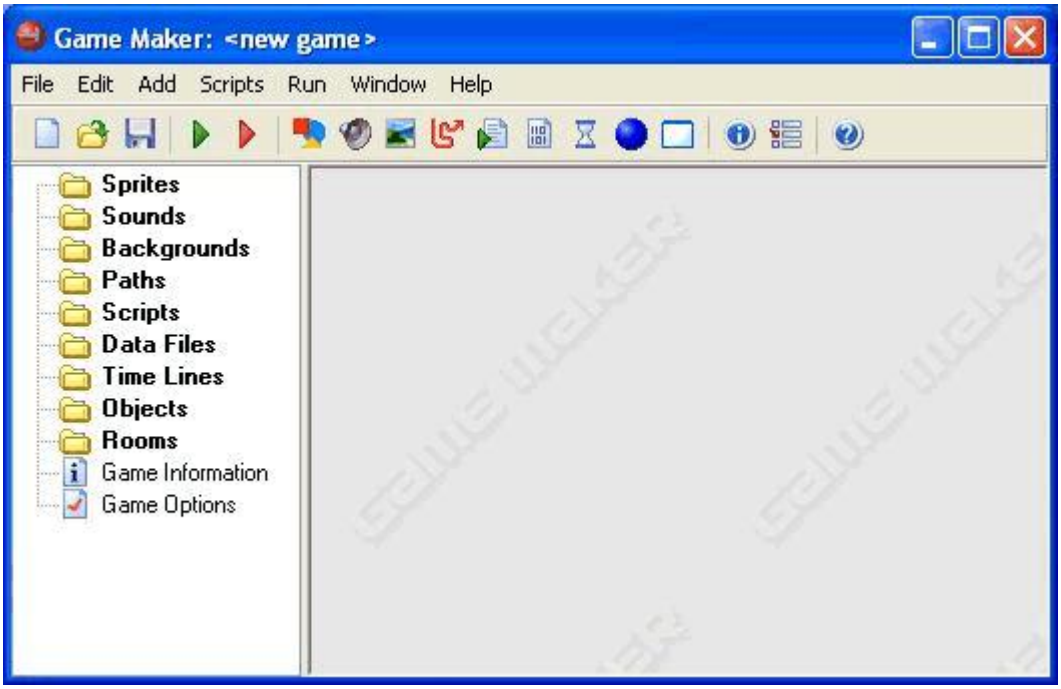
Mit den Informationen aus den vorausgegangenen Kapiteln, kannst du eigene Spiele erstellen. Hast du ein nettes Spiel erstellt, willst du es wahrscheinlich an andere weitergeben, damit sie es spielen können. Du darfst die Spiele,welche du mit Game Maker erstellt hast frei verteilen, du darfst sie sogar verkaufen.
Beachte die enthaltene "license agreement" (Lizenzbestimmung) mit weiteren Informationen. Es gibt drei Wege, wie du dein Spiel weitergeben kannst. Der einfachste ist, die *.gmd-Datei weiterzugeben, welche das Spiel enthält. Das setzt voraus, dass der Spieler auch den Game Maker besitzt. (Es ist dir nicht erlaubt,den Game Maker mitzugeben aber er kann von der Game Maker-website runtergeladen werden.).

Die anderen können dein Spiel so auch verändern. Der zweite Weg ist, eine ausführbare Datei des Spiels zu erstellen. Dies kann erreicht werden,wenn du den Menüpunkt "create executable" im "file menu" wählst. Du wirst nach dem Namen der Exe-Datei gefragt, welche dein Spiel enthalten soll. Gib einen Namen an, drück auf OK und du hast dein Exe-File zum Weitergeben. Du kannst noch ein Icon für dein Spiel in den Spieloptionen festlegen. (Wenn dein Spiel noch irgendwelche anderen Dateien braucht, solltest du sie in den Spielordner packen.) Jetzt solltest du dein Spiel weitergeben (evtl.noch zippen).

Der dritte Weg ist ein Installer-Programm zu benutzen. Eine Zahl von "freeware"-Installer-Proggies findet sich im Web (oder auf Anfrage in der Community). Wieder machst du zuerst eine Exe-Datei und verwendest diese dann mit dem Installer, um eine Installationsroutine zu erstellen. Wie das funktioniert, hängt vom verwendeten Installer ab.

15. Der erweiterte Modus

Bis jetzt betrachteten wir nur die einfachen Features des Game Maker. Aber es gibt viel mehr Möglichkeiten.
Um diese zu nutzen, müsst ihr den Game Maker in den Fortgeschrittenen-Modus umschalten. Das geht ganz einfach. Im File Menü klickt auf den Eintrag Advanced mode. (Um den Effekt zu sehen solltet ihr den Game Maker neu starten oder eurer Spiel speichern und es neu laden).
Wenn ihr Game Maker im Fortgeschrittenen-Modus startet, seht ihr folgendes Bild:



Es beinhaltet alles, was auch im Anfänger-Modus vorhanden ist, aber es gibt einige zusätzliche Ressourcen, Buttons und Menu-Einträge. Wie wir in den nächsten Kapitel sehen werden, haben die verschiedenen Ressourcen zusätzliche Optionen. Wir werden jetzt die zusätzlichen Menu-Einträge betrachten.

15.1 File menu – Das Datei Menü

Im File menu gibt es folgende zusätzliche Kommandos:

Merge Game:
Mit diesem Kommando kannst du alle Ressourcen (Sprites, Sounds, Objekte, Räume, etc.) von einem anderem Spiel in das aktuelle Spiel kopieren. Das ist sehr nützlich, wenn ihr einige Teile wiederverwenden wollt (z.B.Menüsysteme). (Bedenkt, dass alle Ressourcen, Instanzen und Tiles eine neue Id bekommen, was manchmal Probleme mit den Scripts verursacht.). Du musst selber dafür sorgen, dass die Resourcen in den zwei Dateien verschiedene Namen haben, sonst gibt es Probleme.

Preferences:
Hier kannst du den Game Maker anpassen. Diese Informationen werden für die nächsten Aufrufe des Game Maker gespeichert. Die folgenden Punkte können eingestellt werden:

Show recently edited games in the file menu:
Zeigt die letzten 8 bearbeitenden Spiele unter Recent files im File menu an..

Load last opened file on startup:
Die letzte Datei wird automatisch geöffnet.

Keep backup copies of files:
Das Programm erstellt Backupdateien mit der Dateierweiterung *.gb0-*.gb9.
Diese Dateien können mit Game Maker geöffnet werden.

Maximal number of backups:
Hier könnt ihr eingeben, (1-9) wieviele verschiedene Backups das Programm machen soll.

Show progress while loading and saving files:
Beim Speichern oder Laden einer Datei wird ein Fortschrittsbalken angezeigt.

At startup check for, and remove old temporary files.
Der *Game Maker* und dessen Spiele erzeugen temporäre Dateien. Normalerweise werden diese wieder automatisch entfernt, aber manchmal, beispielsweise beim Absturz eines Spieles, bleiben sie zurück. Wenn diese Option aktiviert ist überprüft *Game Maker* beim Start ob solche Dateien existieren und entfernt diese.

Hide the designer and wait while the game is running:
Hiermit wird das Hauptprogramm solange ausgeblendet, bis das Spiel beendet worden ist.

Run games in secure mode:
In diesem Modus erlaubt Game Maker es nicht, externe Programme auszuführen; genauso wie das Verändern oder Löschen von Dateien , die woanders liegen, als die des Spieles. (Das ist eine Sicherheitsmassnahme gegenüber Trojaner,obwohl der Erfolg nicht garantiert ist!). Manche Spiele können dadurch nicht korrekt funktionieren. Dieses Einstellung funktioniert nur, wenn der Game Maker läuft.
Wenn du also ein Spiel unabhängig vom Game Maker startest, läuft dieses nicht im Secure mode.

Show the origin and bounding box in the sprite image:
Zeigt den Bezugspunkt (origin) und den Begrenzungsrahmen (bounding box) eines Sprites an.

In object properties, show hints for actions:
Zeigt eine Beschreibung der Action im Objektfenster an, wenn die Maus länger darauf bleibt.

When closing, remove instances outside the room:
Das Programm warnt vor Objekten, die ausserhalb des Raumes liegen und lässt diese löschen.

Remember room settings when closing the form:
Einige Raum-Einstellungen, wie das Anzeigen des Grid (Gittters), das Löschen übereinanderliegender Objekte, etc, kann hier gespeichert werden.

External sound editors:
Hier kann man einen externen Soundeditor für die verschiedenen Sounddateien einstellen. (Game Maker besitzt keinen eingebauten Soundeditor, deshalb gibt es keine Möglichkeit die Dateien zu bearbeiten, wenn hier nichts eingestellt ist..)

Scripts and code:
Gehe zu Kapitel 23 für nähere Informationen.

Colors:
Gehe zu Kapitel 23 für nähere Informationen.

Image editor:
Normalerweise benützen Game Maker-User den eingebauten Editor für das Bearbeiten von Bildern.
Hier kann man ein anderes Bildbearbeitungsprogramm für das Bearbeiten der Bilder einstellen.

15.2 Edit menu – Das Bearbeiten Menü

In diesem Menü findest du folgende zusätzliche Kommandos:

Insert group:
Ressourcen können in Gruppen zusammengefasst werden. Dies ist sehr nützlich, wenn ihr grössere Spiele erstellt, z.B. könnt ihr alle Sounds eines bestimmten Objektes in einer Gruppe organisieren, oder alle Objekte eines Levels in einer Gruppe zusammenfassen. Dieses Kommando erstellt eine neue Gruppe im aktuell gewählten Ressourcentyp. Du wirst nach einen Namen gefragt. Gruppen können wiederum Gruppen enthalten. Du kannst dann die Ressourcen in die Gruppen verschieben.

Find Resource:
Mit diesem Kommando gibst du den Namen der Ressource ein und dann wird das Eigenschaftsfenster geöffnet.

Show Object Information:
Dieses Kommando zeigt einen Überblick über das Objekt an.

15.3 Add menu – Das Hinzufügen Menü

In diesem Menü kannst du jetzt auch zusätzliche Ressourcen hinzufügen. Beachte, es gibt für alle auch einen Button in der Toolbar und ein Tastenkürzel.

15.4 Scripts menu – Das Skript Menü

Im Skript-Menu gibt es folgende zusätzliche Kommandos:

Import Scripts:
Importieren von wichtigen Skripts aus Dateien. (Im Kapitel 21 wird darauf näher eingegangen.)

Export Scripts: Speichern von Skripts in eine Datei, damit es andere nützen können. Wenn du eine Skriptressource auswählst wird nur das eine Skript gespeichert. Wenn du eine Gruppe selektierst, wird die ganze Gruppe gespeichert. Wenn du die root Ressource wählst werden

alle Skripts gespeichert. Siehe dazu Kapitel 21. Dieser Menü-Eintrag ist auch verfügbar, wenn du mit der rechten Maustaste auf ein Skript klickst.

Show Built-in Variables:
Zeigt eine sortierte Liste aller eingebauten Variablen an, lokale und globale.

Show Built-in Functions:
Zeigt eine sortierte Liste aller eingebauten Funktionen.

Show Constants:
Zeigt die Liste aller eigebauten und in den Optionen definierten Konstanten.

Show Resource Names:
Zeigt eine sortierte Liste aller Ressourcen. Klicke darauf, um die betreffende Ressource zu öffnen.
Search in Scripts:
Du kannst hier nach einer Zeichenkette (string) in allen Scripten suchen. Du kannst darauf klicken , um die betroffene Stelle zum Bearbeiten zu öffnen.

Check Resource Names:
Prüft alle Ressourcennamen. Wenn diese nicht korrekt sind, werden sie angezeigt, wenn es doppelte gibt oder wenn ein Ressourcenname der Name einer Variablen, Funktion oder Konstante ist. Du kannst darauf klicken, um die betroffene Ressource zu ändern.

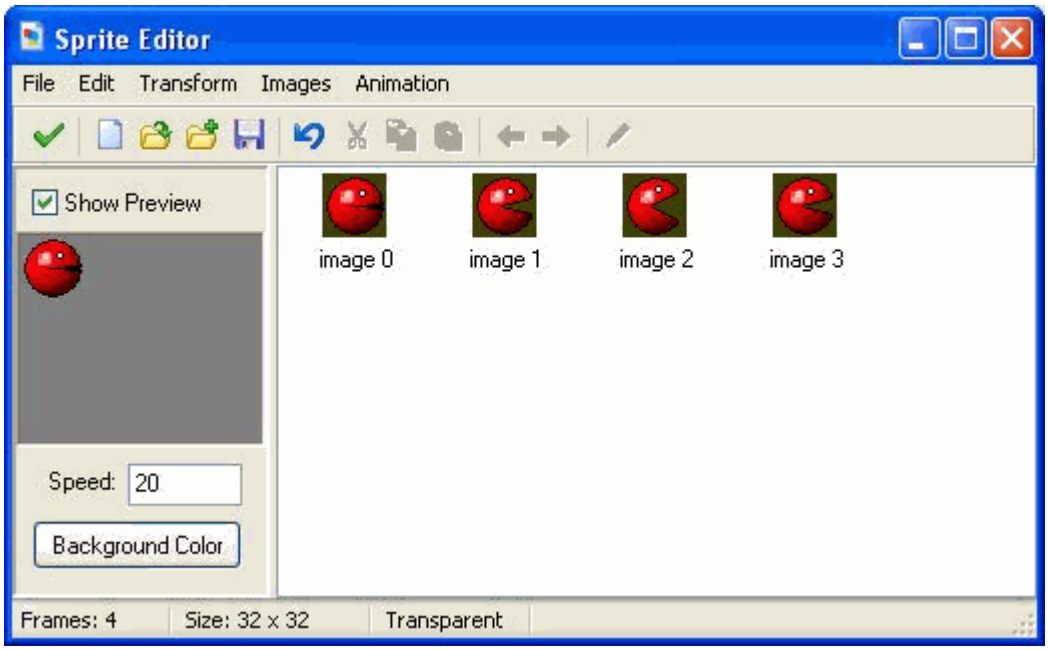
Check All Scripts:
Prüft alle Scripts auf Fehler. Du kannst an die Stelle klicken und der angezeigte Platz wird zum Verändern geöffnet.

16. Mehr über Sprites

Bist jetzt haben wir Sprites aus Dateien geladen. Es ist aber auch möglich, sie zu erstellen und sie im Game Maker zu bearbeiten. Dies geschieht durch Doppelklick auf das Sprite(oder durch Erstellen eines Neuen), ein Fenster öffnet sich. Nun drückt den Button Edit Sprite. Ein neues Dialogfenster erscheint, welches alle Einzelbilder des Sprites zeigt.

16.1 Bearbeiten deines Sprites

Der Sprite-Editor sieht wie folgt aus:



Rechts seht ihr die verschiedenen Bilder des Sprites. Alle Einzelbilder eines Sprites müssen im Game Maker die gleiche Grösse haben. Links wird eine Animation des Sprites abgespielt (Wenn diese nicht angezeigt wird, markiere die Box "Show Preview"). Darunter kann die Geschwindigkeit der Animation und die Hintergrundfarbe geändert werden. So kannst du sehen wie die Animation im Spiel aussehen wird. (Diese Geschwindigkeit ist nur für die Vorschau. Die Geschwindigkeit der Animation im Spiel hängt von der Raumgeschwindigkeit ("room speed") ab).

Der Sprite-Editor beinhaltet viele Befehle für das Erstellen und Verändern von Sprites. Diese werden alle über das Menü gesteuert. (Manche besitzen einen Button in der Symbolleiste). Manche Befehle funktionieren nur bei einzelnen Bildern, daher muss zuerst einen Einzelbild mit der Maus markiert werden.

File menu –Das Datei Menü

Das "File Menu" beinhaltet einige Befehle für das Laden und Speichern von Sprites.

-New. Erstellt ein neues, leeres Sprite. Du musst die Grösse des Sprites angeben (alle Bilder eines Sprites müssen die gleiche Grösse haben).
-Create from file. Erzeugt das Sprite aus einer Datei. Viele verschiedene Dateitypen können verwendet werden. Dies erstellt ein Sprite mit einem einzelnen Bild, ausgenommen eine „Animated-GIF-Datei“, welches die Daten in Einzelbilder speichert.

Das Pixel links unten wird als Transparenzfarbe genommen und nicht die Transparenzfarbe der GIF-Datei. Du kannst mehrere Bilder auswählen, diese müssen aber die gleiche Grösse haben.
-Add from file. Fügt ein Bild (oder Bilder) aus einer Datei zum aktuellen Sprite hinzu. Wenn das Bild nicht die gleiche Grösse hat, kannst du auswählen, wo das Bild platziert wird oder ob es gedehnt wird. Du kannst mehrere Bilder auswählen, diese müssen aber die gleiche Grösse haben.
-Save as GIF. Speichert das Sprite als „Animated-Gif “.
-Save as strip.Speichert das Sprite als Bitmap, die Bilder werden nebeneinander gespeichert.
-Create from strip. Erstellt ein Sprite aus einem Strip. Weiter unten gibt es nähere Informationen.
-Add from strip. Fügt Bilder aus einem Strip hinzu. Weiter unten gibt es nähere Informationen.
-Close saving changes. Schliesst das Fenster und speichert die Änderungen des Sprites. Wenn du die Änderungen nicht speichern willst, klicke auf den Schliessen-Button des Fensters.

Edit menu –Das Bearbeiten Menü

Das Edit-Menu enthält einige Befehle, die sich mit dem aktuell gewählten Sprite befassen. Du kannst es in die Zwischenablage kopieren, ein Bild von der Zwischenablage einfügen, das aktuelle Sprite löschen, es entfernen und das Sprite nach links oder rechts bewegen. Zuletzt gibt es einen Befehl, um ein einzelnes Bild im eingebauten Bildbearbeitungsprogramm zu bearbeiten (siehe unten).

Transform menu – Das Veränderungsmenü

Im Transform-Menu kannst du einige Veränderungen an den Bildern vornehmen.

-Mirror horizontal. Spiegelt das Bild waagrecht.
-Flip vertical. Dreht das Bild senkrecht.
-Shift. Das Bild waagrecht oder senkrecht verschieben.
-Rotate. Das Bild 90 Grad,180 Grad, oder einen anderen Wert drehen. Ausserdem kann beim letzten Punkt die Qualität gewählt werden. Experimentiere um den besten Effekt zu erzielen.
-Resize Canvas. Ändert die Grösse der Leinwand. Ausserdem kann angegeben werden, wo das alte Bild in der neuen Leinwand erstellt wird.
-Stretch. Das Bild zu einer neuen Grösse dehnen. Der Veränderungsfaktor und die Qualität kann gewählt werden.
-Scale. Verändert die Grösse des Bildes (aber nicht die der Leinwand). Der Veränderungsfaktor, die Qualität und die Position im aktuellen Bild kann gewählt werden.

Images menu –Das Bild Menü

Im Images Menu könnt ihr Funktionen für das Bearbeiten von Bildern finden..

-Cycle left. Verschiebt alle Bilder um einen Platz nach rechts. Dies startet die Animation von einem anderen Punkt aus.
-Cycle right. Verschiebt alle Bilder einen Platz nach rechts.
-Black and white. Wandelt das Sprite in schwarz-weiß (verändert nicht die Transparenzfarbe!).
-Colorize. Verändert die Farbe (Farbton) der Bilder. Benutze den Regler, um verschiedene Farben auszuwählen.
-Colorize Partial. Hier kannst du Teile des Bildes verfärben. Du kannst eine vorhandene Farbe wählen und einen Toleranzbereich.
Gib den neuen Farbton an, der die alte Einfärbung (samt Toleranz) ersetzt.
-Shift Hue. Ein anderer Weg die Bildfarbe zu ändern. Hier werden alle Farben, je nach Reglerstellung, mehr oder weniger verändert, was interessante Effekte ergibt.

-Intensity. Verändert die Intensität durch Veränderung der Werte für die Farbsättigung und Helligkeit des Bildes.
-Fade. Angabe einer Farbe und der Menge. Die Farbe des Bildes wird jetzt zu dieser Farbe ausgeblendet.
-Transparency. Angabe des Levels der Transparenz (screen-door transparency). Es wird eine bestimmte Anzahl von Pixeln transparent gemacht.
-Blur.Verwischt das Bild, die Farben werden vermischt, macht es mehr verschwommen. Je höher der Wert, desto verschwommener wird es.
-Crop. Macht das Bild so klein wie möglich. Das ist sehr nützlich, weil grössere Bilder mehr Videospeicher benötigten. Manchmal will man einen kleinen Rand um das Bild lassen, um Transparenzprobleme zu vermeiden.

Du müsst mit diesen Befehlen experimentieren, um das gewünschte Sprite zu erhalten.

Animation menu –Das Animations Menü

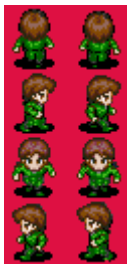
Im Animation Menu werden neue Animationen aus der aktuellen Animation erstellt. Es gibt viele Möglichkeiten und du solltest ein bisschen experimentieren, um den gewünschten Effekt zu erzielen. Vergiss nicht: Du kannst immer eine Animation speichern und später zur aktuellen hinzufügen. Ausserdem kannst du leere Bilder hinzufügen und nicht benötigte löschen. Ich werde kurz die verschiedene Möglichkeiten vorstellen.

-Set Length. Verändert die Länge einer Animation. Die Animation wird solange wiederholt, bis die angegebene Anzahl der Einzelbilder erreicht wird.
-Stretch. Dieser Befehl verändert auch die Länge der Animation. Aber diesmal werden Einzelbilder verdoppelt oder gelöscht, um die richtige Anzahl zu bekommen. Wenn du die Anzahl erhöhst, wird die Animation langsamer, wenn du erniedrigst wird die Animation schneller.
-Reverse. Dreht die Animation um. Die Animation wird also rückwärts gespielt.
-Add Reverse. Die Umkehrsequenz wird diesmal hinzugefügt, verdoppelt also die Anzahl der Einzelbilder. Das ist nützlich, um Objekte links und rechts gehen zu lassen, die Farbe zu ändern und umzudrehen. Manchmal wird man die auftretenden doppelten, ersten und mittleren, Einzelbilder löschen.
-Translation sequence. Erstellt eine Animation, die Bilder werden langsam mit jedem Schritt in die gewünschte Richtung verändert. Du musst die Anzahl der Einzelbilder und die gesamte Menge der waagerechten und senkrechten Veränderung angeben.
-Rotation sequence. Erstellt eine Animation, die das Bild dreht. Wähle im Uhrzeigersinn oder gegen den Uhrzeigersinn. Gibt die Anzahl der Einzelbilder und den Winkel in Grad (360 ist eine komplette Drehung). (Möglicherweise musst du die Grösse der Leinwand verändern, um sicherzugehen, dass das Bild die ganze
Drehung über sichtbar ist.)
-Colorize. Erstellt eine Animation, welche das Bild in eine bestimmte Farbe wechselt.
-Fade to color. Erstellt eine Animation, welche das Bild zu einer bestimmten Farbe ausblendet.
-Disappear. Das Bild verschwindet langsam.
-Shrink. Verkleinert das Bild ins Nichts. Die Richtung kann angegeben werden.
-Grow. Entwickelt das Bild aus dem Nichts.
-Flatten. Drückt das Bild ins Nichts in der angegebenen Richtung.
-Raise. Zieht das Bild aus dem Nichts in die gewünschte Richtung.
-Overlay. Überlagert eine Animation mit einer anderen Animation oder einem Bild aus einer Datei.
-Morph. Morpht die Animation in eine andere Animation bzw. Bild aus einer Datei. Der beste Erfolg wird erzielt, wenn die zwei Animationen den gleichen Bereich eines Bildes bedecken. Sonst erscheinen teilweise manche Pixel und andere verschwinden.

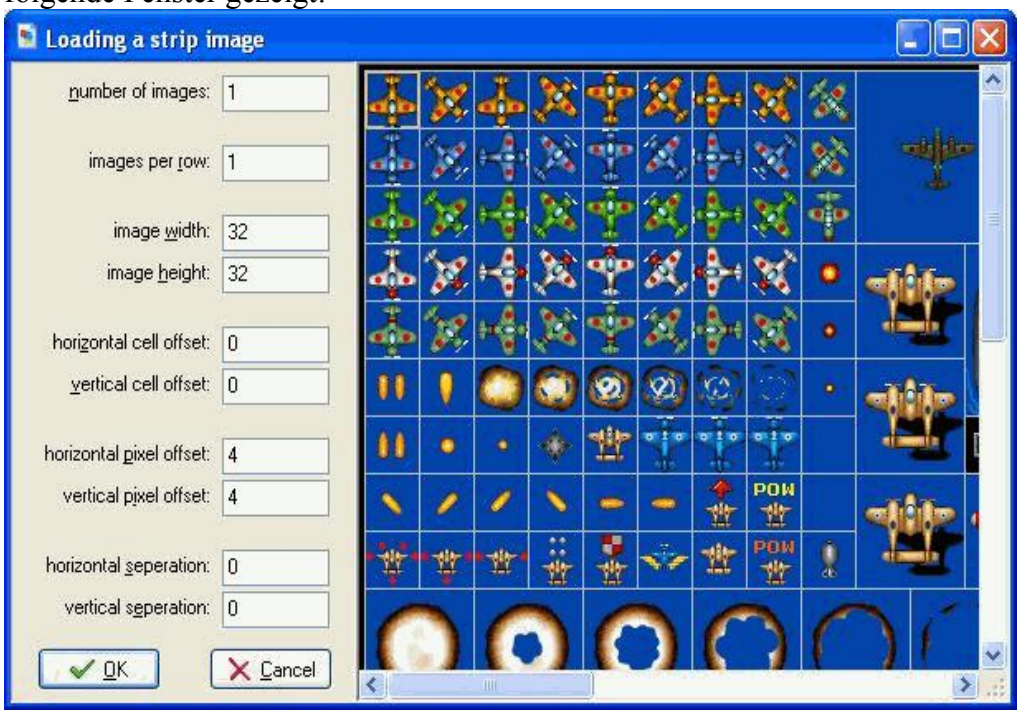
Besonders mächtig sind die beiden letzten Befehle. Um z.B. ein Objekt aufzublasen(sprengen), nimm eine Anzahl von Kopien und eine Anzahl von leeren Einzelbilder, dann überziehe es mit einer Explosionsanimation (Gehe sicher, dass die Anzahl der Bilder übereinstimmen). Alternativ morphe es in die Explosion. Mit ein bisschen Erfahrung kannst du grossartige Sprites erstellen.

Strips -Streifen

Wie oben angegeben, werden Sprites normalerweise als „Animated-Gif "-Datei oder als Strip gespeichert. Ein Strip ist eine große Bitmap, welche die einzelnen Bilder nebeneinander speichert. Das einzige Problem dabei ist, dass die Grösse der einzelnen Einzelbilder nicht im Bild gespeichert wird. Ausserdem speichern die meisten Strip-Dateien aus dem Internet vielfältige Sprites in einer Datei. Zum Beispiel beinhaltet folgender Teil eines Strips vier verschiedene Animationen.



Um einzelne Sprites von einer solchen Dateie zu wählen, benutze Create from Strip oder Add from Strip vom File Menu. Nach Auswählen eines geeigneten Strip von einer Strip-Datei, wird das folgende Fenster gezeigt:



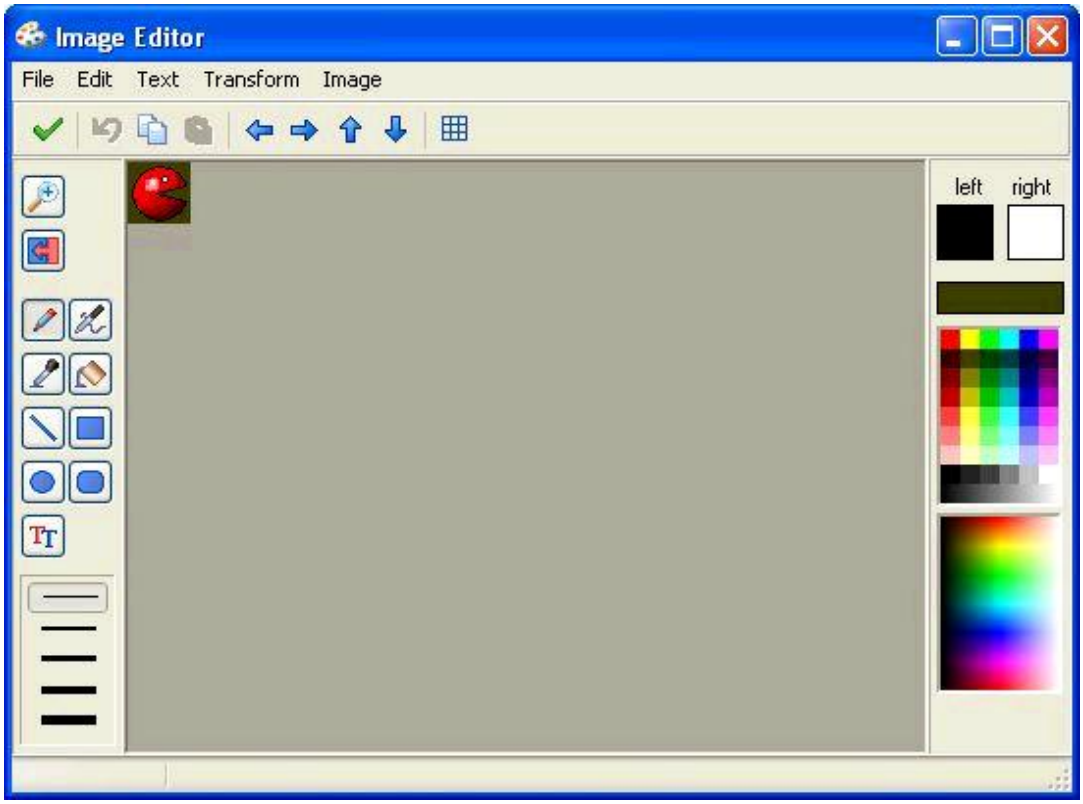
Rechts siehst du (einen Teil) des Strip-Bildes, welches du gewählt hast. Links kannst du einige Parameter sehen, wo du die benötigten Einzelbilder auswählst. Beachte, dass ein oder mehr Rechtecke im Bild, die Bilder die du gewählt hast, anzeigen. Folgende Parameter können eingestellt werden:

-Number of images. Anzahl der Bilder die vom Strip benötigt werden.

-Images per row. Anzahl der Bilder in einer Reihe. Zum Beispiel, wenn 1 ausgewählt wurde, bekommst du eine senkrechte Sequenz von Bildern.
-Image width. Breite des einzelnen Bildes.
-Image height. Höhe des einzelnen Bildes.
-Horizontal cell offset. Wenn du nicht das linke obere Bild benötigst, kannst du hier einstellen wie viele Bilder waagerecht ausgeschlossen werden.
-Vertical cell offset. Wieviele Bilder senkrecht übersprungen werden.
-Horizontal pixel offset. Manchmal gibt es leeren Platz in der oberen Ecke. Hier kannst du die Menge angeben(in Pixel).
-Vertical pixel offset. Vertikale Menge des leeren Platzes.
-Horizontal separation. In manchen Strips gibt es leeren Platz zwischen den Bildern. Hier kannst du die waagerechte Menge zwischen den Bildern (in Pixeln) überspringen.
-Vertical separation. Senkrechter Abstand zwischen den Bildern .

Wenn die korrekte Zahl der Bilder gewählt wurde, drücke OK, um das Sprite zu erstellen. Beachte, dass Sprites von anderen nur mit deren Erlaubnis verwendet werden dürfen oder wenn sie Freeware sind.

16.2 Bearbeiten von Einzelbildern



Du kannst auch die Einzelbilder bearbeiten. Selektiere ein Einzelbild und wähle "Edit Image" vom "Image Menu". Dies öffnet das kleine eingebaute Bildbearbeitungsprogramm. Aber bedenke, dies ist ein eingeschränktes Programm für kleine Änderungen von bestehenden Bildern und nicht eins, um neue Bilder zu zeichnen. Dafür werden vollwertige Bildbearbeitungsprogramme empfohlen, lade diese Dateien dann (oder kopiere und füge es ein) im Game Maker.

Dieses Fenster zeigt das Bild in der Mitte und einige einfache Zeichenfunktionen links. Hier kannst du ein- und auszoomen, Pixel, Linien, Rechtecke, Text etc. zeichnen. Die Farbe hängt davon ab, welche Maustaste benützt wird. Für manche Zeichenfunktionen kannst du Eigenschaften setzen (wie Linienstärke oder Sichtbarkeit des Randes). Es gibt einen speziellen Button, um alle Pixel einer Farbe in eine andere zu ändern. Dies ist vor allem nützlich, um die Hintergrundfarbe für die Transparenz zu ändern. In der Werkzeugleiste gibt es einige besondere Buttons, die alle Pixel eines Bildes in eine bestimmte Richtung bewegen.
Ausserdem kann man einstellen, ob ein Gitter/Raster angezeigt wird, wenn das Bild gezoomt wird (arbeitet nur mit einem Zoomfaktor von mindestens 4).

Rechts im Fenster wird die Farbe gewählt (eine mit der linken Maustaste und eine mit der rechten Maustaste). Es gibt vier Arten, die Farbe zu ändern. Zuerst kannst du mit der Maustaste (links oder rechts) in eine der 16 Basisfarben klicken. Beachte, dass es eine spezielle Farbenbox gibt, welche die Farbe des linken unteren Pixels des Bildes zeigt, die als Transparenzfarbe im Sprite verwendet wird, wenn das Sprite transparent ist. Du kannst diese Farbe verwenden, um Teile des Bildes transparent zu machen. Der zweite Weg ist, in das Bild zu klicken, um die Farbe zu ändern. Hier kannst du viel mehr Farben wählen. Du kannst die Maus halten, um die gewählte Farbe zu sehen. Drittens kannst du mit der linken Maustaste in die Box klicken. Ein Farbdialog öffnet sich, wo du die Farbe ändern kannst. Letzte Möglichkeit, du kannst das Droptool links anklicken, dann auf eine Position im Bild, um die Farbe zu kopieren.

Im Menu sind die gleichen Veränderungsmöglichkeiten und Bildbearbeitungskommandos, wie im Sprite-Editor. Doch diesmal wirken sie sich nur auf das gewählte Bild aus. (Wenn das Sprite mehrere Bilder hat, sind einige Befehle, z.B. Größe ändern, nicht verfügbar). Die Bilder können als Bitmap-Datei gespeichert werden.
Es gibt zwei zusätzliche Befehle im Image Menu:

-Clear. Füllt das Bild komplett mit der linken Farbe (welche automatisch die transparente Farbe wird).
-Gradient fill. Füllt ein Bild mit einem Farbverlauf (nicht sehr nützlich für Sprites, aber es sieht gut aus und kann für Hintergründe genützt werden).

Es gibt keine Möglichkeit, Teile eines Bildes auszuwählen. Auch fehlen einige schicke Zeichenfunktionen.
Dafür solltest du ein professionelles Zeichenprogramm verwenden (oder einfach das Paint Programm von Windows). Der einfachste Weg ist das Kopieren des Bildes in die Zwischenablage. Im Zeichenprogramm drücke dann Einfügen, um das Bild zu erhalten. Ändere es und kopiere es wieder in die Zwischenablage. Jetzt füge das Bild im Game Maker ein.

16.3 Fortgeschrittene Eigenschaften

Im Fortgeschrittenen-Modus gibt es im Sprite-Dialogfenster einige zusätzliche Optionen, die wir hier betrachten.

Zuerst gibt es Optionen bezüglich der Kollisionsprüfung. Immer, wenn zwei Instanzen sich treffen, gibt es ein Kollisions-Event. Kollisionen werden wie folgt ausgeführt.

Jedes Sprite hat einen Begrenzungsrahmen. Dieser Rahmen beinhaltet den nicht-transparenten Teil des Einzelbildes. Wenn die Bengrenzungsrahmen sich überlappen, wird geprüft ob zwei Pixel im aktuellen Einzelbild der 2 Sprites überlappen. Die zweite Option ist aufwendig und benötigt zusätzlichen Speicherplatz.

Wenn du nicht an präziser Kollisionsprüfung eines bestimmte Sprites interessiert bist, solltest du die Box "Precise collision checking" ausschalten. In diesem Falle ist nur der Begrenzungsrahmen maßgebend. Du kannst auch den Begrenzungsrahmen verändern. Dies wird kaum benötigt, doch manchmal willst du den Begrenzungsrahmen kleiner machen, so dass Kollisionen mit erweiterten Teilen des Sprites nicht in Betracht gezogen werden.
Sprites können auf zwei Arten gespeichert werden: Videospeicher und Arbeitsspeicher. Videospeicher gibt es in der Grafikkarte und er ist schneller. Also wenn du viele Instanzen eines Sprites hast, ist der Videospeicher die beste Wahl. Aber der Videospeicher ist begrenzt, abhängig von der Grafikkarte des PC. Also wird empfohlen große Sprites nicht in den Videospeicher zu laden.
Manche Sprites werden in nur einem oder zwei Levels deines Spieles benötigt. Es ist eine Verschwendung, wenn du diese Sprites die ganze Zeit im Speicher behältst. In diesem Fall kannst du die Box "Load only on use" ankreuzen. Das Sprite wird nun in dem Moment geladen, in dem es benötigt wird. Wenn der Raum beendet ist, wird es wieder gelöscht und der Speicher wird freigegeben. Für große Spiele mit vielen Sprites ist es wichtig, welche Sprites geladen werden und welche in den Videospeicher gegeben werden. (Du kannst Sprites auch Laden und Entfernen vom Code aus).

Schlussendlich kannst du den Bezugspunkt (origin) des Sprites setzen. Das ist der Punkt des Sprites, welcher mit der Position korrespondiert. Wenn du eine Instanz bei einer bestimmte Position setzt, wird der Bezugspunkt hier gesetzt. Die obere linke Ecke des Sprites ist voreingestellt, aber manchmal ist es besser die Mitte oder irgendeinen anderen Punkt auszuwählen. Du kannst auch einen Punkt ausserhalb des Sprites verwenden. Du kannst den Punkt auch durch Klicken in den Sprite setzen (wenn der Bezugspunkt im Sprite-Bild angezeigt wird).

17. Mehr über Sound und Musik

Wenn du deinem Spiel eine Soundkomponente hinzufügst, kannst du eine Anzahl an Einstellungen vornehmen - sie sind nur im "advanced mode" zugänglich.
Für alle "sounds" kannst du angeben, ob sie bei Bedarf geladen werden sollen. Dies ist für MIDI-Dateien voreingestellt aber nicht für WAV-Dateien. Wenn du hier einen Haken hinmachst, wird der "sound" nicht schon bei Spielbeginn in den Speicher geladen. Nur wenn er benötigt wird, wird er geladen. Das kann zu einer kleinen Verzögerung führen aber es spart eine Menge an Speicherplatz und bedeutet für das Spiel, dass es kürzere Ladezeiten hat. Am Ende des Raumes, wird der "sound" aussortiert und der belegte Speicherplatz freigegeben. Nur wenn er nochmals benötigt wird, wird er wieder geladen. Verwende dies nicht bei kurzen SFX aber lange Hintergrundmusik oder große Teile, welche nur gelegentlich abgespielt werden, sind dafür gedacht.
Für WAV-Dateien kannst du die Anzahl der Pufferspeicher (buffer) angeben. Diese Zahl gibt an, wie oft der "sound" gleichzeitig wiedergegeben werden kann. Beispielsweise, wenn du eine Explosions-SFX hast und einige Explosionen gleichzeitig stattfinden, kannst du die Zahl erhöhen, sodass alle Explosionen gehört werden können. Aber Vorsicht: Viele Pufferspeicher benötigen viel RAM (Soundkartenabhängig).

Du kannst auch angeben, ob der "sound" für Sound-Effekte vorbereitet werden soll. Diese Effekte (Panorama, Lautstärke) können nur aus dem Programmcode heraus gesteuert werden. "Sounds", welche für Effekte benutzt werden, benötigen mehr Speicher. Der Game Maker hat keinen eingebauten Sound-Editor. Aber in den Voreinstellungen, kannst du einen externen Editor für SFX angeben. Wenn du dieses Feld ausgefüllt hast, kannst du einfach den "Edit sound"-Knopf drücken, um den "sound" zu bearbeiten - der GM bleibt solange im Hintergrund.
Neben MIDI- und WAV-Dateien gibt es noch eine 3. Sorte von "sounds": MP3-Dateien. Das sind komprimierte WAV-Dateien. Auch wenn du sie in der Sound-File-Auswahl nicht sehen kannst, benutzen kannst du sie trotzdem. Wähle "show all files" im Dialogfenster und sie werden angezeigt. Aber Vorsicht, da gibt es einige Nachteile!
Zuerst müssen sie dekomprimiert werden, was Prozessorzeit kostet und das Spiel verlangsamen kann. Die Tatsache, dass die Datei klein ist, heisst noch lange nicht, dass sie auch weniger Speicher verwendet. Zweitens werden MP3’s nicht von allen Systemen unterstützt, so dass dein Spiel dann nicht auf allen Rechnern läuft. Vorzugweise solltest du keine MP3’s verwenden oder, wenn du sie dennoch verwenden willst, sie vorher in WAV-Dateien konvertieren oder nur als Hintergrundmusik einsetzen.

18. Mehr über Hintergrundbilder

Neben dem Laden aus Dateien, kannst du auch eigene Hintergrundbilder erstellen. Um das zu tun, drücke auf den "Edit Background"-Knopf. Ein kleines eingebautes Malprogramm wird geöffnet, in dem du deinen Hintergrund erstellen/bearbeiten kannst. Beachte bitte, dass dies kein vollausgewachsenes Malprogramm ist.
Für fortgeschrittene Bearbeitungsaufgaben, verwende bitte ein eigenständiges Malprogramm. Da ist eine Option, die sehr nützlich ist. Im "Image Menu" findest du einen Befehl "Gradient Fill". Dieser kann verwendet werden, um einen schönen Farbverlauf-Hintergrund zu kreieren.
Im "advanced mode" gibt es eine Anzahl von Zusatzoptionen hier.
Üblicherweise werden Hintergrundbilder im VIDEO-RAM gespeichert. Da ist auch gut so, wenn sie klein sind - wenn sie grösser sind, willst du sie vielleicht im normalen Speicher ablegen. Das ist etwas langsamer aber VIDEO-RAM ist begrenzt! Um dies zu bewerkstelligen, entferne den Haken aus dem "Use video memory" Kästchen.

Obgleich voreingestellte Hintergrundbilder geladen werden, wenn sie benötigt werden und wieder entfernt werden, wenn der Raum beendet wird, spart dieses eine Menge an Speicherplatz - verzögert aber das Laden des Raumes (bis hin zu einem kleinem "Ruckler", wenn der Hintergrund innerhalb des Raumes gewechselt wird).
Um dies zu vermeiden, entferne den Haken bei "Load only on use".

19. Mehr über Objekte

Wenn du ein "object" im "advanced mode" (Expertenmodus des GM) erschaffst, kannst du mehr Parameter bestimmen.

19.1 Depth – Zeichenebene

Zuerst einmal, kannst du die Zeichenebene der Instanzen eines Objektes bestimmen. Wenn Instanzen auf dem Bildschirm gezeichnet werden, geschieht dies in der Reihenfolge der Zeichenebenen.
Instanzen mit dem höchsten "Depth"-Wert, werden zuerst gezeichnet (auf einem Stapel ganz unten, wenn man draufschaut). Instanzen mit dem kleinsten Wert werden zuletzt (obendrauf) gezeichnet. Wenn zwei Instanzen die gleichen "Depth"-Werte aufweisen, werden sie gezeichnet, wenn sie erschaffen (created) werden. Wenn du sichergehen willst, dass ein Objekt vor allen anderen liegt, gib ihm einen negativen Wert für "Depth". Wenn es dahinter liegen soll, gib ihm einen möglichst hohen positiven Wert. Du kannst die Zeichenebene sogar während des Spiels ändern, indem du die Variable depth verwendest.

19.2 Persistent Objects - bleibende Objekte

Zweitens, kannst du bleibende Objekte erschaffen. Ein bleibendes Objekt übersteht den Wechsel von Raum zu Raum. Es verschwindet nur, wenn es explizit vernichtet wird. Du brauchst also nur eine Instanz des Objekts in den ersten Raum setzen und sie ist dann in allen darauffolgenden Räumen verfügbar. Dies ist nützlich z.B. für eine Hauptfigur, welche sich von Raum zu Raum durchspielt. Der Gebrauch von "persistent objects" ist eine mächtige Funktion, welche leicht zu Fehlern führt.

19.3 Parents – Eltern

Jedes Objekt kann ein "parent-object" haben. Wenn ein Elternobjekt vorhanden ist, "erbt" das Kind die Eigenschaften der Eltern. Anders ausgedrückt, das Objekt ist eine Art Spezialfall vom Elternobjekt. Als Beispiel: Wenn du 4 verschiedene Bälle (bezeichnet mit: ball1, ball2, ball3 und ball4) hast und alle haben das gleiche Verhalten nur andere sprites, dann kannst du ball1 zum "parent-object" von den drei anderen Bällen machen. Jetzt muss du nur noch in ball1 die Parameter ändern - die restlichen Bälle beinhalten diese dann auch und verhalten sich wie ball1. (Gilt für "events" und "actions"). Wenn du beispielsweise ball1 Instanzen vernichtest, werden auch alle Instanzen von ball2, ball3 und ball4 vernichtet. Das spart eine Menge an Arbeit. Oftmals sollen sich Objekte genau gleich verhalten, bis auf wenige Unterschiede. Zum Beispiel soll ein Monster auf- und abwärts bewegt werden, ein anderes nach links und rechts. Ansonsten verhalten sie sich gleich. In diesem Fall sollten alle "events" und "actions" gleich sein, bis auf ein oder zwei Unterschiede. Wieder können wir eins der Objekte zum "parent" des anderen machen. Aber in diesem Fall definieren wir auch für das Kindobjekt (child-object) bestimmte "actions". Diese Festlegungen haben Vorrang, vor denen des "parent-object". Wenn also ein "child-object" eigene Anweisungen enthält, werden diese anstelle der des "parent-object" ausgeführt. Wen du auch das "parent-event" ausführen willst, musst du das sogenannte "inherited-event" mit der entsprechenden Anweisung aufrufen. Es ist in solchen Fällen üblich, zuerst ein Basisobjekt (ball0) zu erschaffen. Dieses Objekt enthält die Eigenschaften aber wird nie im Spiel benutzt. Alle aktuellen Objekte, haben dieses Basisobjekt als "parent-object". "Parent-objects" könnten selbst wiederum "parent-objects" haben und so weiter (Schleifen sind nicht erlaubt).

Auf diese Weise kannst du eine Hierarchie aufbauen. Es ist höchst nützlich, dein Spiel strukturiert zu halten und ich kann dir nur wärmstens empfehlen, dir diesen Mechanismus zu eigen zu machen. Da gibt es noch eine zweite Möglichkeit "parent-objects" zu gebrauchen. Sie beinhaltet auch das Kollisionsverhalten für andere Objekte. Lass es mich an einem Beispiel erläutern. Angenommen, du hast 4 verschiedene Bodenobjekte. Wenn ein Ball den Boden berührt, soll er seine Bewegungsrichtung ändern.
Dieses muss im "collision-event" vom ball (Kollision ball mit Boden) festgelegt werden. Weil da aber vier verschiedene Bodenobjekte sind, müsstest du vier verschiedene "collision-events" für ball festlegen. Wenn du jetzt aber ein Bodenobjekt zum "parent-object" der anderen Bodenobjekte machst, brauchst du nur einmal ein "collision-event" mit diesem "parent-object" machen - die anderen "erben" es ja. Diese anderen Kollisionen lösen ja das gleiche Ereignis aus. Hier sparst du eine Menge an Kopierarbeit.
Wie aufgezeigt, wann immer du ein Objekt benutzt, beziehst du immer dessen Nachkommen mit ein. Das geschieht, wenn du in einer "action" festlegst, dass Anweisung für Instanzen eines bestimmten Objektes gelten. Es passiert auch, wenn du das "with()-statement" im GML-Code (Script,GML-Segment) verwendest (siehe unten). Und es greift, wenn du Funktionen wie "instance_position, instance_number, etc." aufrufst.
Schliesslich arbeitet dieses Prinzip auch, wenn du auf Variablen in anderen Objekten verweist. Um beim obigen Beispiel zu bleiben: Wenn du "ball1.speed" auf 10 setzt, gilt das auch für ball2, ball3 und ball4.

19.4 Masks - Masken

Wenn zwei Instanzen sich berühren wird ein "collision event" (Kollisionsereignis) ausgelöst. Um zu bestimmen, ob zwei Instanzen sich berühren, nimmt man die zugehörigen "sprites" als Entscheidungsgrundlage. In den meisten Fällen ist das auch gut so, manchmal aber soll die Abfrage einen anderen Umriss als Grundlage nehmen. Zum Beispiel, wenn du ein isometrisches Spiel machst, haben Objekte typischerweise eine Höhe (um einen 3D-Effekt zu erzielen). Für die Kollisionserkennung willst du aber nur den "Bodenteil" des "sprites" benutzen. Dies kannst du machen, indem du ein 2. "sprite" erstellst (mit den benötigten Dimensionen) und es als Maske für die Kollisionserkennung einsetzt.

19.5 Informations - Kurzüberblick

Dieser Knopf zeigt beim Betätigen eine Überblicksansicht des Objektes, die auch ausgedruckt werden kann. Das ist besonders nützlich, wenn man mal den Überblick über die ganzen "events" und "actions" verloren hat.

20. Mehr Aktionen

Im erweiterten Modus gibt es einige weiterführende Aktionen.

20.1 Weitere Bewegungsaktionen

Es gibt ein weiteres Set von Bewegung Aktionen, welche nur im Fortgeschrittenen Modus (Advanced Mode) verfügbar sind:

 **Set a path for the instance (Bestimme einen Pfad für die Instanz)**

Mit dieser Aktion kann man ein Objekt dazu bringen, einem gegebenem Pfad zu folgen. Man stellt den Pfad ein und danach die Geschwindigkeit (in Pixel pro Schritt). Falls die Geschwindigkeit positiv ist startet das Objekt am Anfang, anderenfalls am Ende des Pfades. Als nächstes Stellt man ein, was passieren soll, wenn das Ende des Pfades erreicht ist. Man kann die Bewegung anhalten, von vorne anfangen, von der aktuellen Position aus neu anfangen (was bei einem geschlossenen Pfad das gleiche ist) oder die Bewegung umdrehen. Zum Schluss muss man einstellen, ob der Pfad absolut, - das bedeutet, dass seine Position genauso ist, wie eingestellt (dies ist nützlich, wenn man den Pfad für eine bestimmte Position im Raum erstellt hat) - oder relativ – dann wird der Startpunkt (oder Endpunkt, wenn die Geschwindigkeit negativ ist) des Pfades an der aktuellen Position des Objektes platziert.

Weitere Informationen findet man im Kapitel „Pfade“.

 **End the path for the instance (Beende den Pfad für die Instanz)**

Mit dieser Aktion beendet man den Pfad für die Instanz.

 **Set the position on the path (Bestimme die Position auf dem Pfad)**

Mit dieser Aktion kann die aktuelle Position der Instanz auf dem Pfad verändert werden. Der Wert muss zwischen 0 und 1 liegen (0 = Anfang, 1 = Ende).

 **Set the speed for the path (Bestimme die Geschwindigkeit des Pfades)**

Hiermit ändert man die Geschwindigkeit der Instanz auf dem Pfad. Ein negativer Wert veranlasst die Instanz sich rückwärts zu bewegen. Ein Wert von 0 hält die Instanz an.

 **Perform a step towards a point (Führe einen Schritt in Richtung eines Punktes aus)**

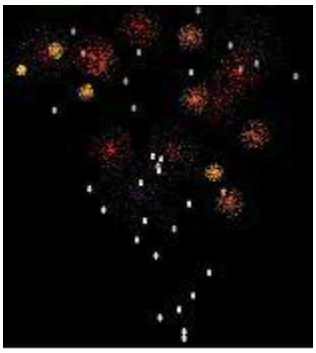
Diese Aktion sollte im Step-Event eines Objektes ausgeführt werden, um es in Richtung eines gewählten Punktes zu bewegen. Falls die Instanz den gewählten Punkt erreicht hat, wird sie anhalten. Man stellt die Zielposition und die Geschwindigkeit (die Größe des Schrittes in Pixel) ein. Außerdem kann man bestimmen, ob die Bewegung angehalten werden soll, wenn eine solide oder eine beliebige Instanz berührt wird.

 **Step towards a point avoiding objects (Führe einen Schritt in Richtung eines Punktes aus und weiche dabei Objekten aus)**

Diese Bewegungsaktion ist sehr praktisch. Man sollte sie im Step-Event eine Objektes platzieren. Genauso wie die obige Aktion veranlasst sie die Instanz einen Schritt in Richtung eines bestimmten Punktes zu machen, allerdings wird sie nun versuchen, Hindernissen auszuweichen. Wenn sich die Instanz in ein solides Objekt (oder optional in irgendeines) bewegen würde, ändert sie die Richtung, um zu versuchen, sich um das Objekt herumzubewegen. Man kann nicht garantieren, dass die Methode funktioniert, aber in den meisten einfachen Fällen wird sie die Instanz effizient zum Zielpunkt bewegen. Für kompliziertere Fälle gibt es die Bewegungsplanungs-Funktionen (motion planning) (siehe das Kapitel über Bewegunsplanung). Man stellt die Zielposition, die Geschwindigkeit (die Größe des Schrittes in Pixel) und, ob die Bewegung nur soliden oder allen Objekten ausweichen soll, ein.

20.2 Partikelaktionen

Partikelsysteme sind dazu gedacht Spezialeffekte zu erzeugen. Partikel sind kleine Elemente (dargestellt durch einen Pixel oder eine andere Form). Solche Partikel bewegen sich nach festgelegten Regeln umher und können während der Bewegung die Farbe ändern. Viele solcher Partikel zusammen können z.B. Feuerwerk, Flammen, Regen, Schnee, Weltraum usw. erzeugen.



Game Maker beinhaltet ein umfangreiches Partikelsystem das durch Funktionen genutzt werden kann. Ein eingeschränktes Partikelsystem kann durch die unten beschriebenen Aktionen benutzt werden.

Ein Partikelsystem kann mit verschiedenen Partikeltypen umgehen. Nachdem das Partikelsystem erstellt wurde, muss als erstes der Partikeltyp festgelegt werden. Mit unten genannten Funktionen kann aus 16 Typen gewählt werden. Jeder Typ hat eine feste Form, Grösse, Startfarbe und Endfarbe. Die Farbe wechselt langsam von der Startfarbe zur Endfarbe. Partikel haben eine begrenzte Lebenszeit. Mit dem Typ kannst du die minimale und die maximale Lebenszeit angeben. Partikel haben auch Geschwindigkeit und Richtung. Schliesslich können Gravitation und Fliehkraft auf Partikel wirken.

Nachdem du die Partikeltypen festgelegt hast, musst du sie an bestimmten Stellen im Raum erzeugen. Du kannst entweder eine bestimmte Anzahl von Partikeln von einer Stelle wegfeuern oder einen konstanten Strom. Partikel erscheinen bei Emitern. Ein Partikelsystem kann bis zu 8 Emitter haben die zur selben Zeit arbeiten. Somit musst du, nachdem du die Partikeltypen fetsgelegt hast, Emiter erstellen und ihnen sagen ob die Partikel abfeuern oder kontinuierlich ausschütten sollen.

Hier ist das komplette Aktionsset. Experimentiere am Besten etwas herum um den gewünschten Effekt zu bekommen. Diese Funktionen sind nur in der registrierten Version verfügbar.

 **Partikelsystem erstellen**
Diese Aktion erstellt das Partikelsystem. Es muss erstellt werden bevor andere Aktionen ausgeführt werden. Du musst es nur einmal erstellen. Du kannst die Tiefe der Partikel angeben. Bei grossen positiven Werten erscheinen die Partikel hinter der Szene. Wenn du eine negative Tiefe angibst erscheinen sie vor der Szene.

 **Partikelsystem zerstören**
Diese Aktion zerstört das Partikelsystem, gibt dessen Speicher frei. Vergesse nicht dies zu tun (z.B. wenn in einen anderen Raum gewechselt wird) weil Partikelsysteme viel Speicher beanspruchen.

 **Alle Partikel im System entfernen**
Diese Aktion entfernt alle sichtbaren Partikel. Es stoppt nicht die Emitter, deshalb können neue Partikel ausgeschüttet werden (siehe unten).

 **Ein Partikeltyp erstellen**
Mit dieser Aktion kannst du einen Partikeltyp erstellen. Du kannst aus den 16 möglichen auswählen. Für den Partikeltyp kannst du die Form, die minimale und maximale Grösse (wenn die Partikel erschienen werden Zufallswerte dazwischen verwendet), die Farbe für den Partikelstart und die Farbe auf die sich die aktuelle Farbe hin verändert. Beachte das nur ein Partikeltyp erstellt wird und kein Partikel. Für dies werden Emitter benötigt (siehe unten).

 **Setze die Lebenszeit eines Partikeltyps**
Ein Partikel lebt nur für eine begrenzte Anzahl von Schritten. Danach verschwindet er. Mit dieser Aktion kannst du die Lebenszeit festlegen. Du gibst minimal und maximal an, der Wert liegt mit Zufall dazwischen.

 **Die Bewegung eines Partikels festlegen**
Mit dieser Aktion kannst du Richtung und Geschwindigkeit der Bewegung eines Partikeltyps festlegen. Wieder kannst du maximal und minimal angeben, per Zufall dazwischen gewählt. Z.B. um einen Partikel zufällig zu bewegen, gebe 0 und 360 als Grenzen für die Richtung an. Du kannst auch die Gravitation festlegen. Dieser Wert wird in jedem Step von der Geschwindigkeit abgezogen bis sie 0 ist. (Du kannst ihn mit negativen Werten beschleunigen.)

 **Setze die Gravitation für einen Partikeltyp**
Mit dieser Aktion kannst du die Gravitation für einen bestimmten Partikeltyp festlegen. 270 ist abwärts.

 **Sekundäre Partikel erstellen**
Dies ist etwas komplizierter. Partikel können andere Partikel in ihrer Lebenszeit und beim Zerstören erzeugen. Mit dieser Aktion kannst du das festlegen. Du kannst den Typ und die Anzahl der Partikel die jeden Step in der Lebenszeit erzeugt werden und dasselbe beim Zerstören eines Partikels. Sei hier sehr behutsam. Du kannst sehr leicht große Mengen Partikel auf diese Weise erstellen, welche das System stark verlangsamen können. Für die Zahlen kannst du auch negative Werte verwenden.

Ein negativer Wert x bedeutet das z.B. in jedem Step ein Partikel mit der Wahrscheinlichkeit -1/x erzeugt wird. Wenn du z.B. jeden 4. Step ein sekundäres Partikel erstellen willst, so gebe -4 an. Sekundäre Partikel sind großartig um z.B. Partikel um Partikel oder explodierende Partikel darzustellen.

 **Erzeuge einen Emitter**
Diese Aktion erzeugt einen Emitter. Partikel werden von Emittern generiert. Du kannst bis zu acht Emitter einsetzen. Wähle den Emitter, seine Form und seine Position und Ausdehnung im Raum (In der Form einer „Randbox“).

 **Einen Emitter zerstören**
Diese Aktion zerstört den angegeben Emitter. Beachte das bereits existierende Partikel von diesem Emitter nicht zerstört werden.

 **Eine Anzahl Partikel vom Emitter wegschleudern**
Selbst wenn due Partikeltyp und Emitter definiert hast. Du musst immer noch dem Emitter sagen, dass er Partikel erzeugen soll. Mit dieser Aktion kannst du einem bestimmten Emitter sagen, dass er eine bestimmte Menge eines bestimmten Partikeltyps wegschleudern soll. Alle Partikel werden auf einmal erstellt. Du kannst für die Zahl auch einen negativen Wert einsetzen. Ein negativer Wert x bedeutet, das ein Partikel mit der Wahrscheinlichkeit von -1/x erzeugt wird. So z.B., wenn du einen Partikel mit 25%iger Wahscheinlichkeit erstellen möchtest, nutze den Wert -4.

 **Partikelstrom vom Emitter erzeugen**
Mit dieser Aktion kannst du einen Emitter dazu bringen, bestimmte Partikel auszuschütten. In jedem Step wird diese Anzahl Partikel ausgeschüttet, was in einem kontinuierlichen Partikelstrom mündet. Der Emitter schüttet Partikel aus bis, er zerstört wird oder du ihm mitteilst 0 Partikel auszuschütten. Für die Anzahl kannst du auch negative Werte verwenden. So z.B., wenn du einen Partikel mit 25%iger Wahrscheinlichkeit erstellen möchtest, nutze den Wert -4.

20.3 Extra Aktionen

Hier kannst du ein paar nützliche Aktionen finden, welche nur im registrierten Game Maker zu finden sind. Diese Funktionen sind nur in der registrierten Version verfügbar.

 **Spiele eine CD ab**
Mit dieser Aktion kannst du einen Track vom Standardlaufwerk des Systems abspielen. Du gibst den Start- und den Endtrack an.

 **Stoppe die CD**
Stoppt die CD-Wiedergabe.

 **Pausiere die CD**
Pausiert die CD-Wiedergabe.

 **Spiele weiter**
Spielt eine pausierte CD weiter.

 **Wenn eine CD im Laufwerk ist**
Wenn eine CD im Laufwerk ist, wird die nächste Aktion ausgeführt.

 **Wenn die CD spielt**
Wenn die CD abgespielt wird, wird die nächste Aktion ausgeführt.

 **Das Sprite mit einer Datei ersetzen**
Diese Aktion kann genutzt werden, um ein Sprite mit dem Inhalt einer Datei zu ersetzen. Du gibst das zu ersetzende Sprite an, den Dateinamen (.bmp, .jpg, oder .gif) und die Zahl der Subimages wenn du ein .bmp oder .jpg lädst. Bei einem .gif Bild wird die Anzahl der Subimages automatisch festgestellt. Andere Sprite-Einstellungen, z.B. ob das Sprite transparent ist, werden nicht verändert. Du kannst dies nutzen um das Speichern aller Sprites im Spiel selbst zu umgehen. Am Start eines Levels kannst du die Sprites mit z.B. dem aktuellen Charaktersprite nach Wahl ersetzen. Ersetze kein Sprite DAS IN BENUTZUNG IST! Dies kann Probleme bei Kollisionen erzeugen.

 **Den Sound mit einer Datei ersetzen**
Mit dieser Aktion kannst du Sound mit dem Inhalt einer Datei (.wav, .mid, oder .mp3). Du gibst den Sound und den Dateinamen an. Du kannst dies nutzen, um das Speichern aller Sounds im Spiel selbst zu umgehen. Z.B., kannst du verschiedene Stücke Hintergrundmusik mit denen ersetzen, die du abspielen möchtest. Ersetze keinen Sound DER IN BENUTZUNG IST!

 **Den Hintergrund mit einer Datei ersetzen**
Mit dieser Aktion kannst du den Hintergrund mit dem Inhalt einer Datei ersetzen (.bmp, oder .jpg). Du kannst den Hintergrund und den Dateinamen angeben. Du kannst dies nutzen um das speichern aller Hintergründe im Spiel selbst zu umgehen. Ersetze keinen Hintergrund DER IN BENUTZUNG IST!

 **Lege den Mauszeiger fest**
Du kannst dies nutzen, um den Windows Mauszeiger mit einem Sprite zu ersetzen. Du legst das Sprite fest und ob der Windows Mauszeiger noch gezeigt werden soll. Das Sprite kann eine Animation sein. Beachte das das Sprite nur im Raum und nicht ausserhalb gezeigt wird.

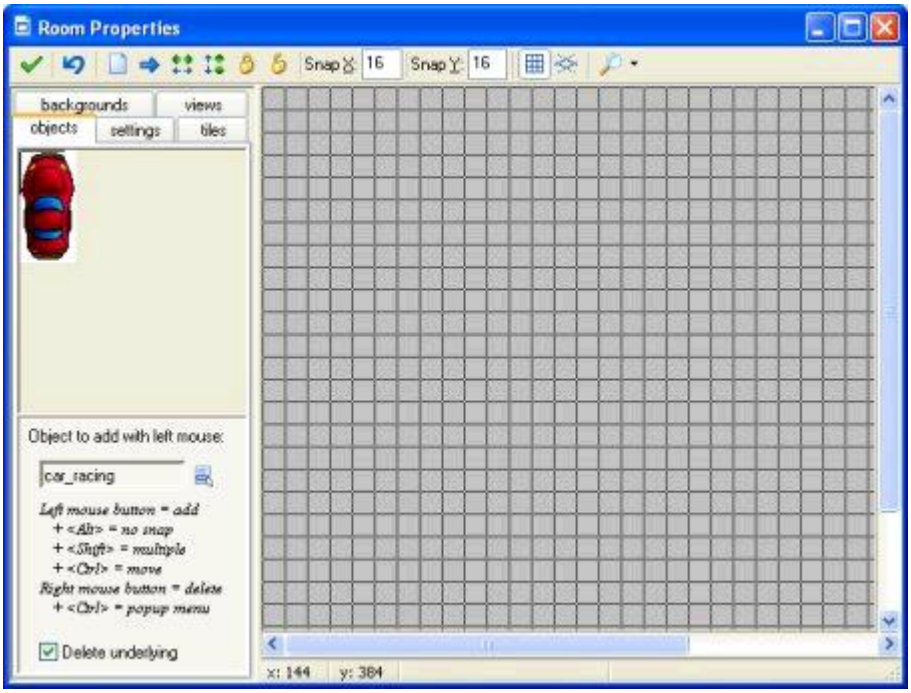
 **Mache einen Bildschirmschnappschuss**
Mit dieser Aktion kannst du ein Bild des aktuellen Geschehens als .bmp Datei speichern. Du gibst den Dateinamen an.

 **Setze den Hintergrund zu einem Verlauf**
Mit dieser Aktion kannst du dem aktuellen Raum einen Farbverlauf als Hintergrund geben. Du kannst die Anfangs- und Endfarbe und die Richtung (Horizontal oder Vertikal) festlegen.

 **Öffne eine Seite im Browser**
Du kannst in dieser Aktion eine Internetadresse angeben. Diese Webseite wird im Standardbrowser des PCs geöffnet werden. (Die Aktion kann auch genutzt werden um andere Dokumente zu öffnen.) Die Aktion funktioniert nicht im sicheren Modus.

21. Mehr über Räume

Räume im Game Maker habe viele Einstellungsmöglichkeiten. In Kapitel 13 betrachteten wir nur die wichtigsten. In diesem Kapitel beschreiben wir die anderen Optionen. Wenn du einen Raum im erweiterten Modus öffnest sieht er folgendermaßen aus:



Wie du siehst wurden, der Toolbar einige neue Buttons hinzugefügt. Diese sortieren die Instanzen horizontal oder vertikal. Dies wird nützlich wenn sich Instanzen überschneiden. (Wenn du Tiles hinzufügst ordnen die Funktionen Tiles anstatt Objekte.) Auch sind dort Buttons um alle Instanzen zu verankern oder zu lösen. Verankerte Instanzen können weder verschoben noch gelöscht werden. Dies bewahrt vor versehentlichem Löschen einer Instanz. Im Kontextmenü (drücke <Ctrl> und klicke mit rechts auf die Instanz) kannst du Instanzen individuell verankern oder lösen.

Schliesslich kannst du angeben ob du ein isometrisches Raster haben möchtest. Dies erleichtert das Erstellen von isometrischen Spielen. Erstens laufen dann die Rasterlinien diagonal. Zweitens ist das Einrasten anders. (Es funktioniert am besten wenn der Befestigungspunkt links oben ist, wie vorgegeben.)

Auch gibt es zwei neue Register, die wir weiter unten besprechen.

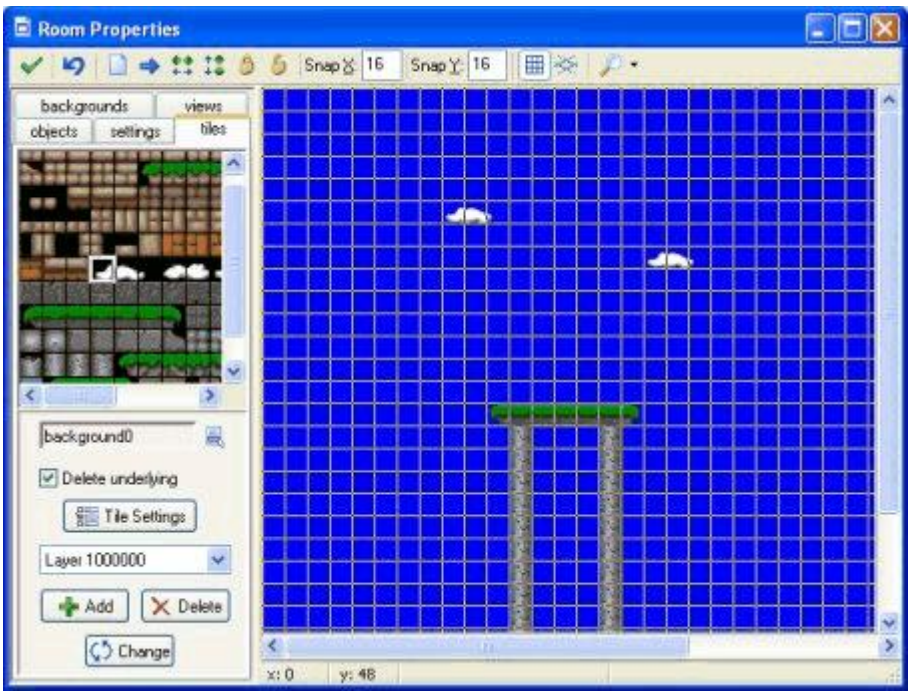
21.1 Fortgeschrittene Einstellungen

Es gibt zwei weitere Dinge im Register "settings", welche wir noch nicht besprochen haben. Zuerst gibt es eine Auswahlbox "Persistent". Normalerweise wird beim Verlassen eines Raumes und späterer Rückkehr der Raum auf seine Anfangseinstellungen zurückgesetzt. Das ist gut so, wenn du einige Levels in dem Spiel hast, aber es wird nicht gewünscht, z.B. in Rollenspielen. Hier soll der Raum so bleiben, wie beim letzten Besuch. Aktivieren der Box "Persistent" macht genau das. Der Raumstatus wird gespeichert und wenn der Raum wieder besucht wird, ist er genauso wie vorher. Nur wenn das Spiel neu gestartet wird, wird der Raum zurückgesetzt. Es gibt aber eine Ausnahme.

Wenn du bei bestimmten Objekten "Persistent" gewählt hast (siehe Kapitel 19), bleiben die Instanzen der Objekte nicht im Raum, sondern gehen zum nächsten Raum mit. Zweitens gibt es einen Button "Creation code". Hier kannst du jede Art von Code in GML (Erklärung später) eingeben, der beim Start des Raumes ausgeführt wird. Das ist nützlich für z.B. bestimmte Variablen für den Raum, Erstellen bestimmter Instanzen, usw. Es ist wichtig zu verstehen, was passiert, wenn man einen bestimmten Raum im Spiel betritt. Im aktuellen Raum bekommen alle Instanzen ein "room-end Event". Danach werden die nicht persistenten Instanzen entfernt (kein "destroy Event" wird erstellt). Danach werden im neuen Raum die persistenten Instanzen vom vorherigen Raum hinzugefügt. Alle neuen Instanzen werden erzeugt und deren "creation Events" werden ausgeführt (wenn der Raum nicht persistent ist oder vorher nie besucht wurde).
Wenn es der erste Raum ist, wird für alle Instanzen das "game-start Event" erstellt. Nun wird das "room-create Event" ausgeführt. Zuletzt bekommen alle Instanzen ein "room-start Event". So z.B. das "room start-event" kann Variablen benutzten, welche im Erzeugungscode des Raumes gesetzt wurden und im Erzeugungscode kannst du dich auf die Instanzen (beide; neue und dauernde(persistente))im Raum beziehen. Es gibt noch eine weitere Option. Im Kontextmenü, das erscheint wenn du mit rechts auf eine Instanz klickst und die <Ctrl> (<Strg>) Taste drückst kannst du einen "creation code" für die spezifische Instanz angeben. Dieser Code wird beim Start des Raumes ausgeführt, gerade vor dem "creation event" der Instanz. Das ist nützlich, z.B. um bestimmte Eigenschaften, die typisch für die Instanz sind, einzugeben.

21. 2 Hinzufügen von Tiles (Kacheln)

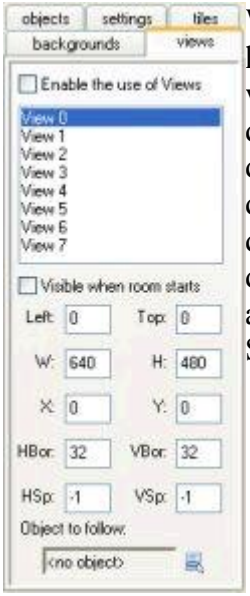
Du kannst auch sogenannte gekachelte Hintergründe erstellen. Der Grund dafür ist folgender. In vielen Spielen willst du einen schönen Hintergrund haben, z.B. sollen in einem Labyrinthspiel die Wände des Labyrinth zusammenpassen und in Plattformspielen willst du viele schön gezeichnete Plattformen, Bäume, usw. sehen. Du kannst im Game Maker viele verschiedene Objekte definieren und den Raum mit diesen Objekten erstellen. Das Problem ist aber die viele Arbeit, grosse Menge an Ressourcen und das Spiel wird langsamer aufgrund der vielen Objekte. Z.B, für schöne Wände in einem Labyrinthspiel werden 15 verschieden gestaltete Wandobjekte benötigt. Der Standardweg ist, wie in vielen Spielen, die Wände und andere statische Objekte werden im Hintergrund gezeichnet. Aber wie weiß das Spiel ob ein Objekt gegen eine Wand stösst, wenn es nur im Hintergrund gezeichnet wird? Der Trick ist folgender: Du erstellst ein Wandobjekt für das Spiel. Es muss die richtige Grösse haben, aber es muss nicht gut aussehen. Wenn der Raum erstellt wird, platziere das Objekt dorthin, wo die Wand ist. Und jetzt der Trick: Mach' das Objekt unsichtbar. Du siehst nur den schönen Hintergrund. Aber die festen Wandobjekte sind noch immer dort und das Objekt im Spiel wird darauf reagieren.
Du kannst diese Technik für jedes Objekt benutzen, welches nicht seine Form oder Position ändert (Du kannst es auch nicht benutzen wenn, das Objekt animiert wird). Für Plattformspiele benötigst du nur einen Boden und ein Wandobjekt, aber du kannst schöne Hintergründe machen, als ob der Spieler auf Gras oder
Baumzweigen geht.
Um dem Raum Kacheln hinzuzufügen, musst du zuerst ein Hintergrundbild, welches eine Kachel ist, einfügen. Einige sind dem Game Maker beigefügt. Wenn du willst, dass die Kacheln teilweise transparent sind, vergewissere dich, dass das Hintergrundbild transparent ist. Nun klicke auf das Register "tiles".
Das folgende Fenster wird gezeigt (hier haben wir schon einige Kacheln dem Raum erstellt).



Links wird die aktuelle Kachel gezeigt. Um das Set auszuwählen, klicke auf den Menu Button darunter und wähle das gewünschte Hintergrundbild.
Mit einem Klick auf "Tile Settings" kannst du einige Einstellungen verändern. Du kannst die Höhe und Breite eines Tiles angeben, den Abstand zwischen den einzelnen Tiles (normalerweise 0 oder 1) und das Offset wo links oben das erste Tile anfängt.
Nun kannst du die Kacheln durch Auswählen der gewünschten Kachel links oben hinzufügen. Dann klicke auf den gewünschten Platz im Raum rechts. Das funktioniert genauso wie das Hinzufügen von Instanzen. Unten liegende Kacheln werden entfernt, solange die Box "Delete underlying" aktiviert ist. Du kannst mit der rechten Maustaste Kacheln löschen. Halte die <Shift> Taste gedrückt, um mehrere Tiles hinzuzufügen. Halte <Ctrl> (<Strg>) gedrückt, um ein existierendes Tile an eine neue Stelle zu bewegen. Drücke die <Alt> Taste, um das einrasten im Raster zu verhindern. Auch erscheint wieder das Kontextmenü wenn, du mit rechts auf ein Tile klickst und die <Ctrl> (<Strg<) Taste gedrückt hälst. Du kannst dort alle Tiles löschen, alle verschieben, die Tiles sortieren oder sie verankern/lösen. (Diese Funktionen arbeiten nur auf dem aktuellen Layer, siehe unten.) In einigen Situationen möchtest du einen Hintergrund einfügen, der nicht exakt die Grösse eines Tiles hat. Dies kann folgendermaßen erreicht. Im Bild Links oben, drücke die linke Maustaste und halte die <Alt> Taste gedrückt. Nun kannst du eine Fläche markieren die du im Raum wie ein Tile setzen kannst. Tiles können auf Layern mit verschiedenen Tiefen gesetzt werden. Unten siehst du die aktuelle Tiefe. Der Standard ist 1000000 welcher normalerweise hinter allen Instanzen liegt. Deshalb befinden sich die Instanzen vor den Tiles. Du kannst den "Add" Button benutzen neue Ebenen (Layer) hinzuzufügen, jede mit einer anderen Tiefe. Negative Tiefen können genutzt werden, um Tiles vor den Instanzen zu setzten. Wenn du auch den Objekten verschiedene Tiefen gibst kannst du sie zwischen verschiedene Ebenen setzen. Wenn du "Delete" drückst wird die aktuelle Ebene mit allen Tiles darauf gelöscht. (Es muss immer mindestens eine Ebene vorhanden sein.) Wenn du "Change" drückst kannst du die Tiefe der Ebene verändern. Wenn du einer Ebene dieselbe Tiefe wie eine Andere angibst, werden sie kombiniert. Das Benutzen solcher Kacheln ist ein mächtiges Instrument, dass so oft wie möglich verwendet werden soll. Es ist viel schneller als Objekte zu benutzen und die Kachel-Bilder werden nur einmal gespeichert. So kannst du große Räume machen mit sehr geringem Speicherverbrauch.

21.3 Views

Zuletzt gibt es ein Register "views". Dies gibt dir einen Mechanismus in die Hand, um bestimmte Teile deines Raumes an verschieden Plätzen, am Bildschirm zu zeichnen. Es gibt viele Anwendungen für Views. Zuallererst in einigen Spielen willst du nur einen Teil des Raumes immer zeigen, z.B. im den meisten Plattformspielen, folgt der Sichtbereich dem Hauptcharakter. In Zwei-Spieler-Spielen willst du des öfteren einen geteilten Bildschirm. Drittens wird in manchen Spielen ein Teil des Bildschirms gescrollt, während der andere Teil stehen bleibt (z. B. eine Statusanzeige). Das kann leicht mit dem Game Maker gemacht werden. Wenn du das Register "views" anklickst, werden folgende Informationen gezeigt. Oben gibt es eine Auswahlbox "Enable the use of Views". Wenn Views gewünscht werden, muss das angekreuzt werden. Darunter siehst du eine Liste mit 8 Views. Darunter kannst du Informationen über das aktuelle View sehen. Zuerst muss du angegeben, ob das View sichtbar ist, wenn der Raum gestartet wird. Gehe sicher das einer sichtbar ist! Sichtbare Views werden fettgedruckt gezeigt. Danach kannst du ein Feld
des Raumes, welches gezeigt werden soll, angeben. Du gibst die linke und obere Position, die Breite und die Höhe des Views an. Darunter kannst du die Position des Views am Bildschirm angeben.



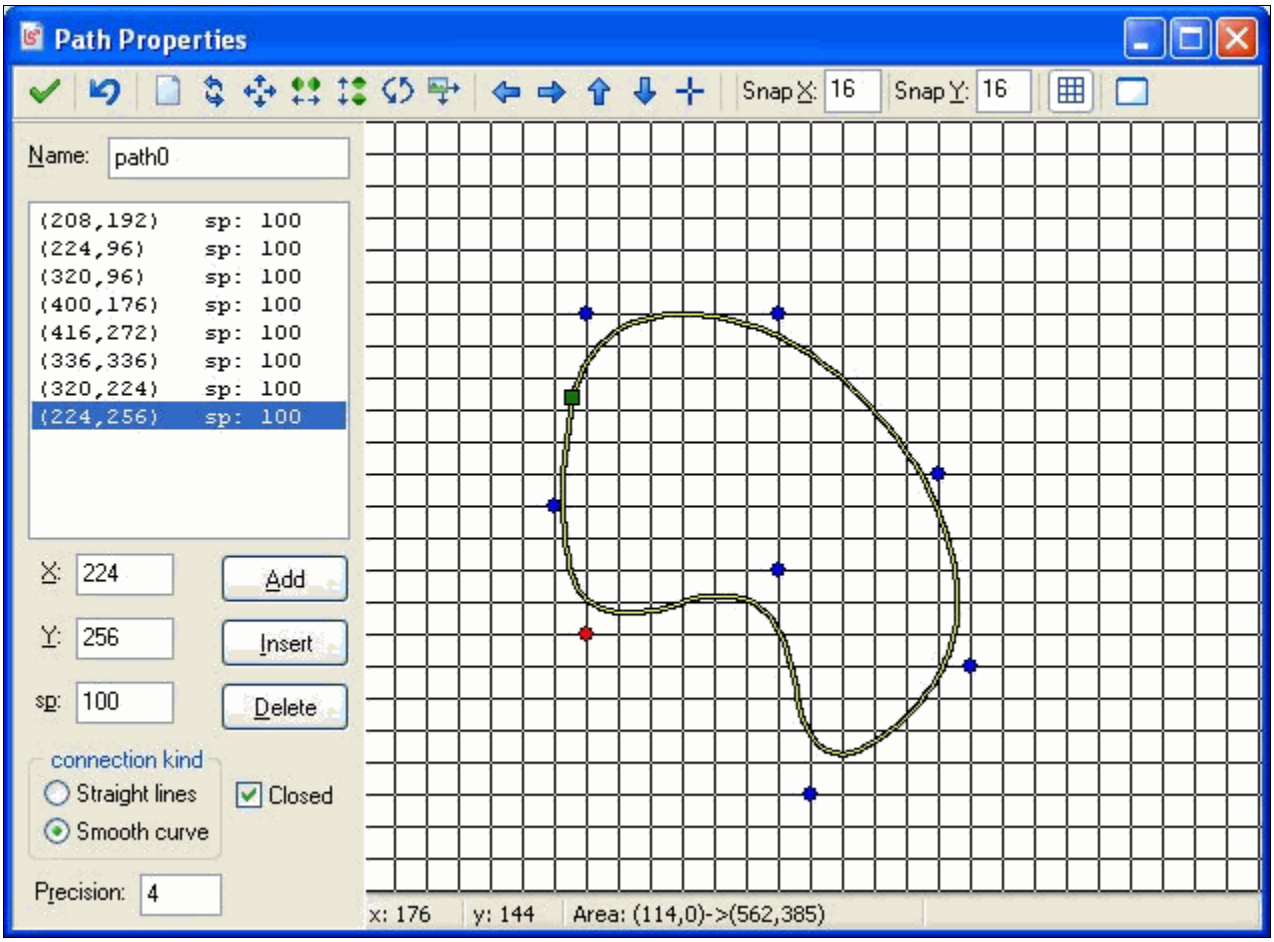
Wie oben erwähnt, willst du häufig ein bestimmtes Objekt verfolgen. Ganz unten kannst du das Objekt angeben. Wenn mehrere Instanzen dieses Objektes vorhanden sind, wird nur das erste in der Sicht verfolgt. (Im Code kannst du diese bestimmte Instanz, der gefolgt werden soll, angeben). Normalerweise soll der Charakter fähig sein zu gehen, ohne dass die Sicht geändert wird. Nur wenn der Charakter nahe am Rand ist, soll sich die Sicht ändern. Du kannst die Grösse des Randes angeben, welcher sichtbar um das Objekt sein soll. Zuletzt kannst du die Geschwindigkeit der Sicht angeben. Das bedeutet, dass sich die Spielfigur ausserhalb des Bildschirmes bewegen kann, aber es gibt ein geschmeidigeres Spiel. Benutze –1, wenn du die Sicht sofort ändern willst.

22. Pfade

In fortgeschrittenen Spielen soll öfters eine Instanz einen bestimmten Pfad folgen. Obwohl so etwas mit Timer oder Code möglich wäre, ist das ziemlich kompliziert. Pfade sind dafür besser geeignet. Die Idee ist einfach. Du bestimmst einen Pfad durch Zeichnen. Nun kannst du eine Aktion erstellen, z. B. im Objekt-Ereignis „Create“. Du befiehlst dem Objekt, einem bestimmten Pfad zu folgen. Dieses Kapitel beschreibt diesen Vorgang im Detail.

22.1 Pfade definieren

Um einen Pfad deinem Spiel hinzuzufügen, wähle **Add Path** vom Add-Menü. Das folgende Dialogfenster wird geöffnet (in dem Bespiel haben wir bereits einen kleinen Pfad hinzugefügt).



Oben links im Fenster kannst du den Namen des Pfades setzen. Darunter findest du die Punkte, die den Pfad bestimmen. Jeder Punkt hat eine Position und eine Geschwindigkeit (speed, kurz sp:).Je nachdem, wie du den Pfad nutzt, ist die Position entweder absolut, d.h. die Instanz, für die du später den Pfad nutzt, wird ihm genau an dieser Stelle folgen, oder relativ, d.h. die Instanz startet immer mit der ersten Position des Pfades und folgt dem Pfad von da an. Die Geschwindigkeit soll wie folgt verstanden werden: Ein Wert von 100 ist die originale Geschwindigkeit der Instanz für den Pfad. Eine niedrigerer Wert verringert die Geschwindigkeit, ein höherer Wert erhöht sie (er zeigt also den Prozentsatz der aktuellen Geschwindigkeit). Die Geschwindigkeit wird zwischen den Punkten interpoliert, sie kann also etwas abweichen. Um einen Punkt hinzuzufügen drücke den Button **Add**. Eine Kopie des aktuell gewählten Punktes wird gemacht. Du kannst jetzt die aktuelle Position und Geschwindigkeit ändern, indem du Werte überschreibst. Immer wenn du einen Punkt in der Liste wählst, kannst du auch dessen Werte ändern.

Drücke **Insert** um einen neuen Punkt vor dem aktuellen einzufügen, und **Delete** um den aktuellen Punkt zu löschen.
Rechts im Dialogfenster siehst du den aktuellen Pfad. Der rote Punkt zeigt den aktuell gewählten Kontrollpunkt. Die blauen Punkte sind die anderen Kontrollpunkte. Das grüne Quadrat zeigt die Position, an der Pfad startet. Du kannst den Pfad auch mit der Maus ändern. Klick irgendwo in das Bild, um einen Punkt hinzuzufügen. Klicke auf einen existierenden Punkt und ziehe in, um die Position zu ändern. Wenn du die <Shift> Taste während des Klickens auf einen Punkt hältst, fügst du einen Punkt ein. Schließlich kannst du die rechte Maustaste verwenden, um Punkte zu entfernen. (Auf diese Art kannst du aber nicht die Geschwindigkeit ändern.) Normalerweise rasten die Punkte im Gitter ein. Die Gitter-Definitionen kannst du an der oberen Werkzeug-Leiste ändern. Hier kannst du auch bestimmen, ob das Gitter sichtbar sein soll oder nicht. Wenn du einen Punkt exakt positionieren willst, halte die <Alt>-Taste beim Hinzufügen oder Bewegen gedrückt.

22.2 Pfade (paths) Objekten zuweisen

Um einen Pfad der Instanz eines Objektes zuzuweisen, kannst du die “path action” (Pfad-Aktion) in einem Ereignis plazieren, wie z.B. im Erstellungs-Ergeignis (create event). Für diese Aktion musst du einen Pfad aus dem herunterklappenden Menü auswählen. Es gibt auch weitere Werte, die du festlegen kannst.

Du musst den Pfad, dem gefolgt werden soll, sowie die Geschwindkeit in Bildpunkten pro Schritt festlegen. Wenn die Geschwindikeit positiv ist, startet die Instanz am Anfang des Pfades. Wenn sie negativ ist, startet sie am Ende. Bedenke, dass wenn du den Pfad festgelegt hast, du die aktuelle Geschwindigkeit relativ zu dieser festgelegten Geschwindkeit festlegst. Es gibt auch eine Aktion, mit der du die Geschwindigkeit ändern kannst, in der der Pfad ausgeführt wird. Du kannst davon Gebrauch machen um z.B. die Instanz langsamer oder schneller werden zu lassen während sie dem Pfad folgt. Beachte, dass die normale Geschwindkeit ignoriert wird (eigentlich wird sie auf 0 gesetzt), wenn ein Pfad ausgeführt wird. Weiterhin haben Dinge wie Gravitation und Reibung keinen Einfluss auf die Bewegung auf dem Pfad.

Als nächstes legst du das Schluss-Verhalten (end behavior) fest, das bestimmt, welches passieren soll, wenn das Ende des Pfades erreicht ist. Du kannst die Bewegung stoppen und den Pfad damit Beenden. Du kannst den Pfad auch von vorne beginnen lassen, die Instanz springt dann zurück zu dem Punkt an dem der Pfad gestartet wurde, und führt den Pfad erneut aus. Eine dritte Option erlaubt es dir, den Pfad von der derzeitigen Position aus neuzustarten, die Instanz folgt dann wieder dem Pfad, aber nun von seiner jetztigen Position aus (Das ist das Selbe, wie wenn der Pfad in sich geschlossen ist). Als Letztes kannst du wählen, dass die Bewegung rückwärts abläuft, und die Instanz damit auf dem Pfad zurückbewegt wird. Beachte, dass am Ende des Pfades zusätzlich das „end of path event“ (Ende des Pfades-Ereignis) ausgeführt wird; siehe unten.

Schließlich kannst du einstellen ob der Pfad absolut oder relativ sein soll. Ein absoluter Pfad wird an der Stelle ausgeführt, an der er festgelegt wurde. Die Instanz wird an die Startposition gesetzt und von dort aus bewegt (bzw. von der Endposition aus wenn die Geschwindkeit negativ ist). Das ist nützlich wenn du z.B. eine Rennstrecke und einen Pfad dafür angelegt hast. Wenn du “relative” auswählst fängt die Instanz dem Pfad von ihrer derzeitigen Position aus zu folgen. Das ist nützlich wenn die Instanz eine Bewegung an Ort und Stelle ausführen soll. Zum Beispiel Raumschiffe in einem „Space Invaders“-Spiel können einen bestimmten Zug von ihrer Position aus machen.

Wenn du die Instanz auf eine andere Position auf dem Pfad plazieren willst, kannst du die Aktion “set the path position” (Setze die Pfad-Position) benutzen. Eine Pfad-Position liegt immer zwischen 0 und 1, wobei 0 für die Startposition und 1 für die Endposition des Pfades steht. Beachte, dass während jedem Schritt die Richtungs-Variable (direction) automatisch in korrekte Richtung eingestellt wird. So kannst du diese Variable benutzten um z.B. das passende Einzelbild zu wählen.

Wenn du Skripte oder pieces of code benutzt hast du mehr Kontrolle über die Möglichkeiten der Ausführung des Pfades. Es gibt eine Funktion um einen Pfad für eine Instanz zu starten. Die Variable `path_position` (Pfad-Position) bestimmt die derzeitige Position auf dem Pfad (zwischen 0 und 1, wie oben bereits beschrieben). Die Variable `path_speed` (Pfad-Geschwindigkeit) bestimmt die Geschwindkeit auf dem Pfad. Eine Variable `path_scale` (Pfad-Skalierung) kann benutzt werden um den Pfad zu skalieren. Ein Wert von 1 ist die normale Größe. Ein größerer Wert macht den Pfad größer; Ein kleinerer Wert macht ihn kleiner. Die Variable `path_orientation` (Pfad-Ausrichtung) bestimmt die Ausrichtung in der der Pfad ausgeführt wird (in Grad gegen den Uhrzeigersinn). Dies erlaubt dir den Pfad in einer anderen Ausrichtung (z.B. von oben nach unten statt von links nach rechts). Weiterhin gibt es eine Variable um das Endverhalten (end behavior) zu bestimmen. Schließlich gibt es noch eine Menge Funktionen um die Eigenschaften von Pfaden zu erfragen (z.B. die x- und y-Koordinate an bestimmten Positionen), sowie Funktionen um Pfade zu erstellen. Es gibt sogar Funktionen, die kollisionsfreie Pfade erstellen, um ein bestimmtest Ziel zu erreichen. Schaue in spätere Kapitel über GML für Details.

Du fragst dich vielleicht, was passiert, wenn die Instanz mit einer anderen Instanz kollidiert, während sie einem Pfad folgt. Grob gesagt passiert das gleiche wie wenn die Instanz sich mit einer Geschwindkeit bewegt. Wenn dort eine feste (solid) Instanz ist, wird die Instanz auf ihre voherige Position zurückgesetzt. Wenn beide Instanzen fest (solid) sind werden sie auf ihre neuen Positionen gesetzt . Als nächstes werden die Kollision(s)-Ereignis(se) durchgeführt und es wird kontrolliert ob die Kollision sich aufgelöst hat. Falls nicht und die andere Instanz ist fest (solid), stoppt die Instanz, sowie sie sollte (angenommen ein Kollisions-Ereignis wurde definiert). Zusätzlich, wird die Vaiable `path_position` (Pfad-Position) nicht erhöht. Wenn die blockierende Instanz verschwindet wird die Instanz weiter ihrem Pfad folgen. Um die Kollisionen selbst zu handhaben kann die Variable `path_positionprevious` (Vorherige Pfad-Position) nützlich sein. Sie beinhaltet die vorherige Position für den Pfad und du kannst die Pfad-Position gleich dieser Variable setzen, um Vorwärtskommen auf dem Pfad zu vermeiden.

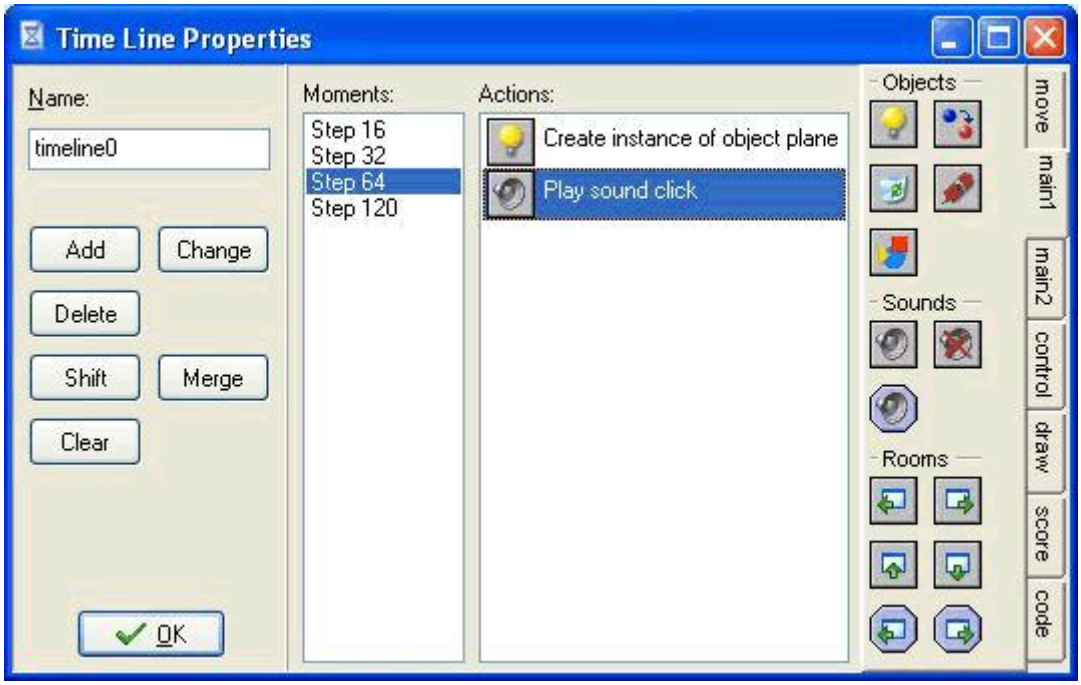
22.3 Das Pfad-Ereignis (path event)

Wie oben bereits beschrieben, kannst du festlegen was passieren soll wenn die Instanz das Ende des Pfades erreicht. An diesem Punkt tritt das Ende des Pfades-Ereignis auf (End of path event). Du findest es unter den “Other events” (Andere Ereignisse). Hier kannst du dann Aktionen plazieren. Vielleicht möchtest du die Instanz dann zerstören, oder sie einem neuen (anderen) Pfad folgen lassen.

23. Zeitleisten

In vielen Spielen müssen bestimmte Dinge zu einem gewissen Zeitpunkt geschehen. Du kannst dies auch mit Alarm-Events machen, aber wenn es zu kompliziert wird, funktioniert das nicht mehr. Eine Zeitleiste-Ressource ist dafür gedacht. In einer Zeitleiste gibst du an, welche Aktion zu einem bestimmten Zeitpunkt geschehen soll.
Du kannst alle Aktionen, die für die verschiedenen Ereignisse vorhanden sind, benutzen. Wenn eine Zeitleiste erstellt ist, kannst du sie mit einer Instanz eines Objekte verbinden. Diese Instanz führt die Aktion zum angegebenen Zeitpunkt aus. Lass' mich ein Beispiel zeigen. Vorrausgesetzt du willst eine Wache erstellen. Diese Wache sollte 20 Zeitschritte lang nach links, 10 nach oben, 20 nach rechts, 10 nach unten gehen und dann stehen bleiben. Dafür musst du eine Zeitleiste erstellen, beim Start setzt du die Bewegung nach links. Beim Zeitpunkt 20 setzt du die Bewegung nach oben, beim Zeitpunkt 30 die Bewegung nach rechts, beim Zeitpunkt 50 eine Bewegung nach unten und beim Zeitpunkt 60 hälst du die Bewegung an. Nun kannst du die Zeitleiste mit der Wache verbinden und die Wache wird genau das Geplante machen. Du kannst auch eine Zeitleiste benutzen, um die Steuerung des Spiel globaler zu halten. Erstelle ein unsichtbares Kontrollerobjekt, erstelle eine Zeitleiste, welche an bestimmten Zeitpunkten Feinde erstellt, und weise es dem Kontrollerobjekt zu. Wenn du anfängst, damit zu arbeiten, wirst du feststellen, dass dies ein sehr wirkungsvolles Konzept ist.

Um eine Zeitleiste zu erstellen wähle "Add Time Line" vom "Add Menu". Das folgende Fenster wird geöffnet:



Es sieht so ähnlich wie das Objekteigenschaftsfenster aus. Links kannst du den Namen sehen und es sind Buttons zum Hinzufügen und Verändern von Zeitleisten vorhanden. Daneben ist eine Liste von Zeitpunkten. Diese Liste gibt die Zeitpunkte in Zeitschritten an, wenn die zugeteilte Aktion stattfinden soll. Daneben werden die Aktionen des gewählten Zeitpunktes aufgelistet und ganz rechts ist das ganze Set von den Aktionen verfügbar. Um einen Zeitpunkt hinzuzufügen drücke den "Add Button". Gib den Zeitpunkt an(das ist die Anzahl der Schritte seit die Zeitleiste gestartet wurde). Nun kannst du die Aktion eintragen in die Liste genauso wie im Event der Objekte.
Es gibt auch einen Button, um den gewählten Zeitpunkt zu löschen, den Zeitpunkt zu ändern und zum Löschen der Zeitleiste. Zuletzt gibt es zwei spezielle Schaltflächen.

Mit dem "Merge Button" kannst du alle Zeitpunkte eines Zeitraumes in einen Zeitpunkt vermischen. Mit dem Shift Button kannst du alle Zeitpunkte in einem Zeitraum durch Angabe eines Wertes nach vorne oder hinten verschieben. Gehe sicher, keine negativen Zeitmomente zu erschaffen. Diese werden nie ausgeführt.

Es gibt zwei Aktion die sich auf Zeitleisten beziehen.

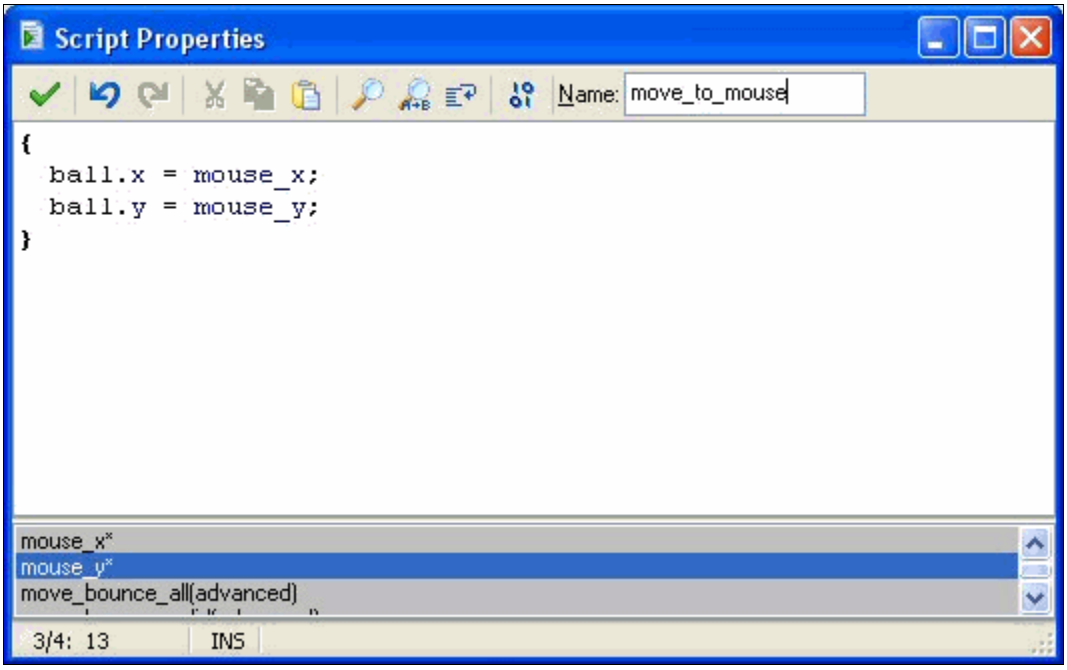
 **Set a time line**
Mit dieser Aktion setzt du die bestimme Zeitleiste für eine Instanz eines Objektes. Du gibst die Zeitleiste und die Startposition in der Zeitleiste (0 zum Beginnen) an. Du kannst diese Aktion auch für das Ende einer Zeitleiste benützen, einfach "No Time Line" als Wert wählen.

 **Set the time line position**
Mit dieser Aktion kannst du die Position in der aktuellen Zeitleiste (beide absolut oder relativ) bestimmen. Das kann benutzt werden, um bestimmte Teile der Zeitleiste zu überspringen oder bestimmte Teile zu wiederholen. Wenn du also eine Zeitleiste in einer Schleife laufen lassen willst, füge diese Aktion im letzten Moment ein und setze als Position 0. Du kannst dies genauso benutzen um auf irgendetwas wartest. Füge die Test-Aktion hinzu, und wenn nicht diese nicht "wahr" (true) ist, setzte die Zeitleistenposition relativ zu –1.

24. Skripte

Game Maker hat eine eingebaute Programmiersprache. Wenn du einmal mit dem Game Maker vertraut bist, und den vollen Umfang ausreizen willst, ist es vorteilhaft, das Verwenden der Sprache zu erlernen. Für eine komplette Beschreibung siehe Kapitel 29. Es gibt zwei Wege die Sprache zu nutzen. Zuerst kannst du Skripte erstellen. Das sind Teile eines Codes, denen du einen Namen gibst. Diese werden im Ressourcenbaum gezeigt und können in einer Datei gespeichert oder von einer Datei geladen werden. Sie können zum erstellen von Bibliotheken verwendet werden, welche die Möglichkeiten vom Game Maker stark erweitern. Du kannst auch eine Code Aktion in manchen Events einfügen und ein Stück Code dort eingeben. Code Aktionen haben keinen Namen und können keine Argumente verwenden. Auch haben sie das bekannte Auswahlfeld um zu bestimmen für welches Objekt die Aktion ausgeführt wird. Dann gibst du den Code auf dieselbe Weise wie bei Skripten ein. Wir konzentrieren uns jetzt aber auf Skripte in diesem Kapitel.

Wie vorher beschrieben, ist ein Skript ein Stück Code in der eingebauten Programmiersprache, dass einen bestimmten Task(Aufgabe) ausführt. Ein Skript kann einige Argumente beinhalten. Um ein Skript von irgendeinem Event aus auszuführen, kannst du die Skript Aktion benutzen. In dieser Aktion gibt du den auszuführenden Skript an, gemeinsam mit höchstens 5 Argumenten. (Du kannst auch eine Skript im "Piece of Code" ausführen genauso wie du Funktionen ausführst, dann kannst du bis zu 16 Argumente angeben). Wenn das Skript einen Wert zurückgibt, kannst du es als Funktion nutzen um Werte in anderen Aktionen nutzbar zu machen.
Um Skripte in dein Spiel hinzuzufügen, wähle "Add Script" vom "Add Menu". Das folgende Fenster wird geöffnet (im folgende Beispiel haben wir bereits ein kleines Skript erstellt, welcher das Produkt der zwei Argumente zurückgibt).



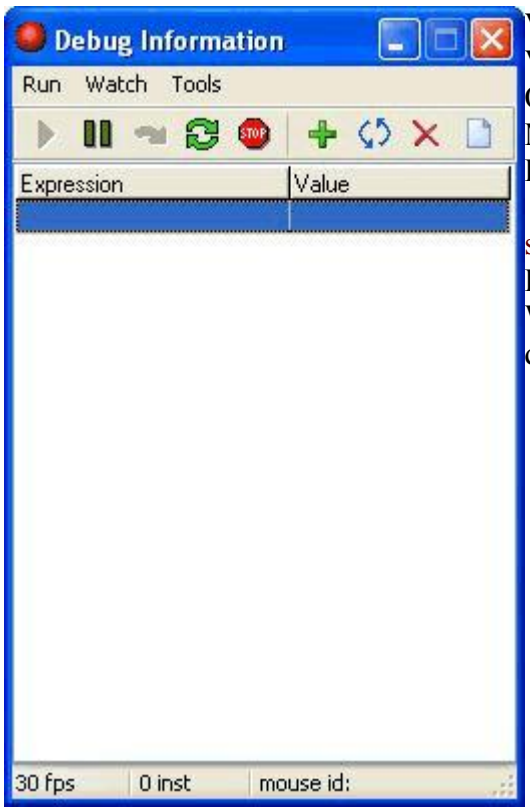
(Das ist der eingebaute Skripteditor. In den Einstellungen kannst du auch einen externen Editor angeben).
Rechts oben kannst du den Namen des Skriptes eingeben. Im kleinen Editor kannst du das Skript eingeben. Beachte das unten eine Liste aller Funktionen, eingebauter Variablen und Konstanten angezeigt wird. Dies hilft dir die zu finden die du brauchst. Per Doppelklick (oder per <Ctrl>-P) kann man sie hinzufügen. Diese Liste kann in den Optionen abgeschaltet werden..
Der Editor hat einige nützliche Eigenschaften, die meisten sind mit Schaltflächen erreichbar. (drück

die rechte Maustaste für zusätzliche Befehle):

- Mehrfaches Rückgängig machen und Wiederholen einer Eingabe entweder pro Tastendruck oder in Gruppen (kann in den Einstellungen geändert werden)
- Automatisches intelligentes Einrücken von Zeilen dass sich nach der vorherigen Zeile richtet (kann in den Einstellungen gesetzt werden)
- Intelligenter Tabulator der bis zum ersten nichtfreien Platz in der vorherigen Zeile springt (kann in den Einstellungen gesetzt werden)
- Benutze <Strg> um gewählte Linien einzurücken und <Shift><Strg> um gewählte Linien nicht einzurücken
- Ausschneiden und Einfügen
- Suchen und Ersetzen
- Benutze <Strg> und <oben>, <unten>, <Bild auf> oder <Bild ab> zum Scrollen ohne die Cursorposition zu ändern
- Drücke F4 um einen Skript oder Ressource, deren Name auf der Cursorposition ist, zu öffnen (funktioniert nicht in der Code Aktion; nur in Skripts)
- Speichern und Laden des Skriptes als Textdatei

Es gibt auch eine Schaltfläche zum Testen ob das Skript korrekt ist. Nicht alle Aspekte können getestet werden zu diesem Zeitpunkt, aber der Syntax deines Skriptes wird getestet, zusammen mit der Existenz der benutzten Variablen.
Wie du wahrscheinlich bemerkt hast, sind Teile des Skripttextes farbig. Der Editor kennt die existierenden Objekte, eingebauten Variablen und Funktionen, usw. Farb-Coding ist sehr hilfreich um Fehler zu vermeiden. Du siehst sofort ob du einen Namen falsch geschrieben hast oder ein Schlüsselwort als Variable verwendet hast. Farb-Coding ist ein bisschen langsam. In den Einstellungen im "File Menu" kannst du Farb-Codung ein- und ausschalten. Hier kannst du auch die Farbe der verschiedenen Komponenten des Programms einstellen.
(Wenn manchmal etwas beim Farb-Coding schief geht, drücke zweimal <F12>, um es aus- und wieder einzuschalten)

Ausserdem kannst du die Schriftart der Skripte und des Codes ändern. Skripte sind sehr nützlich um die Fähigkeiten des Game Maker zu erweitern. Das bedeutet dass du das Design der Skripte sorgfältig erledigen musst. Skripte können in Bibliothek gespeichert werden und deinem Spiel hinzugefügt werden. Um eine Bibliothek zu importieren, benutze den Import scripts vom File Menu. Um Skripte im Form von Bibliotheken zu speichern benutze Export scripts. Skript Bibliotheken sind einfache Textdateien (obwohl sie die Dateierweiterung .gml besitzen). Am besten bearbeite sie nicht direkt, da sie eine spezielle Struktur haben. Manche Bibliotheken mit nützlichen Skripten sind beigefügt. (Um unnötige Arbeit beim Laden des Spieles zu vermeiden, nach dem Importieren der Bibliothek, lösche die Skripte die du nicht benötigst).
Wenn du Skripte erstellst, passieren leicht Fehler. Teste immer die Sripte mit dem entsprechen Button. Wenn ein Fehler während der Ausführung passiert, wird dies mitgeteilt, mit der Angabe der Art des Fehler und des Platzes. Wenn du Dinge genauer betrachten musst, kannst du das Spiel im Debug Mode starten. Nun wird ein Fenster geöffnet wo du viele Informationen deines Spieles sehen kannst.
Im Run Menu kannst du das Spiel pausieren, es Schritt für Schritt laufen lassen und es neu starten. Im Watch Menu kannst du einen Wert oder einen bestimmten Ausdruck beobachten. Benutze Add um einen Ausdruck einzugeben, der Wert wird in jedem Schritt des Spieles ausgegeben. Damit kannst du sehen ob in deinem Spiel alles so passiert, wie du es dir vorgestellt hast. Du kannst viele Ausdrücke beobachten. Du kannst sie speichern für spätere Benutzung (beispielsweise wenn du eine Änderung des Spieles gemacht hast). Im Tools Menu findest du Einträge für noch nähere Informationen. Du kannst eine Liste aller Instanzen im Spiel sehen, du kannst alle globalen



Variablen sehen (gut, die wichtigsten halt) und die lokalen Variablen einer Instanz (entweder benutze den Objektnamen oder die ID der Instanz). Du kannst auch Mitteilungen sehen, welche du von deinem Code mit Funktionen gesendet hast.

show_debug_message(str) Zuletzt kannst du dem Spiel Befehle geben und die Geschwindigkeit des Spieles ändern. Wenn du kompliziertere Spiel machst solltest du lernen mit den Debug Optionen umzugehen.

25. Daten-Dateien

In fortgeschrittenen Spielen benötigst du öfters zusätzliche Dateien, z. B. Dateien die bestimmte Eigenschaften beschreiben, Hintergründe und Sprites welche du während des Spieles laden willst, Filme, DLL Dateien (siehe später) und deinen eigenen Schriftsatz. Du kannst diese Dateien mit deinem Spiel mitgeben aber ist es schöner sie in das Spiel selbst einzubinden. Dafür kann die Ressource Data File eingesetzt werden. Ein solche Ressource beinhaltetet einfach den Inhalt einer Datei. Wenn das Spiel startet wird diese Dateien auf die Platte geschrieben und kann dann im Spiel verwendet werden.

Um eine Daten-Datei hinzuzufügen wähle Add Data File vom Add Menü. Das folgende Fenster wird gezeigt:

Du kannst der Ressource einen Namen geben. Drücke den Button Load Data File um eine Datei in die Ressource zu laden. Du kannst den File Name (Dateiname) wählen welcher, für die Ressource gespeichert muss (kein Pfad ist im Dateinamen erlaubt). Dann kannst du angeben, was geschehen soll, wenn das Spiel startet:
- Don’t Export. Nicht exportieren (Es gibt eine GML Funktion zum Exportieren.)
- To Temporary Folder. Die Datei wird in ein temporäres Verzeichnis des Spieles geschrieben. Das ist nützlich für das Laden von Hintergründen, usw.
- To Working Folder. Die Datei wird in das Verzeichnis, wo das Spiel gespeichert ist, geschrieben. Viele Routinen suchen Dateien, aber du sollst vorsichtig sein, da der Spieler vielleicht das Spiel von einem Read-Only-Medium (Nur Lesen, z. B. CD-ROM) laufen lässt.
- Install a Font. Wenn deine Datei eine Schriftart-Datei ist, kannst du diese Option wählen. Die Datei wird im temporären Verzeichnis gespeichert und nachher als Schriftart installiert welche vom Spiel benützt werden kann.

Zuletzt gibt es folgende Optionen
- Free Data Memory. Wenn gewählt, wird der Speicher, der für die Daten-Dateien benutzt wurde, nach dem Exportieren freigegeben. Das spart Speicher, aber bedeutet dass die Ressource nicht mehr von Funktionen benutzt werden kann.
- Overwrite the File. Wenn gewählt wird die Datei überschrieben, natürlich nur wenn sie bereits vorhanden ist.
- Remove at Game End. Wenn gewählt wird die Datei entfernt wenn das Spiel zu Ende ist. (Bedenke dass Dateien im temporären Verzeichnis auch immer entfernt werden wenn das Spiel zu Ende ist.)

Wenn ein Fehler während des Exportierens auftaucht läuft das Spiel trotzdem weiter. Aber die Datei oder Schriftart ist vielleicht nicht verfügbar.
Daten-Dateien können eine Menge Speicher benötigen, aber sie sind schnell geladen und exportiert.

26. Spielbeschreibungen

Ein gutes Spiel versorgt den Spieler mit Informationen, wie man das Spiel spielt. Diese Informationen werden angezeigt, wenn der Spieler während des Spiels die <F1>-Taste drückt. Um solch eine Spielinformation zu erstellen, klicke doppelt auf "Game Information" (es hat als Symbol ein i) im "resource tree" (links, wo alle Spielbestandteile aufgeführt werden). Ein kleiner eingebauter Editor wird geöffnet, in dem du die Spielinformationen bearbeiten kannst. Du kannst verschiedene Schriftarten, verschiedene Farben und Formate verwenden. Die Hintergrundfarbe kannst du auch bestimmen.
Eine interessante Option im "Format-menu" ist "Mimic Main Form". Wenn du hier einen Haken machst, wird das Help-Fenster an der gleichen Stelle und mit den gleichen Dimensionen angezeigt, wie das Spiel-Fenster.

Es sieht dann so aus, als erschiene der Text im Spiel-Fenster. Wenn du jetzt noch eine passende Hintergrundfarbe wählst, hast du einen schönen Effekt.(Evtl. machst du noch einen kleinen Hinweis am unteren Bildschirm, dass es mit der "esc"-Taste zum Spiel zurück geht.)
Ein guter Ratschlag ist, die Hilfe kurz aber treffend zu gestalten. Natürlich solltest du deinen Namen zufügen, weil du das Spiel erschaffen hast. Alle Beispiele (example games) haben eine Info-Datei über das Spiel und seine Herstellung.
Wenn du eine etwas raffiniertere Help erstellen willst, benutze zum Beispiel "Word" oder "OpenOffice.org" und kopiere dann einfach mit copy&paste (STRG+C;STRG+V) den "Wordinhalt" in den "game information editor" (den eingebauten Editor).

27. Spieloptionen

Es gibt zahlreiche Optionen, welche du für dein Spiel einstellen kannst. Du kannst sie finden, wenn du im Spielkomponentenbaum (linke Bildschirmseite) doppelt auf "Game Options" klickst. Sie sind eingeteilt durch mehrere Registerkarten (einige Optionen sind nur im erweiterten Modus sichtbar):

27.1 Graphics options (Grafikoptionen)

Auf dieser Karte sind einige Optionen, bezüglich des grafischen Erscheinungsbildes deines Spiels, von dir festzulegen. Es ist gewöhnlich hilfreich, hier kurz reinzuschauen, weil die meisten Effekte das Aussehen des Spiels beeinflussen. Erinnere dich daran, dass andere Benutzer/Spieler andere Rechner haben. Vergewissere dich, dass deine Einstellungen auch bei anderen funktionieren.

Start in fullscreen mode (als Vollbild starten)
Wenn es aktiviert ist, läuft das Spiel als Vollbild, wenn nicht, dann im Fenster.

Scale percentage in windowed mode(Skalierungsprozentsatz im Fenster-Modus)
Hier kannst du angeben, ob das Bild im Fenstermodus skaliert werden soll. 100 bedeutet keine Skalierung.
Du brauchst es typischerweise dann, wenn die "sprites" und "rooms" sehr klein sind. Skalierung ist langsamer, aber auf modernen Grafikkarten fällt das nicht mehr ins Gewicht. Verwende keine Werte unter 100, weil Verkleinern doch sehr langsam ist.

Scale percentage in fullscreen mode(Skalierungsprozentsatz im Vollbildmodus)
Wie oben, nur gilt dies für den Vollbildmodus.

Only scale when there is hardware support(nur mit Hardwareunterstützung skalieren)
Wenn es aktiviert ist, wird nur skaliert wenn die Hardware auch mitspielt. Leider behaupten manche Grafikkarten, sie könnten es und brechen dann aber doch ein.

Don't draw a border in windowed mode(rahmenlos im Fenstermodus)
Fest legen, ob ein Rahmen um ein Fenster gezeichnet werden soll; mit Titelleiste und so.

Don't show the buttons in the window caption(keine Knöpfe darstellen)
Wenn aktiviert, werden keine Schaltflächen für minimieren, schliessen etc. angezeigt.

Wait for vertical blank before drawing(vorm Zeichnen auf Rasterstrahl warten)
Der Bildschirminhalt deines PCs wird ein paar mal pro Sekunde aufgefrischt (normalerweise so 50 bis 100 mal). Nachdem der Bildschirm dargestellt wurde, gibt es eine sog. "senkrechtes Aus". ~Erklärung: Das Monitorbild wird Zeilenweise von links oben nach rechts unten aufgebaut. Wenn der Elektronen-/Rasterstrahl unten rechts angekommen ist, muss er ja wieder zurück zur Anfangsposition, wobei er ausgeblendet wird), wo nix auf dem Bildschirm geschieht. Wenn du den Bildschirm fortlaufend zeichnest, wird ein Teil des Bildes erst angezeigt und dann beim nächsten Bildschirm der Rest des Bildes - was einen schlechten optischen Eindruck hinterlässt. Wenn du auf diese Lücke wartest, bevor du ein neues "Frame" (Einzelbild) darstellst, verschwindet dieses Problem von selbst. Der Nachteil ist, dass das Programm manchmal auf diese Lücke warten muss, was aber kaum merklich bremst.

Display the cursor(darstellen des Mauszeigers)
Hier kannst du angeben, ob der Mauszeiger sichtbar oder unsichtbar sein soll. Ausgeschaltet sieht's besser und schneller aus. Du kannst deinen eigenen Mauszeiger ganz leicht im GM erstellen.

Display the caption in fullscreen mode(Titelleiste im Vollbild)
Wenn hier ein Haken ist, wird im Vollbildmodus eine kleine Box angezeigt, worin der Titel, der Punktestand und die Anzahl der Leben dargestellt wird. Es ist normalerweise schöner, wenn du selber etwas entwirfst und an geeigneterer Stelle einfügst.

Freeze the game when the form looses focus(Spiel einfrieren, wenn Fenster nicht aktiv)
Friert das Spiel ein, solange es nicht aktiv ist (wenn beispielsweise einen anderes Programm gestartet wird).

27.2 Resolution (Auflösung)

Hier kannst du festlegen in welcher Auflösung dein Spiel laufen soll.

Set the resolution of the screen(Bildschirmauflösung festlegen)
Der Bildschirm hat bestimmte Auflösungen. Drei Aspekte spielen hier eine Rolle: Pixelanzahl (horizontal und vertikal) auf dem Bildschirm, Anzahl der Bits für Farbdarstellung, und die Bildwiederholfrequenz. Gewöhnlich ändert der Game Maker diese Einstellungen nicht, was bedeutet, er verwendet die vorgegebenen Einstellungen des Rechners, worauf das Spiel läuft. Dies kann zu schlechter Grafik führen. Wenn deine Räume klein sind, der andere aber zum Beispiel eine hohe Auflösung verwendet, findet für ihn das Spiel in einem winzigen Fenster statt. Du kannst dieses Problem lösen, indem du Vollbilddarstellung und Skalierung verwendest, was die Sache evtl. verlangsamt. Die beste Lösung ist, Game Maker mitzuteilen, welche Auflösung verwendet werden soll, wenn das Spiel gestartet wird. Nach dem Spiel wird wieder umgeschaltet. Um dies zu tun, mach hier einen Haken hin.
Eine Anzahl von zusätzlichen Optionen erscheint.
Zuerst kannst du die Farbtiefe bestimmen (16 oder 32 bit; normal ist 16 bit das beste; auf Windows'98 läuft alles in 16 bit Farbtiefe, auch wenn man 32 angibt). Zweitens die Bildschirmgröße (320x240, 640x480, 800x600, 1024x768, 1280x1024, oder 1600x1200). An dieser Stelle eine Warnung. Wenn du eine kleine Auflösung wählst (z.B. 320x240) verkleinert Windows alle Fenster, wenn du das Spiel startest. Das könnte zu Problemen mit anderen laufenden Anwendungen führen (ja sogar mit dem GM selbst). (Verwende den Exklusiven Modus, um es zu verhindern.) Wenn du die beiden grössten auswählst, solltest du bedenken, dass nicht alle Rechner so etwas unterstützen. Du kannst auch angeben, die Bildschirmauflösung nicht zu wechseln. Abschliessend kannst du noch die Bildwiederholfrequenz einstellen (60, 70, 85, 100; Wenn die von Dir angegebene zu hoch ist, wird die voreingestellte verwendet; Du kannst auch angeben, die Voreingestellte zu verwenden).

Folgende Optionen kommen noch vor:

Use exclusive graphics mode(Exklusiven Grafikmodus verwenden)
Im exklusiven Modus hat das Spiel volle Kontrolle über den Bildschirm. Keine andere Anwendung funktioniert dazwischen. Das macht die Grafikdarstellung meist etwas schneller und erlaubt einige Spezialeffekte (gamma). Wenn du sichergehen willst, dass der Rechner mit der richtigen Auflösung läuft, verwende den Exklusiven Grafikmodus. Eine Warnung: Im exklusiven Modus können keine anderen Fenster dargestellt werden. Das heißt, dass auch keine Nachrichtenfenster oder andere pop-ups eingeblendet werden können.

Wenn gewöhnlich dann was schief geht, schmiert das Spiel ab und der Screen hängt, was zu einem RESET verleitet. Vergewissere dich also, das dein Spiel Top ist. Im exklusiven Modus kann die Debug Funktion nicht verwendet werden!

27.3 Other Settings (Andere Optionen)

Hier können eine Reihe von Optionen bestimmt werden, zuerst die Tastenoptionen:

Let <Esc> end the game(Esc = Spielende)
Wenn hier einen Haken dran ist, wird das Spiel beendet, wenn der Spieler auf ESC drückt. Fortgeschrittenere Spiele wollen so etwas gewöhnlich nicht, weil sie evtl. noch Spielstände sichern oder ähnliches...
Auf das Kreuzchen des Fensters zu klicken generiert übrigens auch ein ESC-event

Let <F1> show the game information(mit F1 die Spielinfos anzeigen)
Funktioniert nicht im Exklusiven Modus.

Let <F4> switch between screen modes (mit F4 zwischen den Auflösungen umschalten)
Wechselt - wenn Häkchen drin - per F4 zwischen Vollbild und Fenster Modus (nicht im exklusiven Modus).

Let <F5> and <F6> load and save the game (F5=speichern und F6=Laden)
Der Spieler kann mit F5 bzw. F6 das Spiel laden und speichern.

Die Prozesspriorität kann auch gesetzt werden, diese Option legt fest wieviel Prozessorleistung dem Spiel zugewiesen wird. Im normalen Modus versucht das Betriebssystem die Prozessorleitsung gleichmässig zu verteilen. Je höher der Wert dest, höher die Prozessorleistung die dem Spiel zugeteilt wird und dadurch läuft es flüssiger und schneller. Aber andere Prozesse bekommen weniger Leistung (auch Windowsprozesse, so kann es vorkommen, das sich nicht einmal mehr die Maus bewegen kann.).

27.4 Loading options (Ladeoptionen)

Hier kannst du einstellen, was passieren soll, wenn das Spiel geladen wird. Zuerst kannst du angeben, ob du ein eigenes Lade-Bild verwenden willst. Zweitens kannst du festlegen, ob ein Fortschrittsbalken angezeigt werden soll; unten am Ladebild. Du hast drei Möglichkeiten hier. Entweder wird kein Balken angezeigt, oder der voreingestellte oder du kannst zwei Bilder bestimmen: Das Hintergrundbild des Ladebalkens und den Vordergrund. Sie werden skaliert, um den Größenanforderungen zu entsprechen. (Du musst beide Bilder angeben - eins reicht nicht).
Du kannst ein Icon für "alleinstehende" Spiele festlegen. Nur 32x32 Icons. Wenn du was anderes versuchst, wirst du gewarnt.
Schliesslich kannst du noch die einzigartige "Game id" ändern. Diese id wird verwendet, um Highscores abzuspeichern und Spielstände zu sichern. Wenn du eine neuere Version deines Spiels rausbringst, möchtest du nicht die alten Highscores übernehmen, dann solltest du diese Nummer ändern.

27.5 Konstanten

In diesem Tab kannst du globale Konstanten definieren, die in allen Skripten, Codes oder Aktionen verwendet werden können. Jede Konstante hat einen Namen und einen Wert. Für den Namen gelten die gleichen Regeln wie für Variablen. Er muss also aus Buchstaben, Ziffern und Unterstrichen bestehen, darf aber nicht mit einer Ziffer beginnen. Du solltest alle Konstanten unterscheidbar machen. Eine übliche Konvention ist, nur Großbuchstaben und Unterstriche zu verwenden.

Der Wert einer Konstante sollte ein konstanter Ausdruck sein. Das ist entweder eine Zahl oder eine Zeichenkette (in Anführungszeichen) oder ein Ausdruck. Der Ausdruck wird ausgewertet, bevor irgendetwas anderes im Spiel passiert. Deswegen kann er nicht den aktuellen Raum, Instanzen oder Skripte referenzieren. Er kann aber die eingebauten Konstanten und Ressourcennamen enthalten.

Du kannst Konstanten mit "Add" einfügen und mit "Delete" löschen. Du kannst den Name oder Wert einer Konstante ändern, indem du ihn anklickst. Es gibt auch Buttons, um alle Konstanten zu löschen oder nach dem Namen zu sortieren.

27.6 Error options(Fehlerbehandlung)

Stelle hier ein, was mit Fehlermeldungen geschieht.

Display error messages (Anzeigen von Fehlermeldungen)
Ausser: Im exklusiven Modus, werden hier die Fehlermeldungen dem Spieler auch gezeigt. In der Final-Version stell' es besser ab.

Write error messages to file game_errors.log(Fehlermeldungen in Datei schreiben)
Wenn aktiviert, werden die Fehlermeldungen in die Datei "game_errors.log" im Spielverzeichnis geschrieben.
Abort on all error messages(Spielabbruch bei allen Fehlern)
Manche Fehler sind schlimm andere weniger. Wenn hier ein Haken ist, werden kleine Fehler nicht mehr ignoriert, sondern führen, genau wie heftige Fehler zum Abbruch des Spiels. Wenn du dein Spiel mal verteilst mach hier einen haken rein.

Treat uninitialized variables as 0(nicht initialisierte Variablen mit Wert 0 versehen)
Ein häufiger Fehler ist es Variablen zu benutzen, bevor ihnen ein Wert zugewiesen wurde. Manchmal ist es schwierig zu verhindern. Wenn du hier nen Haken machst, werden solche uninitialisierten Variablen nicht mehr gemeldet und mit dem Wert 0 versehen. Aber Vorsicht: Es kann bedeuten, dass du Tippfehler übersiehst!

27.7 Info options(Informationsoptionen)

Gib hier den Spielautoren an (meistens du), die Version des Spiels, ein paar Infos. Das letzte Änderungsdatum wird angezeigt, was hilfreich ist, wenn man mit mehreren an einem Projekt arbeitet. Diese Info gibt es nicht, wenn das Spiel läuft.

28. Gedanken zur Geschwindigkeit

Wenn du komplizierte Spiele entwirfst, willst du sie wahrscheinlich so schnell ablaufend, wie möglich machen. Obwohl der Game Maker schon sein Bestes gibt, um dein Spiel schnell zu machen, hängt ne Menge davon ab, wie du dein Spiel gestaltest. Ausserdem ist es ziemlich einfach ein Spiel zu machen, was viel Speicher verbraucht. In diesem Kapitel gibts ein paar Tipps, wie du dein Spiel schneller und kleiner machen kannst.

Zuallererst betrachte sorgfältig deine "sprites" und "backgrounds", welche du verwendest. Animierte "sprites" brauchen viel Speicherplatz und das Zeichnen vieler "sprites" kostet viel Zeit. Mach deine "sprites" so klein, wie möglich. Entferne jedes unsichtbare Gebiet um es herum (dafür gibt es ein Befehl im "sprite editor"). Entscheide wohlüberlegt, welches "sprite" ins VIDEO-RAM kommt und welche nur bei Bedarf geladen werden. Das gleiche gilt für "Background"-Bilder. Gewöhnlich lädt man sie bei Gebrauch und - besonders wenn sie gross sind - speichert man sie nicht im Video-Speicher. Wenn du einen abdeckenden Hintergrund verwendest, schalte die Hintergrundfarbe ab.

Wenn du den Vollbildschirm oder den Exklusiven-Modus verwendest, vergewissere dich, dass die Grösse des "room"/"window" niemals grösser als die Bildschirmgröße ist. Die meisten Grafikkarten sind effizient im grösser Skalieren von Bildern - beim Verkleinern schwächeln sie ab! Zeichne vorzugsweise so wenig andere Sachen - ausser Sprites - wie es geht. Das ist langsam. Wenn du sie brauchst, zeichne sie am besten direkt hintereinander. Wann immer möglicher, schalte den "cursor" ab - er bremst die Grafikdarstellung. Sei auch mit dem Gebrauch von "views" sparsam. Für jeden "view", wird der Raum neu gezeichnet (Zeitfaktor!).

Neben den Grafiken gibt es noch andere Aspekte, welche die Spielgeschwindigkeit beeinträchtigen. Verwende so wenige Instanzen, wie möglich. Besonders nicht mehr benötigte Instanzen, sollten zerstört werden (z.B. wenn sie den Raum verlassen). Vermeide viel "Arbeit" im "step" und "draw"-Event von Instanzen. Oftmals müssen Dinge nicht jeden Schritt überprüft werden. Die Interpretation des Programmcodes ist vernünftig schnell, aber halt nur interpretiert. Auch sind manche Funktionen sehr Zeitintensiv; besonders, wenn alle Instanzen geprüft werden (wie in der "bounce"-action). Überlege dir, wo du die Kollisionsereignisse behandelst. Du hast gewöhnlich 2 Möglichkeiten. Objekte, welche kein Kollisionsereignis haben werden schneller behandelt, deshalb solltest du sie vorzugsweise in solchen Objekten behandeln, von denen wenige Instanzen vorhanden sind.
Sei vorsichtig, bei Gebrauch großer Soundfiles. Sie verwenden viel Speicher und lassen sich schlecht Komprimieren. Prüfe deine SFX, ob du sie nicht runtersamplen kannst.
Abschliessend lässt sich noch sagen, wenn du willst, dass viele Leute dein Spiel spielen können, dann teste es auch auf älteren Rechnern.

29. GML im Überblick

(Anm.: nicht wörtlich übersetzt)

Variablen

Variablen sind Bereiche im Arbeitsspeicher, in denen Informationen gespeichert werden können. Entweder als Zahl (value), oder als Zeichenkette(string). In GML gibt es sehr viele eingebaute Variablen. Einige sind "global" , wie z.B. mouse_x und mouse_y, welche die Mausposition angeben, und andere sind "lokal" d.h. dass sie nur in der Instanz, in der das Programm ausgeführt wird, vorhanden sind, wie z.B. x und y, welche die Position der Instanz angeben. Der Variablenname muss mit einem Buchstaben beginnen und darf nur Buchstaben, Zahlen und das Unterstrichsymbol "_" enthalten.(die maximale Länge beträgt 64 Zeichen). Wenn ihr eine neue Variable erstellt ist sie lokal zu der derzeitigen Instanz.

Assignments

Ein Assignment weist der Variable einen Wert zu. Es muss in dieser Form sein:<Variable> = <Ausdruck> Ein Ausdruck kann eine normale Zahl (z.B. 5.8 !Achtung ihr dürft kein "," Komma als Dezimaltrennzeichen benutzen. Game Maker ist ein englisches Programm, deshalb müsst ihr einen "." Punkt verwenden!) aber auch komplizierter sein. Ihr könnt z.B. einen Wert zu dem derzeitigen Wert der Variable dazuzählen, indem ihr "+=" benutzt. Ferner könnt ihr subtrahieren durch "-=", multiplizieren durch "*=" und dividieren durch "/=".

Ausdrücke

Ausdrücke können Zahlen, Zeichenketten zwischen Anführungsstrichen (entweder die einzelnen ' oder die doppelten ") (z.B. 'Hallo' oder "Hallo") oder kompliziertere Ausdrücke sein. Für Ausdrücke gibt es die folgenden Operatoren:

&& || ^^ kombiniert boolesche Variablen (Variablen die entweder true oder false sein können (für alle die vom RPG Maker kommen: es handelt sich hierbei um Switches, die ja entweder ON (true) oder OFF (false) sein können)) (&& = und, || =oder , ^^ = xor)

<(kleiner als) <=(kleiner oder gleich) ==(gleich) !=(ungleich) >=(grösser oder gleich) >(grösser als) werden dazu benutzt um zu vergleichen (Ergebnis entweder true(1) oder false(0)).

+ - Addition, Subtraktion

* / div mod sind: Multiplikation, Division, Ganzzahldivision, Modulo(hierbei werden nur die Nachkommastellen der Division ausgegeben)
Als Werte könnt ihr Zahlen, Variablen oder Funktionen, die einen Wert zurückgeben, benutzen. Unter-Ausdrücke können in Klammern gesetzt werden. Alle Operatoren funktionieren mit Zahlen, Vergleiche funktionieren auch für Zeichenketten und + fügt Zeichenketten zusammen.(z.B. wird aus "Hallo" + "...und Tschüss!" wird "Hallo... und Tschüss!")
Beispiele
Hier sind einige sinnlose Beispiele:

```
x = 23; // x wird dem Wert 23 zugeordnet
str = 'hello world'; // str wird der Zeichenkette ‘hello world’ zugeordnet
```

```
y += 5; // der Wert von y wird 5 addiert.
x *= y; // x wird mit y multipliziert
x = 23*((2+4) / sin(y)); // x bekommt den Wert, den 23*((2+4) / sin(y)) zurückgibt.
str = 'hello' + " world"; // str bekommt den Zusammenschnitt von 'hello' + " world"
b = (x < 5) && !(x==2 || x==4); // b ist eine boolesche Variable, d.h. sie kann nur true oder false sein. Wenn dieser Ausdruck (x < 5) && !(x==2 || x==4) stimmt, dann bekommt es den Wert true, sonst false.
```

Funktionen

Eine Funktion hat die Form: Funktionsname, gefolgt von keinem oder mehreren Argumenten in Klammern, getrennt durch Kommata.
<Funktion>(<Argument1>,<Argument2>)
Es gibt zwei Arten von Funktionen. Erstens gibt es sehr viele "built-in" Funktionen und zweitens kann jedes Script, das in eurem Spiel eingebaut ist, auch als Funktion verwendet werden.
Achtung! Auch bei Funktionen ohne Argumente müssen die Klammern gesetzt werden! Einige Funktionen geben Werte zurück und können als Ausdruck benutzt werden, andere führen einfach Befehle aus.
Es ist nicht möglich Funktionen auf die linke Seite eines Assignments zu schreiben. Beispielsweise ist es nicht erlaubt folgendes zu schreiben: "instance_nearest(x,y,obj).speed = 0."Statt dessen muss: "(instance_nearest(x,y,obj)).speed = 0" (ohne die "-Zeichen) geschrieben werden. (Info: "instance_nearest(x,y,obj)" gibt die Id der Instanz zurück, die dem Punkt, festgelegt durch x und y, am nähesten ist.)

Scripts

Wenn ein Script erstellt wird, willst du häufig auf Argumente zugreifen, die ihm weitergegeben wurden (Entweder wenn du das Script ausführst oder wenn du es in einem Programm, oder von einem anderem Script aus oder sogar gleich im selben Script aufrufst).
Diese Argumente sind in den Variablen argument0, argument1, ..., argument15. Man kann also maximal nur 16 Argumente benutzten. (Merke, dass, wenn du dein Script von einer "Action" aus aufrufst, du nur die ersten 5 Argumente deklarieren kannst.) Du kannst auch argument[0] etc. benutzten.

Scripts können auch einen Wert zurückgeben, sodass sie in Ausdrücken verwendet werden können. Für das ausgeben eines Wertes wird das return Statement benutzt:
return <Ausdruck>
Die Ausführung des Scripts wird nach dem return Statement beendet!

Beispiel
Hier ist die Definition für ein Script, dass das Quadrat des ersten Argumentes errechnet:

```
return (argument0*argument0);
```

Um ein Script aus einem Stück Code heraus aufzurufen, muss man genauso verfahren wie bei einer "normalen" Funktion, d.h. den Scriptnamen und die Argumente in Klammern dahinter schreiben.

Ein simples Statement

Ein simples Statement besteht aus mehreren Befehlen (Funktionen oder Scripts). Ein Programm muss in geschweifte Klammern geschrieben werden ("<color=10>\{</color>" und "<color=10>\}</color>") und nach jedem Befehl muss ein Semikolon stehen(";").Daraus ergibt sich die Form eines simplen Statements:

```
{
<Befehl>
;
<Befehl>
;
}
```

Es gibt noch viele andere Arten von Statements:

Das if-Statement

Ein if-Statement hat die Form:

if (<Ausdruck>) <statement>
oder

Das Statement kann auch ein Block sein.
Wenn euer Ausdruck wahr ist (true oder gerundet 1), dann wird das erste Statement ausgeführt, wenn euer Ausdruck falsch (false oder gerundet 0) ist, dann wird euer Statement ausgeführt, was nach dem else-tag kommt.
Um das if-Statement besser zu verstehen, sollte man es einmal ganz ins Deutsche übersetzt ansehen:
Es besagt nämlich nichts anderes als:
Wenn (Ausdruck erfüllt ist) <Tue dieses Statement> sonst <Tue dieses Statement>
Das war nur für die bessere Veranschaulichung.

Es macht den Code leserlicher, immer klare klammern um die einzelnen Statements zu schreiben.
Dadurch hat ein if-Statement die Form:

```
if
(<Ausdruck>)
{
<statement>
}
else
{
<statement>
}
```

Beispiel:
Das folgende Programm bewegt das Objekt zur Mitte des Bildschirms:

```
{
if (x<200) \{x += 4\} else \{x -=
4\};
}
```

Das repeat-Statement

Ein repeat-Statement hat die Form:

repeat (<Ausdruck>) <statement>

Das Statement wird so oft wiederholt, wie der gerundete Wert des Ausdrucks angibt.
Beispiel
Das folgende Programm erschafft 5 Bälle an zufälligen orten.

```
{
repeat (5)
instance_create(random(400),random(400),ball);
}
```

Das while-Statement

Ein while-Statement hat die Form:

while (<Ausdruck>) <statement>

Das Statement wird solange ausgeführt, solange der Wert des Ausdrucks "true" ist.
Wenn wir das while-Statements wieder wörtlich ins Deutsche übersetzten, dann wird man diese Art der Schleife vielleicht besser verstehen:
Solange (Ausdruck wahr ist) <Tue dieses Statement>
Bei While-Schleifen ist Vorsicht geboten, weil man sie leicht in eine Endlosschleife verwandeln kann. In diesem Fall würde das Programm stehen bleiben und nicht mehr auf Benutzereingaben reagieren.
Beispiel:
Das folgende Programm versucht das Objekt an einen freien Ort zu setzen (Das ist genau das selbe wie die
Action "Move an object to a random position").

```
{
while (!place_free(x,y))
x = random(room_width);
y = random(room_height);
}
```

Das do-Statement

Ein do-Statement hat die Form:

do <statement> until (<Ausdruck>)

Das Statement wird solange ausgeführt bis der Ausdruck "true" ist, dabei wird das Statement mindestens einmal ausgeführt.
Zu einer besseren Veranschaulichung, sollte man das do-Statement mal ganz ins Deutsche übersetzten:
Tue <Das Statement> solange, bis (Ausdruck erfüllt ist)

Bei Do-Schleifen ist Vorsicht geboten, weil man sie leicht in eine Endlosschleife verwandeln kann. In diesem Fall würde das Programm stehen bleiben und nicht mehr auf Benutzereingaben reagieren.

Beispiel
Das folgende Programm versucht das Objekt an einen freien Ort zu setzen (Das ist genau das selbe wie die Action "Move an object to a random position"). Der einzige Unterschied zu dem obigem Programm besteht darin, dass hier das Objekt mindestens einmal versetzt wird, was oben nicht der Fall ist.

```
{
do
{
x =
random(room_width);
y =
random(room_height);
}
until (place_free(x,y))
}
```

Das for-Statement

Ein for-Statement hat die Form:

for (<statement1> ; <Ausdruck> ; <statement2>) <statement3>

Das funktioniert wie folgt:
Am Anfang der for-Schleife wird statement1 ausgeführt, dann wird der Ausdruck bewertet. Wenn er "true" (wahr) ist wird statement3 ausgeführt. Statement2 ist das Statement, was am Ende der for-Schleife ausgelöst wird. Schliesslich wird der Ausdruck nochmals bewertet, dann wird wieder statement3 ausgeführt und... das ganze geht solange, bis der Ausdruck "false" (falsch) ist.
Das klingt kompliziert, ist aber eigentlich ganz einfach:
Das erste Statement (statement1) aktiviert die for-Schleife, dabei wird sie aber nur einmal ausgeführt und das ist zum Zeitpunkt, wo die Schleife aktiviert wird.
Der Ausdruck überprüft, ob die Schleife beendet werden muss.
Statement2 ist ein "Schritt-Statement", das zur nächsten Berechnung der Schleife führt. Meistens wird es dazu benutzt um durch einen bestimmten Bereich von Zahlen Befehle auszuführen.
Beispiel
Das folgende Programm initialisiert einen Array (Erklärung zu einem Array kommt später) mit den zehn
Werten 1-10.

```
{
for (i=0; i<9; i+=1) list[i] =
i+1;
}
```

Das switch-Statement

In einigen Situationen muss das Spiel auf bestimmte Werte reagieren. Das ist mit vielen if-

Statements möglich, aber es ist einfacher mit einem switch-Statement. Ein switch-Statement hat die Form:

```
switch (<Ausdruck>)
{
case <Ausdruck1>: <statement1>; break;
case <Ausdruck2>: <statement2>; break;
default: <statement>; break;
}
```

Das geht so: Zuerst wird der Wert des Ausdrucks errechnet, der dann mit den anderen Ausdrücken (die hinter den case-Statements stehen) verglichen wird. Die Ausführung des Scripts wird da fortgesetzt, wo der erste, zu dem Wert passende, Ausdruck steht und dort beendet, wo ein break-Statement steht. Wenn kein Ausdruck den richtigen Wert hat, wird die Ausführung beim default-Statement fortgesetzt (es ist kein default-Statement notwendig, es kann auch unter Umständen entfallen) Es sei angemerkt, dass mehrere case-Statements für ein und dasselbe Statement gesetzt werden können. Das break-Statement ist ebenfalls nicht notwendig. Wenn kein break-Statement vorhanden ist, wird die Ausführung bei dem nächsten case-Statement fortgesetzt.

Beispiel
Das folgende Programm reagiert auf bestimmte Tasten, die gedrückt sind/werden.

```
switch
(keyboard_key)
{
case vk_left:
case vk_numpad4:
x -= 4; break;

case vk_right:
case vk_numpad6:
x += 4; break;
}
```

Das break-Statement

Das break-Statement hat die Form:

```
break
```

Wenn es innerhalb einer for-Schleife, while-Schleife, repeat-Schleife, einem switch-Statement oder einem with-Statement (Erklärung zu diesem folgt später) vorkommt, beendet es die Ausführung

dieser Schleife oder des Statements. Ausserhalb dieser Statements beendet es die Ausführung des Scripts (nicht die des Spiels).

Das continue-Statement

Das continue-Statement hat die Form:

```
continue
```

Wenn es innerhalb einer for-Schleife, while-Schleife, repeat-Schleife, einem switch-Statement oder einem with-Statement vorkommt, fährt es mit dem nächsten Wert der Schleife oder des Statements fort.

Das exit-Statement

Das exit-Statement hat die Form:

```
exit
```

Es beendet einfach die Ausführung des Scripts oder des "piece of code". (Es beendet nicht die Ausführung des Spiels! Dafür wird die Funktion game_end() benötigt.)

30. Berechnungen

Der Game Maker enthält eine Menge an Funktionen, um gewisse Dinge zu berechnen.
Hier eine komplette Auflistung:

30.1 Konstanten

Folgende Konstanten gibt es:

true (wahr) ist gleich 1.
false (falsch) ist gleich 0.
pi (Pi) ist gleich 3.1415...

30.2 Funktionen mit reellen Zahlen

Folgende Funktionen gibt es, die sich mit reellen Zahlen befassen:

random(x) Liefert eine zufällige reelle Zahl zwischen 0 und x ,wobei sie immer kleiner als x ist.

abs(x) Gibt den absoluten Wert von x wieder.

sign(x) Liefert das Vorzeichen von x(-1, 0 oder 1).

round(x) Gibt den zur nächsten ganzen Zahl gerundeten Wert von x.

floor(x) Gibt den abgerundeten ganzzahligen Wert von x.

ceil(x) Gibt den aufgerundeten ganzzahligen Wert von x.

frac(x) Gibt den gebrochenen Anteil von x (Teil nach dem Komma).

sqrt(x) Zieht die Quadratwurzel aus einem nicht-negativem x.

sqr(x) Gibt x zum Quadrat wieder.

power(x,n) Liefert den Wert von x hoch n.

exp(x) Liefert den Exponenten e zur Potenz x.

ln(x) Natürlicher Logarithmus von x.

log2(x) Logarithmus zur Basis 2 von x.

log10(x) Logarithmus zur Basis 10 von x.

logn(n,x) Logarithmus zur Basis n von x.

sin(x) sinus von x (x in rad).

cos(x) cosinus von x (x in rad).

tan(x) tangens von x (x in rad).

arcsin(x) arcus sinus von x.

arccos(x) arcus cosinus von x.

arctan(x) arcus tangens von x.

arctan2(y,x) Berechnet den arcus tangens(y/x), und gibt einen Winkel im richtigen Quadranten

wieder.

degtorad(x) Umwandlung grad->rad.

radtodeg(x) Umwandlung rad->grad.

min(**val1,val2,val3,…)** Gibt das Minimum der Werte an. Die Funktion kann bis zu 16 Argumente verarbeiten. Diese müssen alle Zahlen oder Strings sein.

max(**val1,val2,val3,…)** Gibt das Maximum der Werte an. Die Funktion kann bis zu 16 Argumente verarbeiten. Diese müssen alle Zahlen oder Strings sein.

mean(**val1,val2,val3,…)** Gibt den Mittelwert der Werte an. Die Funktion kann bis zu 16 Argumente verarbeiten. Diese müssen alle Zahlen oder Strings sein..

point_distance(x1,y1,x2,y2) Punktabstand von (x1, y1;x2, y2).

point_direction(x1,y1,x2,y2) Richtung von Punkt 1 (x1, y1) zu Punkt 2 (x2, y2) in Grad.

is_real(x) Ermittelt, ob x eine Zahl ist(im Gegensatz zu einem string).

is_string(x) Ermittelt, ob x ein string ist(im Gegensatz zu einer Zahl).

30.3 Funktionen zum Bearbeiten von strings (Zeichenketten)

Die folgenden Funktionen befassen sich mit Zeichen und Zeichenketten:

chr(val) Gibt das Zeichen, welches dem ASCI-Code "val" entspricht wieder.

ord(str) Liefert den ASCI-Code des ersten Zeichens des strings.

real(str)Wandelt eine Zeichenkette in eine Zahl. str kann ein Minus-Zeichen, einen Dezimalpunkt und sogar einen exponentiellen Teil haben.

string(val)Wandelt den Wert "val" in eine Zeichenkette im Standardformat (keine Kommastellen bei Ganzzahlen, ansonsten 2 Dezimalstellen).

string_format(val,tot,dec) Wandelt val in einen string bei selbstgewähltem Format: tot bezeichnet die Anzahl aller Stellen und dec bezeichnet die Dezimalstellen.

string_length(str)Anzahl der Zeichen in Zeichenkette str.

string_pos(substr,str) Liefert die Position von substr in str (0=kommt nicht vor).

string_copy(str,index,count) Liefert den Teilstring von str beginnend bei index mit der Länge count.

string_char_at(str,index) Gibt den Zeichen des strings (str) an Position index zurück.

string_delete(str,index,count) Kopie von string ohne den Teil ab index mit der Länge count(Ausschneiden).

string_insert(substr,str,index) Fügt den Teilstring substr in str ein an der Position index.

string_replace(str,substr,newstr) Liefert eine Kopie von str, wobei substr gegen newstr an der ersten Stelle ausgetauscht wurde.

string_replace_all(str,substr,newstr) Wie oben, nur werden alle substr ausgetauscht.

string_count(substr,str) Gibt an, wie oft substr in str enthalten ist.

string_lower(str) Kopie von str kleingeschrieben.

string_upper(str) Kopie von str grossgeschrieben.

string_repeat(str,count) Gibt einen string bestehend aus count Kopien von str wieder.

string_letters(str) Gibt nur die Buchstaben in str wieder.

string_digits(str) Gibt nur die Zahlen des strings (str) wieder.

string_lettersdigits(str) Gibt Zahlen und Buchstaben des strings (str) an.

Folgende Funktionen befassen sich mit der Textspeicherung in der Zwischenablage:

clipboard_has_text() Gibt an, ob Zwischenablage Text enthält.

clipboard_get_text() Gibt den aktuellen Text der Zwischenablage wieder.

clipboard_set_text(str) Übergibt str an die Zwischenablage.

30.4 Umgang mit Datumsangaben und Uhrzeiten

Im Game Maker gibt es eine Anzahl von Funktionen, um sich mit Daten und Zeitangaben befassen. Eine Datums-Zeit Kombination wird in einer reellen Zahl abgespeichert. Der Bestandeil eines Datums-Zeit-Wertes ist die Anzahl der Tage, die seit 12/30/1899 verstrichen sind. Die Nachkommastellen des Datum-Zeit-Wertes stellen den Bruchteil eines 24-Stunden Tages dar, der verstrichen ist. Nachstehende Funktionen gibt es:

date_current_datetime() Gibt den Datum-Zeit-Wert wieder, der dem aktuellen Moment entspricht.

date_current_date() Gibt den Datum-Zeit-Wert wieder, der nur dem aktuellen Datum entspricht (die Zeit wird ignoriert).

date_current_time() Gibt den Datum-Zeit-Wert wieder, der nur der aktuellen Zeit entspricht (das Datum wird ignoriert).

date_create_datetime(year,month,day,hour,minute,second) Erzeugt einen Datum-Zeit-Wert, entsprechend dem angegebenen Datum und der angegebenen Zeit.

date_create_date(year,month,day) Erzeugt einen Datum-Zeit-Wert, entsprechend dem angegebenen Datum.

date_create_time(hour,minute,second) Erzeugt einen Datum-Zeit-Wert, entsprechend der angegebenen Zeit.

date_valid_datetime(year,month,day,hour,minute,second) Gibt an, ob die angegebenen Datum- und Zeitwerte gültig sind.

date_create_date(year,month,day) Gibt wieder, ob das angegebene Datum gültig ist.

date_create_time(hour,minute,second) Gibt wieder, ob die angegebene Zeit gültig ist.

date_inc_year(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Jahre hinter dem

angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_month(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Monate hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_week(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Wochen hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_day(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Tage hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_hour(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Stunden hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_minute(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Minuten hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_inc_second(date,amount) Liefert ein neues Datum, das "amount" (Betrag) Sekunden hinter dem angegebenen Datum liegt. "amount" muß eine ganzzahlige Zahl sein.

date_get_year(date) Liefert das Jahr, entsprechend des angegebenen Datums.

date_get_month(date) Liefert den Monat, entsprechend des angegebenen Datums.

date_get_week(date) Liefert die Woche, entsprechend des angegebenen Datums.

date_get_day(date) Liefert den Tag des Monats, entsprechend des angegebenen Datums.

date_get_hour(date) Liefert die Stunde, entsprechend des angegebenen Datums.

date_get_minute(date) Liefert die Minute, entsprechend des angegebenen Datums.

date_get_second(date) Liefert die Sekunde, entsprechend des angegebenen Datums.

date_get_weekday(date) Liefert den Wochentag, entsprechend des angegebenen Datums.

date_get_day_of_year(date) Liefert den Tag des Jahres, entsprechend des angegebenen Datums.

date_get_hour_of_year(date) Liefert die Stunde des Jahres, entsprechend des angegebenen Datums.

date_get_minute_of_year(date) Liefert die Minute des Jahres, entsprechend des angegebenen Datums.

date_get_second_of_year(date) Liefert die Sekunde des Jahres, entsprechend des angegebenen Datums.

date_year_span(date1,date2) Liefert die Anzahl der Jahre zwischen zwei Daten. Unvollständige Jahre werden als Bruchteil angegeben.

date_month_span(date1,date2) Liefert die Anzahl der Monate zwischen zwei Daten. Unvollständige Monate werden als Bruchteil angegeben.

date_week_span(date1,date2) Liefert die Anzahl der Wochen zwischen zwei Daten. Unvollständige Wochen werden als Bruchteil angegeben.

date_day_span(date1,date2) Liefert die Anzahl der Tage zwischen zwei Daten. Unvollständige Tage werden als Bruchteil angegeben.

date_hour_span(date1,date2) Liefert die Anzahl der Stunden zwischen zwei Daten. Unvollständige Stunden werden als Bruchteil angegeben.

date_minute_span(date1,date2) Liefert die Anzahl der Minuten zwischen zwei Daten. Unvollständige Minuten werden als Bruchteil angegeben.

date_second_span(date1,date2) Liefert die Anzahl der Sekunden zwischen zwei Daten.

Unvollständige Sekunden werden als Bruchteil angegeben.

date_compare_datetime(date1,date2) Vergleicht die beiden Datum-Zeit-Werte. Rückgabewert -1, 0, oder 1 je nachdem, ob der erste Wert kleiner, gleich oder größer als der zweite Wert ist.

date_compare_date(date1,date2) Vergleicht die beiden Datum-Zeit-Werte. Nur der Datumsteil wird berücksichtigt. Rückgabewert -1, 0, oder 1 je nachdem, ob der erste Wert kleiner, gleich oder größer als der zweite Wert ist.

date_compare_time(date1,date2) Vergleicht die beiden Datum-Zeit-Werte. Nur der Zeitteil wird berücksichtigt. Rückgabewert -1, 0, oder 1 je nachdem, ob der erste Wert kleiner, gleich oder größer als der zweite Wert ist.

date_date_of(date) Liefert den Datumsteil des angegebenen Datum-Zeit-Wertes. Setzt den Zeitteil auf 0.

date_time_of(date) Liefert den Zeitteil des angegebenen Datum-Zeit-Wertes. Setzt den Datumsteil auf 0.

date_datetime_string(date) Liefert einen "string" (Zeichenkette) des angegebenen Datums und der angegebenen Zeit im voreingestellten Format des Systems.

date_date_string(date) Liefert einen "string" (Zeichenkette) des angegebenen Datums im voreingestellten Format des Systems.

date_time_string(date) Liefert einen "string" (Zeichenkette) der angegebenen Zeit im voreingestellten Format des Systems.

date_days_in_month(date) Liefert die Anzahl der Tage im Monat des angegebenen Datum-Zeit-Wertes.

date_days_in_year(date) Liefert die Anzahl der Tage im Jahr des angegebenen Datum-Zeit-Wertes.

date_leap_year(date) Gibt an, ob das Jahr des angegebenen Datum-Zeit-Wertes ein Schaltjahr ist.

date_is_today(date) Gibt an, ob der angegebene Datum-Zeit-Wert auf den heutigen Tag fällt.

31. Spielgeschehen

Es gibt eine Vielzahl von Variablen und Funktionen, die du verwenden kannst, um das Spielgeschehen zu definieren. Diese beeinflussen im speziellen die Bewegung und die Erschaffung von Instanzen, das "Timing", und den Umgang mit "events" (Ereignissen).

31.1 Moving around (Umherbewegen)

Offensichtlich ist es ein wichtiger Aspekt bei Spielen, die Objektinstanzen auch zu bewegen. Jede Instanz besitzt zwei eingebaute Variablen x und y, welche die Position der Instanz markieren. (Um präzise zu sein: Sie geben den Bezugs-/Ursprungspunkt des "sprites" an.) Position (0,0) ist die linke obere Ecke des Raumes.
Du kannst die Position der Instanz ändern, indem du seine x und y Variablen veränderst. Wenn dein Objekt komplizierte Bewegungen vollführen soll, ist das hier der richtige Weg. Typischerweise wird dieser Programmcode ins "step event" (Schrittereignis) des Objektes gesetzt.

Wenn das Objekt mit konstanter Geschwindigkeit und gleichbleibender Richtung bewegt werden soll, gibt es einen einfacheren Weg: Jedes Objekt besitzt eine horizontale Geschwindigkeitskomponente(hspeed) und eine vertikale Geschwindigkeitskomponente (vspeed). Beide werden in "pixels per step" (Bildpunkte pro Programmschritt) angegeben. Eine positive horizontale Geschwindigkeitskomponente bedeutet eine Bewegung nach rechts, eine negative nach links. Positive vertikale Geschwindigkeit bewegt nach unten, negative nach oben. Somit musst du nur einmal diese Variablen setzen (beispielsweise im "creating event"), um der Objektinstanz eine konstante Bewegung zu geben. Es gibt noch eine andere Art, Bewegung zu spezifizieren - Richtungsangabe (0° bis 359°) und Geschwindigkeit (sollte nicht negativ sein). Du kannst diese Variablen setzen und auslesen, um eine beliebige Bewegung zu spezifizieren. (innerhalb des Game Makers werden diese Werte in "hspeed" und "vspeed" umgewandelt.) Ferner gibt es noch die Reibung (friction) und die Anziehungskraft (gravity) samt Richtung (gravity direction).
Abschliessend sei noch die Funktion: motion_add(dir,speed) erwähnt, welche eine Bewegung zur vorhandenen addiert.
Um vollständig zu sein, hat jede Instanz folgende Variablen und Funktionen, welche sich mit Position und Bewegung befassen:

x x-position der Instanz.

y y-position der Instanz.

xprevious vorherige x-position.

yprevious vorherige y-position.

xstart Anfangs-x-position im Raum.

ystart Anfangs-y-position iIm Raum.

hspeed horizontale Komponente der Geschwindigkeit (speed).

vspeed Vertikale Komponente der Geschwindigkeit (speed).

direction Momentane Richtung (direction) {0°-360°, gegen den Uhrzeigersinn, 0 = nach rechts}.

speed Momentane Geschwindigkeit (in "pixels per step" [pps]).

friction Momentane Reibung (friction) (in "pixels per step").

gravity Momentaner Wert der Anziehungskraft (gravity) (in "pixels per step").

gravity_direction Richtung der Anziehungskraft (270 ist nach unten).

motion_set(dir,speed) Setzt die Bewegung mit angegebener Richtung(dir) und Geschwindigkeit(speed).

motion_add(dir,speed) Addiert die Bewegung zur vorhandenen (Vektoraddition).

Es gibt ein Menge an Funktionen, welche dir bei Bewegungsdefinitionen helfen:

place_free(x,y) Gibt an, ob die Instanz an angegebener Position(x,y) eine Kollision auslöst. Sie wird verwendet als Prüfroutine bevor man sich auf die neue Stelle bewegt.

place_empty(x,y) Gibt an, ob die Instanz an angegebener Position(x,y) auf nichts trifft. (Diese Funktion beachtet auch "non-solid" Instanzen.)

place_meeting(x,y,obj) Gibt an, ob die Instanz an angegebener Position(x,y) das Objekt obj trifft. obj kann ein Objekt sein - in so einem Fall gibt die Funktion den Wert "true" zurück, wenn es auf eine Instanz des Objekts trifft. Es kann aber auch eine "instance id" sein, das besondere Wort "all" (bedeutet irgendeine Instanz irgendeines Objektes), oder das besondere Wort "other".

place_snapped(hsnap,vsnap) Gibt an, ob die Instanz am Raster ausgerichtet ist(hsnap,vsnap).

move_random(hsnap,vsnap) Bewegt die Instanz zu einer freien, zufälligen und am Raster ausgerichteten Position.

move_snap(hsnap,vsnap) Richtet die Instanz am Raster aus.

move_towards_point(x,y,sp) Bewegt die Instanz mit der Geschwindigkeit (sp) zum Punkt(x,y).

move_bounce_solid(adv) Abprallen von "solid" Instanzen. adv gibt an, ob genauer berechnet wird (schräge Wände).

move_bounce_all(adv) Abprallen von allen Instanzen - nicht nur "solide".

move_contact_solid(dir,maxdist) Bewegt die Instanz solange in die angegebene Richtung, bis ein Kontakt mit einer anderen "solid" Instanz erreicht wird. Wenn an der aktuellen Position keine Kollision vorliegt, wird die Instanz kurz vor der Kollision platziert. Wenn schon eine Kollision vorliegt, wird die Instanz nicht bewegt. Du kannst die maximale Bewegungsweite(maxdist) angeben (negative Werte bedeuten beliebig lange).

move_contact_all(dir,maxdist) Wie oben, nur diesmal stoppt die Instanz bei jedem Kontakt mit einem anderen Objekt - nicht nur "solid" Objekte.

move_outside_solid(dir,maxdist) Bewegt die Instanz solange in die angegebene Richtung, bis sie sich nicht mehr innerhalb eines "solid" Objekts befindet. Wenn an der angegebenen Stelle keine Kollision auftritt, wird die Instanz auch nicht bewegt. Auch hier kannst Du die maximale Bewegungsweite angeben (maxdist negativ = beliebig weit).

move_outside_all(dir,maxdist) Wie oben, nur diesmal alle Objekte nicht nur "solid objects".

distance_to_point(x,y) Gibt die Distanz der Umrissbox (bounding box) der Instanz zum Punkt (x,y) wieder.

distance_to_object(obj) Gibt die Distanz der Instanz zur nächstgelegenen Instanz des angegebenen Objektes (obj) an.

position_empty(x,y) Gibt an, ob sich nix an Position (x,y) befindet.

position_meeting(x,y,obj) Gibt an, ob an Position (x,y) eine Instanz von Objekt (obj) vorhanden ist. obj kann ein Objekt sein, ein "instance id" oder eins der Schlüsselwörter: "self"(selbst), "other"(andere), oder "all"(alle).

31.2 Pfade

Im Game Maker kannst du Pfade definieren und Instanzen beauftragen, solchen Pfaden zu folgen. Obwohl du dafür die Aktionen (actions) verwenden kannst, gibt es Funktionen und Variablen dafür, welche mehr Flexibilität bieten:

path_start(path,speed,endaction,absolute) Startet einen Pfad für die aktuelle Instanzs. Der "path" ist der Name des Pfades, der beschritten werden soll. Die "speed" ist die Gewschwindigkei, mit der der Pfad abgeschritten werden soll. Eine negative Geschwindigkeitsangabe bedeutet, daß der Pfad rückwärts von der Instanz beschritten werden soll. Die "endaction" gibt an, was geschehen soll, wenn das Ende des Pfades erreicht worden ist. Folgende Werte können eingesetzt werden:

0 : stoppt den Pfad

1 : setzt den Pfad an der Startposition fort (wenn der Pfad nicht geschlossen ist, wird zur Anfangsposition gesprungen)

2 : setzt den Pfad an der aktuellen Position fort

3 : verfolgt den Pfad zurück, das heißt Vorzeichenänderung der Geschwindigkeit (speed)

Das Argument "absolute" sollte wahr (true) oder falsch (false) sein. Wenn es wahr ist, werden absolute Koordinaten des Pfades verwendet. Wenn es falsch ist, werden die Koordinaten relativ zur aktuellen Position der Instanz verwendet. Um es genauer zu sagen: Wenn die Geschwindigkeit positiv ist, wird der Startpunkt des Pfades auf die aktuelle Position der Instanz gesetzt und der Pfad wird von dort aus abgeschritten. Wenn die Geschwindigkeit negativ ist, wird der Endpunkt des Pfades an die aktuelle Position der Instanz gesetzt und der Pfad wird rückwärts abgeschritten.

path_end() Beendet das Abschreiten eines Pfades für die aktuelle Instanz.

path_index* Index des aktuellen Pfades, dem die Instanz folgt. Du kannst das nicht direkt ändern, sondern mußt obenstehende Funktion verwenden.

path_position Position im aktuellen Pfad. 0 entspricht dem Anfang des Pfades. 1 entspicht dem Ende des Pfades. Der Wert muß zwischen 0 und 1 liegen.

path_positionprevious Vorherige Position im aktuellen Pfad. Dies kann beispielsweise in Kollisionsereignissen verwendet werden, um die Position im Pfad auf die vorherige Stelle zu setzen.

path_speed Geschwindigkeit (in pixel pro step) mit der der Pfad abgeschritten wird. Verwende einen negativen Wert, für die Rückwärtsbewegung.

path_orientation Orientierung (gegen den Uhrzeigersinn "counter-clockwise") in welcher der Pfad ausgeführt wird. 0 ist die normale Orientierung des Pfades.

path_scale Skalierung des Pfades. Vergrößern, um den Pfad aufzublähen. 1 ist der voreingestellte Wert.

path_endaction Die Aktion, die am Ende des Pfades ausgeführt werden soll. Du kannst die obengenannten Werte verwenden.

31.3 Motion planning (Bewegungsplanung)

Bewegungsplanung hilft dir bestimmte Instanzen von einer gegebenen Stelle zu einer anderen Stelle zu bewegen, wobei Kollisionen mit bestimmten anderen vermieden werden (Wände zum Beispiel). Bewegungsplanung ist ein schwieriges Problem. Es ist unmöglich allgemeine Funktionen bereitzustellen, die in allen Situationen zufriedenstellend arbeiten. Auch sind die Berechnungen für kollisionsfreie Bewegungen recht zeitaufwändig. Deshalb sei sorgsam wie und wann Du es einsetzt. Bitte behalte diese Anmerkungen immer im Hinterkopf, wenn du irgendeine folgender Funktionen verwendest.

Verschiedene Arten der Bewegungsplanung werden vom Game Maker bereitgestellt. Die einfachste Art läßt eine Instanz einen Schritt in Richtung Zielposition machen, direkt und gradlienig, wenn möglich - aber auch in anderer Richtung, falls erforderlich. Diese Funktionen sollten im "step event" einer Instanz verwendet weren. Sie korrespondieren mit den Bewegungsplanungsaktionen ("motion planning actions"), die auch verfügbar sind:

mp_linear_step(x,y,stepsize,checkall) Diese Funktion läßt die Instanz einen Schritt in Richtung der angegebene Stelle (x,y) ausführen. Die Schrittweite des Schrittes wird durch "stepsize" angezeigt. Wenn die Instanz sich schon an der Stelle befindet, wird sie sich nicht weiter bewegen. Wenn "checkall" wahr ("true") ist stoppt die Instanz, sobald sie auf eine Instanz irgendeinenes Objektes trifft. Wenn es nicht wahr ("false") ist, stoppt sie nur, wenn eine "solid" Instanz getroffen wird. Beachte, daß diese Funktion nicht versucht Umwege zu machen, wenn sie auf Hindernisse stößt. Sie schlägt einfach fehl in diesem Fall. Die Funktion gibt wieder, ob die Zielposition erreicht wurde.

mp_potential_step(x,y,stepsize,checkall) Wie voranstehende Funktion, läßt auch diese Funktion die Instanz einen Schritt auf eine bestimmte Position hin machen. Aber diesmal versucht sie Hindernisse zu umgehen. Wenn die Instanz mit "solid" Instanz (oder irgendeiner, wenn "checkall" "true (wahr) ist) zusammen zu stoßen droht, wird sie die Richtung ändern, um die Instanz zu meiden und es umgehen. Diese Näherung unterliegt keiner garantiert funktionierenden Arbeitsweise aber in den meisten einfachen Fällen bewegt es die Instanz effektiv auf die Zielposition. Die Funktion gibt an, ob das Ziel errreicht wurde.

mp_potential_settings(maxrot,rotstep,ahead,onspot) Die vorherige Funktion verrichtet ihre Arbeit, indem sie eine Anzahl von Parametern verwendet, welche mit dieser Funktion verändert werden können. Prinzipiell funktioniert diese Methode wie folgt. Zuerst wird versucht, das Ziel auf direktem Weg zu erreichen. Es wird eine Anzahl von Schritten vorausgeschaut, was mit dem Parameter "ahead" (voreingestellt auf 3) festgelegt werden kann. Vermindern dieses Wertes führt dazu, daß die Instanz die Richtungsänderung später beginnt. Erhöhen bedeutet eine frühere Richtungsänderung. Falls diese Prüfung zu einer Kollision führt, schaut sie nach anderen Richtungen, die mehr links oder rechts, von der bestmöglichen Richtung liegen. Dies macht sie in Schritten der Größe "rotstep" (voreingestellt 10). Vermindern dieses Wertes ermöglicht der Instanz mehr Bewegungsmöglichkeit, ist aber langsamer. Der Parameter "maxrot" ist ein wenig schwierig zu erklären. Die Instanz besitzt eine aktuelle Bewegungsrichtung ("direction"). maxrot (voreingestellt 30) gibt die Abweichung zur aktuellen Bewegungsrichtung an, die maximal in einem Schritt geändert werden darf. Auch wenn sie sich beispielsweise direkt zum Ziel bewegen könnte, macht sie es nur so, daß die maximale Abweichung der Bewegungsrichtung nicht überschritten wird. Falls Du "maxrot" hoch ansetzt, kann die Instanz die Bewegungsrichtung mehr ändern in einem Schritt. Dies macht es einfacher einen kurzen Weg zu finden aber der Weg wird häßlicher sein. Falls Du aber den Wert kleiner machst, wird der Weg geschmeidiger aber möglicherweise mit längeren Umwegen (manchmal wird das Ziel auch garnicht erreicht). Falls kein Schritt gemacht werden kann, hängt das Verhalten vom Wert des Parameters "onspot" ab. Wenn "onspot" wahr

(true) ist (der voreingestellte Wert), wird die Instanz auf dem Fleck um den Betrag, der durch "maxrot" vorgegeben ist, rotieren. Falls der Wert unwahr (false) ist, findet keine Bewegung statt. Den Wert auf "false zu setzen ist beispielsweise nützlich für Autos, setzt aber die Wegfindungschancen herab.

Beachte bitte, daß die Potenzial_Näherung nur lokale Informationen verwendet. Somit wird nur eine Lösung gefunden, wenn diese lokalen Informationen hinreichend sind, die richtige Richtung der Bewegung zu bestimmen. Es wird beispielsweise scheitern, einen Weg aus einem Labyrinth zu finden (fast immer).

Die zweite Art von Funktionen berechnet einen kollisionsfreien Weg für die Instanz. Sobald der Weg kalkuliert worden ist, kannst Du ihn der Instanz zuweisen, damit sie aufs Ziel zubewegt wird. Die Berechnung des Pfads nimmt einige Zeit in Anspruch aber dafür wird danach die Abschreitung des Pfads zügig sein. Dies gilt selbstverständlich nur, wenn sich die Situation nicht in der Zwischenzeit verändert. Zum Beispiel, wenn Hindernisse sich verändern, mußt du den Weg wahrscheinlich neu berechnen. Beachte auch hier, daß diese Funktionen fehlschlagen können. Diese Funktionen sind nur in der registrierten Funktion vom Game Maker verfügbar.

Die ersten beiden Funktionen verwenden lineare-Bewegung- und Potential-Feld-Näherung, die auch für die "step"-Funktionen verwendet werden (Anmerkung des Übersetzers: Bin mir nicht sicher, ob man es so übersetzt).

mp_linear_path(path,xg,yg,stepsize,checkall)Diese Funktion berechnet einen geradlinigen Weg für die Instanz von der aktuellen Position zur Position (xg,yg), wobei die angegebene "stepsize" verwendet wird. Sie gebraucht Schritte (steps) wie in der Funktion mp_linear_step(). Der angegebene Pfad (path) muß schon existieren und wird durch einen neuen Pfad überschrieben. (In einem späteren Kapitel wird das Erstellen und Vernichten von Pfaden erläutert.) Die Funktion gibt an, ob ein Pfad gefunden wurde. Die Funktion stoppt und berichtet das Fehlschlagen, wenn kein direkter Pfad zwischen Start und Ziel existiert. Wenn sie fehlschlägt wird trotzdem ein Pfad definiert, der bis zu der Stelle verläuft, an der die Instanz blockiert wurde.

mp_potential_path(path,xg,yg,stepsize,checkall,factor) Diese Funktion berechnet einen Pfad für die Instanz von der aktuellen Position und mit der momentanten Orientierung zur Position (xg,yg), wobei die angegebene Schrittweite (stepsize) verwendet wird und versucht wird Kollisionen mit Hindernissen zu vermeiden. Sie verwendet Potential-Feld-Schritte, wie in der Funktion mp_potential_step() und zusätzlich die Parameter die mit der Funktion mp_potential_settings() eingestellt werden können. Der angegebenen Pfad muß schon vorhanden sein und wird durch einen neuen Pfad überschrieben. (In einem späteren Kapitel wird das Erstellen und Vernichten von Pfaden erläutert.) Die Funktion gibt an, ob ein Pfad gefunden wurde. Um zu vermeiden, daß die Funktion endlos weiterrechnet, mußt Du einen Längenfaktor (length factor) größer 1 bereitstellen. Die Funktion stoppt und berichtet das Fehlschlagen, wenn sie keinen kürzeren Pfad, zwischen Start und Ziel, kleiner als dieser faktorabhängigen Distanz findet. Ein Faktor von 4 ist normalerweise ausreichend aber wenn Du lange Umwege erwartest, kannst Du ihn möglicherweise verlängern. Wenn sie fehlschlägt, wird trotzdem ein Pfad erstellt, der in Richtung des Zielpunktes weist, ihn aber nicht erreicht.

Die anderen Funktionen verwenden einen wesentlich komplexeren Mechanismus, wobei eine Gitter-basierte Näherung genutzt wird (manchmal A* Algorithmus genannt). Es ist erfolgreicher in der Wegfindung (obwohl es auch manchmal fehlschlagen kann) und findet kürzere Pfade aber es erfordert mehr Aufwand von deiner Seite. Die Grundlegende Idee ist folgende. Zuerst legen wir ein Gitter über den (relevanten Teil des) Raum. Du kannst wählen zwischen einem feinmaschigen

Gitter (was langsamer sein wird) oder einem groben Gitter. Als nächstes bestimmen wir für alle relevanten Objekte die Gitterzellen, die sie überlappen (sowohl "bounding boxes" alsauch präzise Prüfung können verwendet werden) und markieren diese Zellen als verboten. Somit wird eine Zelle als verboten markiert, sogar wenn sie nur teilweise von einem Hindernis überdeckt wird. Schließlich spezifizieren wir eine Start- und eine Zielposition (welche innerhalb der freien Zellen liegen müssen) und die Funktion berechnet den kürzesten Pfad (eigentlich dicht am kürzesten) zwischen diesen. Der Pfad wird durch die Zentren der freien Zellen verlaufen. Wenn also die Zellen groß genug sind, so daß die im Zentrum platzierte Instanz vollständig innerhalb liegt, wird es erfolgreich sein. Diesen Pfad kannst du nun einer Instanz übergeben, damit sie ihm folgt. Die Gitter-basierte Näherung ist sehr mächtig (und wird in vielen professionellen Spielen angewandt) aber es erfordert sorgsames Bedenken von dir. Du mußt bestimmen, welche Gebiete und Zellgrößen ausreichend sind, um das Spiel zu lösen. Auch mußt du festlegen, welche Objekte gemieden werden müssen und ob präzise Kollisionsprüfung wichtig ist. All diese Parameter beeinflußen stark die effizienz dieser Näherung.

Insbesondere die Größe der Zellen ist entscheidend. Erinnere dich daran, daß die Zellen groß genug sein müssen, um das sich bewegende Objekt, was mit seinem Bezugspunkt (origin) im Zentrum der Zelle liegt, ganz zu umfassen. (Sei sorgsam mit der Position des Bezugspunkts des Objektes. Realisiere auch, daß du den Pfad verschieben kannst, wenn der Bezugspunkt des Objektes nicht in dessen Zentrum liegt!) Andererseits bestehen mehr mögliche Pfade, je kleiner die Zellen sind. Wenn du die Zellen zu groß machst, werden Öffnungen/Durchgänge zwischen den Hindernissen möglicherweise verschlossen, weil alle Zellen ein Objekt schneiden. Die Funktionen für die Gitter-basierte Näherung sind folgende:

mp_grid_create(left,top,hcells,vcells,cellwidth,cellheight) Diese Funktion erstellt das Gitter. Sie gibt einen Index zurück, der bei allen anderen Aufrufen verwendet werden muß. Du kannst mehrfache Gitter-Strukturen zur selben Zeit erstellen und verwalten. "left" und "top" geben die Position der linken oberen Ecke des Gitters an. "hcells" und "vcells" geben die Anzahl der horizontalen und vertikalen Zellen an. Schließlich geben "cellwidth" und "cellheight" die Größe der Zellen an.

mp_grid_destroy(id) Zerstört das angegebene Gitter und gibt belegten Speicher frei. Vergiß nicht dies aufzurufen, sobald Du das Gitter nicht mehr benötigst.

mp_grid_clear_all(id) Markiert alle Zellen des Gitters als frei.

mp_grid_clear_cell(id,h,v) Löscht die angegebene Zelle . Zelle 0,0 ist die linke obere Zelle.

mp_grid_clear_rectangle(id,left,top,right,bottom) Löscht alle Zellen die das angegebene Rechteck schneiden (in Raumkoordinaten).

mp_grid_add_cell(id,h,v) Markiert die angegebene Zelle als verboten. Zelle 0,0 ist die linke obere Zelle.

mp_grid_add_rectangle(id,left,top,right,bottom) Markiert alle Zellen die das angegebene Rechteck schneiden als verboten.

mp_grid_add_instances(id,obj,prec) Markiert alle Zellen die eine Instanz des angegebenen Objektes schneiden als verboten. Du kannst auch eine individuelle Instanz verwenden, indem du "obj" die id der Instanz zuweist. Ferner kannst du das Schlüsselwort "all" angeben, um alle Instanzen aller Objekte anzugeben. "prec" gibt an, ob präzise Kollisionsprüfung verwendet werden soll (funktioniert nur, wenn präzise Prüfung für das von der Instanz verwendete sprite aktiviert ist).

mp_grid_path(id,path,xstart,ystart,xgoal,ygoal,allowdiag) Berechnet einen Pfad durch das Gitter. "path" muß einen existierenden Pfad angeben, der durch den Computerpfad ersetzt wird. "xstart"

und "ystart" geben den Start und "xgoal" und "ygoal" das Ziel an. "allowdiag" gibt an, ob diagonale Bewegung zulässig ist statt nur horizontaler oder vertikaler. Die Funktion gibt wieder, ob sie erfolgreich einen Pfad gefunden hat. (Beachte, daß der Pfad unabhängig von der aktuellen Instanz ist; es ist ein Pfad durch das Gitter, nicht ein Pfad für eine besondere Instanz.)

mp_grid_draw(id) Diese Funktion zeichnet das gitter, wobei freie Zellen grün eingefärbt sind und verbotene werden rot gezeichnet. Diese Funktion ist sehr langsam und steht nur als Fehlersuchwerkzeug zur Verfügung.

31.4 Kollisionsprüfung

Wenn Bewegungen geplant werden oder aufgrund bestimmter Aktionen entschieden wird, ist es oftmals wichtig zu wissen, ob Kollisionen mit anderen Objekten an bestimmten Stellen/Positionen bestehen. Folgende Funktionen können dafür verwendet werden. All diese haben drei Argumente gemeinsam:

Das Argument obj kann ein Objekt sein, das Schlüsselwort all, oder die id einer Instanz.

Das Argument prec gibt an, die Prüfung präzise sein soll oder nur auf der "bounding-box" der Instanz basiert. Präzise Prüfung wird nur ausgeführt, wenn das sprite der Instanz es aktiviert hat.

Das Argument notme kann auf wahr (true) gesetzt werden, um anzugeben, daß die aufrufende Instanz nicht geprüft wird. Alle diese Funktionen geben entweder die id von einer der beiden kollidierenden Instanzen zurück oder sie melden einen negativen Wert, wenn keine Kollision vorliegt.

collision_point(x,y,obj,prec,notme) Diese Funktion prüft, ob an Punkt (x,y) eine Kollision mit Instanzen des Objektes obj vorliegt.

collision_rectangle(x1,y1,x2,y2,obj,prec,notme) Diese Funktion prüft, ob eine Kollision zwischen dem (gefüllten) Rechteck mit den angegebenen gegenüberliegenden Ecken und Instanzen des Objektes obj vorliegt. Beispielsweise kannst du sie verwenden, um zu prüfen, ob ein Gebiet frei von Hindernissen ist.

collision_circle(xc,yc,radius,obj,prec,notme) Diese Funktion prüft, ob eine Kollision zwischem dem (gefüllten) Kreis mit Kreismittelpunkt an Position (xc,yc) mit gegebenem Radius und Instanzen des Objektes obj vorliegt. Beispielsweise kannst du damit überprüfen, ob ein Objekt dicht an einer bestimmten Stelle liegt.

collision_ellipse(x1,y1,x2,y2,obj,prec,notme) Diese Funktion prüft, ob eine Kollision zwischen der (gefüllten) Ellipse mit den angegebenen gegenüberliegenden Ecken und Instanzen des Objektes obj besteht.

collision_line(x1,y1,x2,y2,obj,prec,notme) Diese Funktion prüft, ob eine Kollision zwischen dem Liniensegment von (x1,y1) zu (x2,y2) und Instanzen des Objektes obj vorliegt. Dies ist eine mächtige Funktion. Du kannst sie beispielsweise benutzen, um zu prüfen, ob eine Instanz eine andere Instanz "sehen" kann, indem Du prüfst, ob das Liniensegment zwischen den beiden eine Wand schneidet.

31.5 Instances(Instanzen)

Im Spiel sind die grundlegenden Einheiten die Instanzen der verschiedenen Objekte. Während des Spielverlaufs kannst du einige Aspekte dieser Instanzen verändern. Du kannst auch neue Instanzen erschaffen und/oder vernichten. Neben den oben besprochenen Bewegungsvariablen und den unten aufgeführten zeichnenbezogenen Variablen besitzt jede Instanz noch folgende Variablen:

object_index* Index des Objektes wovon dies hier eine Instanz ist. Diese Variable kann nicht verändert werden.

id* Der einzigartige Bezeichner für diese Instanz (>= 100000). (Beachte: Beim Erstellen von Räumen, wird die id immer angezeigt, wenn der Mauszeiger über einem Objekt ist.)

mask_index Index des "sprites", welches als Maske bei Kollisionen verwendet wird. Verwende einen Wert von -1, damit es dasgleiche wie der "sprite_index" benutzt.

solid Gibt an, ob die Instanz "solid" ist. Kann im Spiel verändert werden.

persistent Gibt an, ob die Instanz "persistent" (beständig) ist und nach einem Raumwechsel wieder erscheint.
Oftmals willst du das in gewissen Momenten abstellen (Beispielsweise wenn du zum ersten Raum zurück willst).

Es gibt nur ein Problem, wenn man sich mit Instanzen befasst. Es ist nicht gerade leicht einzelne Instanzen zu identifizieren. Sie haben keinen Namen. Wenn nur eine Instanz eines bestimmten Objektes vorhanden ist, kannst du den Namen des Objektes verwenden - Wenn jedoch mehrere Instanzen vorhanden sind, musst du
die id ermitteln. Dies ist eine einzigartige Identifizierung der Instanz. Du kannst sie bei "with statements" und als Objektbezeichner verwenden. Glücklicherweise gibt es eine Anzahl von Funktionen und Variablen, die dir helfen, die id zu ermitteln.

instance_count* Anzahl der Instanzen, welche momentan im Raum sind.

instance_id[0..n-1]* Die id der bestimmten Instanz. Wobei n die Nummer der Instanz ist. Beachte das die ID der Instanz sich bei jedem Schritt verändert, deswegen benutze nicht die Werte vorheriger Schritte.

Lass mich ein Beispiel geben. Angenommen jede Einheit in deinem Spiel hat ein bestimmte Stärke (power) und du willst die stärkste ermitteln - verwende folgenden Programmcode:

```
{
maxid = -1;
maxpower = 0;
for (i=0; i<instance_count; i+=1)
{
iii = instance_id[i];
if (iii.object_index == unit)
{
if (iii.power > maxpower)
{maxid = iii; maxpower =
iii.power;}
}
}
}
```

Nach der Schleife enthält maxid die id der Instanz mit der höchsten "power" (Stärke). (Vernichte keine Instanzen während eines solchen Schleifendurchlaufs, weil sie automatisch aus dem Array genommen werden und als Folge wirst du Instanzen "auslassen").

instance_find(obj,n) Gibt die id der (n+1)ten Instanz des Objektes obj an. obj kann ein Objekt oder das Schlüsselwort "all" sein. Wenn es nicht existiert, wird das besondere Objekt "noone"

zurückgegeben. Beachte das die ID der Instanz sich bei jedem Schritt verändert, deswegen benutze nicht die Werte vorheriger Schritte.

instance_exists(obj) Gibt an, ob eine Instanz des Objektes obj existiert. obj kann ein Objekt, eine Instanz-ID oder das Schlüsselwort "all" sein.

instance_number(obj) Gibt die Anzahl der Instanzen von Objekt obj an. obj kann ein Objekt oder das Schlüsselwort "all" sein.

instance_position(x,y,obj) Gibt die ID der Instanz des Objektes obj an Position (x,y) an. Falls mehrere Instanzen an dieser Stelle sind, wird nur die erste gemeldet. obj kann ein Objekt oder das Schlüsselwort "all" sein. Wenn es nicht existiert, wird das spezielle Objekt "noone" zurückgegeben.
instance_nearest(x,y,obj) Gibt die ID der Instanz von Objekt obj zurück, welche am nächsten am Punkt (x,y) ist. obj kan ein Objekt oder das Schlüsselwort "all" sein.

instance_furthest(x,y,obj) Wie oben nur jetzt die entfernteste Instanz von Objekt obj.

instance_place(x,y,obj) Gibt die ID der Instanz von Objekt obj an, welche getroffen wird, wenn die aktuelle Instanz an der Position (x.,y) gesetzt wird. obj kann ein Objekt oder das Schlüsselwort "all" sein. Wenn es nicht existiert, wird das besondere Objekt "noone" zurückgegeben.

Folgende Funktionen können verwendet werden, um Instanzen zu erschaffen und zu vernichten:

instance_create(x,y,obj) Erschaftt eine Instanz des Objektes obj an der angegebenen Position (x,y). Die Funktion liefert die ID der neuen Instanz zurück.

instance_copy(performevent) Erschafft eine Kopie der momentanen Instanz. Das Argument gibt an, ob das "creation event" ausgeführt wird für die Kopie. Die Funktion liefert die ID der neuen Kopie der Instanz.

instance_destroy() Vernichtet die momentane Instanz.

instance_change(obj,perf) Wechselt die Instanz nach obj. Ob ein "creation/destroy event" ausgelöst wird, gibt perf an.

position_destroy(x,y) Vernichtet alle Instanzen deren "sprites" Position (x,y) abdecken.

position_change(x,y,obj,perf) Wechselt alle Instanzen an Position (x,y) in obj. "perf" gibt an, ob das "create-/destroy event" ausgelöst wird.

31.6 Instanzen deaktivieren

Wenn du einen großen Raum erstellst, zum Beispiel bei Plattformspielen, kombiniert mit einem kleinen View, liegen viele Instanzen ausserhalb des sichtbaren Bereiches. Diese Instanzen sind aber trotzdem aktiv und führen ihre "events" durch. Auch bei Kollissionsprüfungen werden diese berücksichtigt. Dies kann viel CPU Leistung verbrauchen, was oft nicht erwünscht ist. (Es ist beispielsweise unwichtig ob sich Instanzen ausserhalb des Views bewegen.) Um dieses Problem zu lösen enthält Game Maker Funktionen zum Deaktivieren und Aktivieren von Instanzen. Bevor du sie benutzt musst du verstehen wie sie funktionieren.
Wenn du Instanzen deaktivierst sind sie in gewissem Sinne nicht existent für das Spiel. Sie sind nicht mehr sichtbar und ihre Events werden nicht ausgeführt.

Somit existieren sie für alle Aktionen und Funktionen nicht mehr. Dies spart Rechnerleistung aber sei vorsichtig. Wenn du z.B. alle Instanzen eines Objektes deaktivierst kannst du sie nicht löschen (Weil sie nicht existieren.). So kann ein vom Spieler gefundener Schlüssel keine deaktivierte Tür öffnen.

Der schlimmste Fehler, den du machen kannst, ist die Instanz zu deaktivieren welche für das Aktivieren zuständig ist. Um dies zu verhindern haben einige Funktionen hier die Möglichkeit sich nicht selbst zu deaktivieren

Dies sind die möglichen Funktionen:

instance_deactivate_all(notme) Deaktiviert alle Instanzen im Raum. Wenn notme true ist wird die ausführende Instanz nicht deaktiviert (Was normalerweise das ist was du willst.).

instance_deactivate_object(obj) Deaktiviert alle Instanzen im Raum für das gegebene Objekt. Du kannst auch "all" für alle Instanzen oder die ID einer bestimmten Instanz angeben.

instance_deactivate_region(left,top,width,height,inside,notme) Deaktiviert alle Instanzen in der angegebenen Region (dessen Umrisse im Bereich liegen). Wenn inside false ist werden alle Instanzen ausserhalb der Region deaktiviert.Wenn notme true ist wird die ausführende Instanz nicht deaktiviert (Was normalerweise das ist was du willst.).

instance_activate_all() Aktiviert alle Instanzen im Raum.

instance_activate_object(obj) Aktiviert alle Instanzen im Raum für das gegebene Objekt. Du kannst auch "all" für alle Instanzen oder die ID einer bestimmten Instanz angeben.

instance_activate_region(left,top,width,height,inside) Aktiviert alle Instanzen in der angegebenen Region. Wenn inside false ist werden alle Instanzen ausserhalb der Region deaktiviert.
Zum Beispiel, um alle Instanzen ausserhalb des Views zu deaktivieren kannst du den folgenden Code im Step Event des bewegenden Objekts angeben:

```
{
instance_activate_all();
instance_deactivate_region(view_left[0],view_top[0],
view_width[0],view_height[0],false,true);
}
```

Praktischerweise wirst du eine Region nutzen die etwas grösser ist als der View.

31.7 Timing(Abstimmung)

In guten Spielen sind die Dinge, welche im Spiel geschehen, gut aufeinander abgestimmt. Glücklicherweise erledigt der Game Maker die meisten Abstimmungsangelegenheiten für Dich. Er achtet darauf, das gewisse Dinge mit einer konstanten Rate abgearbeitet werden. Diese Rate wird beim Erstellen eines Raumes festgelegt Aber du kannst sie ändern, indem du die globale Variable "room_speed" verwendest. So kannst du beispielsweise die Spielgeschwindigkeit langsam erhöhen und dadurch das Spiel schwieriger werden lassen, indem du die "room_speed" bei jedem Schritt (step) um einen kleinen Wert erhöhst (z.B.: 0.001). Wenn dein Rechner langsam ist, wird die Spielgeschwindigkeit evtl. nicht erreicht werden können. Dies kann durch die Variable fps, welche kontinuierlich die Bilder pro Sekunde angibt, geprüft werden. Schliesslich gibt es noch für fortgeschrittenere "Timing-Fälle" die Variable "current_time", welche die Anzahl der verstrichenen Millisekunden seit Rechnerstart angibt. Hier ist eine Aufstellung der verfügbaren Variablen (nur die erste kann verändert werden).
room_speed Geschwindigkeit des Spiels im aktuellen Raum ("steps" pro Sekunde).

fps* Anzahl der gezeichneten Bilder pro Sekunde.

current_time* Anzahl der vergangenen Millisekunden seit Systemstart.

current_year* Das aktuelle Jahr.

current_month* Der aktuelle Monat.

current_day* Der aktuelle Tag.

current_weekday* aktueller Wochentag (1=Sonntag,...,7=Samstag).

current_hour* aktuelle Stunde.

current_minute* aktuelle Minute.

current_second* aktuelle Sekunde.

Manchmal willst du das Spiel für eine gewisse Zeit anhalten. Benutze dafür die "sleep"-Funktion.

sleep(numb) schläft numb Millisekunden lang.

Wie du wissen solltest, besitzt jede Instanz 8 verschiedene Alarmuhren (alarm clocks), welche du stellen kannst. Um die Werte, der diversen Uhren, zu verändern (oder, um sie in Erfahrung zu bringen) verwende folgende Variable:

alarm[0..7] Wert der angezeigten Alarmuhr (alarm clock). (Achtung: Alarmuhren werden nur aktualisiert, wenn das "alarm event" des entsprechenden Objektes auch "actions" enthält!)

Wir haben schon erfahren, das du für komplizierte Timing-Aufgaben die "time line" (Zeitlinie) Komponente verwenden kannst. Jede Instanz kann mit einer "time line" verknüpft werden. Folgende Variablen befassen sich damit:

timeline_index Index der "time line", welche mit der Instanz verknüpft ist. Du kannst hier die "time line" angeben, die verwendet werden soll.
Setze es auf -1, um ein Verwenden der "time line" zu unterbinden.

timeline_position Aktuelle Position innerhalb der "time line". Du kannst dies ändern, um Teile auszulassen oder zu wiederholen.
timeline_speed Normalerweise wird bei jedem Schritt (step) die Positionsmarke der "time line" um 1 erhöht. Du kannst diesen Betrag ändern, indem du der Variablen einen anderen Wert zuweist. Du kannst reele Zahlen verwenden, z. B. 0.5(auf den Punkt achten!). Wenn der Wert grösser als eins ist, können mehrere Momente in einem (im selben) Schritt (step) abgearbeitet werden. Sie werden der Reihe nach ausgeführt, so dass keine "action" ausgelassen wird.

31.8 Rooms and score (Räume und Spielstand)

Spiele finden in Räumen (rooms) statt. Jeder Raum besitzt einen Index, welcher durch den Raumnamen angegeben wird. Der aktuelle Raum ist in der Variablen "room" gespeichert. Du kannst nicht davon ausgehen, das Räume fortlaufend nummeriert sind. Addiere oder subtrahiere niemals einen Wert zu/von der
"room"-Variablen. Verwende besser die unten angeführten Funktionen und Variablen. Ein typisches Stück Programmcode würde in etwa so ausschauen:

```
{
if (room != room_last)
{
room_goto_next();
}
else
{
```

```
game_end();
}
}
```

Folgende Funktionen und Variablen befassen sich mit Räumen:

room_index des aktuellen Raumes; kann verändert werden, um zu einem anderen Raum zu gelangen - verwende lieber nachstehende Routinen dafür.

room_first* Index des ersten Raums des Spiels.

room_last* Index des letzten Raums des Spiels.

room_goto(numb) Springe zum Raum mit dem Index "numb".

room_goto_previous() Springe zum vorherigen Raum.

room_goto_next() Springe zum nächsten Raum.

room_restart() Starte den aktuellen Raum erneut.

room_previous(numb) Gibt den Index des Raumes vor dem Raum "numb" an (-1 = keiner) aber wechselt nicht dorthin.

room_next(numb) Gibt den Index des Raumes nach "numb" an (-1 = keiner).

game_end() Beendet das Spiel.

game_restart() Startet das Spiel erneut.

Räume haben einige zusätzliche Eigenschaften:

room_width* Breite des Raumes gemessen in Pixel.

room_height* Höhe des Raumes gemessen in Pixel.
room_caption Zeichenkette, die in der Titelleiste des Fensters vom Raum angezeigt wird.

room_persistent Gibt an, ob der aktuelle Raum persistent (dauerhaft/beständig) ist.

Viele Spiele bieten dem Spieler die Möglichkeit ein Spiel zu speichern und einen gespeicherten Spielstand zu laden. Im Game Maker geschieht dies automatisch, wenn der Spieler <F5> für das Speichern und <F6> für das Laden drückt. Du kannst auch Spiele Laden/Speichern, indem du GML-Programmcode verwendest (beachte, dass der Ladevorgang nur zum Ende des aktuellen Schrittes (step) stattfindet).

game_save(string) Speichert das Spiel unter dem Namen "string" ab.

game_load(string) Lädt einen Spielstand aus der Datei "string".

Ein anderer wichtiger Aspekt bei Spielen ist der Punktestand, die Gesundheit (der Spielfigur) und die Anzahl, der "Leben".

Game Maker führt Buch über den Punktestand in der globalen Variable "score" und die Anzahl der "Leben" in der globalen Variable "lives". Du kannst den Punktestand ändern, indem du einfach die Variable "score" veränderst. Das gleich gilt bei "health"(Gesundheit) und "lives"(Leben). Wenn die Variable "lives" grösser als 0 ist und kleiner oder gleich Null wird, wird das "no-more-lives event" für alle Instanzen ausgeführt. Wenn du nicht willst, dass "score" und "lives" in der Titelleiste des Fensters angegeben werden, setze die Variable "show_score", etc., auf "false". Auch kannst du die

Titelleiste verändern. Bei anspruchsvolleren Spielen, mach besser ne eigene Punktestand-Anzeige. score Der aktuelle Punktestand. lives Anzahl der "Leben". score Der aktuelle Punktestand.

lives Die aktuelle Anzahl der Leben.

health Die aktuelle "health" (0-100).

show_score Gibt an, ob der Punktestand in der Titelleiste des Fensters angezeigt werden soll.

show_lives dasselbe - nur für "Leben".

show_health wie oben - nur für "health" (Gesundheit).

caption_score Text der beim Score angezeigt wird.

caption_lives Text der bei den "Leben" angezeigt wird.

caption_health Text der bei "health" angezeigt wird.

Es gibt sogar einen eingebauten Mechanismus, der eine Highscoreliste für 10 Plätze verwaltet.

31.9 Generating events (auslösen von Ereignissen)

Wie du weisst, ist der Game Maker komplett "event"-gesteuert ist. Sämtliche "actions" geschehen als Resultat von "events". Es gibt viele unterschiedliche Ereignisse (events).

Creation(Erschaffen) und destroy (vernichten) "events" werden ausgelöst wenn eine Instanz erschaffen oder vernichtet wird. In jedem Schritt (step), bearbeitet das System zuerst die "alarm events". Danach werden Tastatur- und Mausereignisse verarbeitet und anschließend das "step event".

Dann werden die Instanzen an ihre neuen Positionen gesetzt und das "collision event" ausgewertet. Schliesslich wird das "draw event" verwendet, um die Instanzen zu
zeichnen (beachte, dass wenn mehrere "views" eingesetzt werden, das "draw event" mehrmals pro "step" aufgerufen wird). Du kannst auch ein Ereignis mittels GML-Programmcode einer Instanz zuweisen. Folgende Funktionen gibt es:

event_perform(type,numb) Löst das Ereignis "numb" mit angegebenen "type" für die aktuelle Instanz aus.

Folgende Ereignistypen können angegeben werden:
ev_create
ev_destroy
ev_step
ev_alarm
ev_keyboard
ev_mouse
ev_collision
ev_other
ev_draw
ev_keypress
ev_keyrelease

Wenn mehrere Ereignisse desselben Typen vorhanden sind, kann "numb" verwendet werden, das genaue Ereignis zu bestimmen. Für das "alarm event" liegt "numb" im Bereich 0 bis 7. Fürs

Tastaturereignis verwende den "keycode" (Tastencode) der Taste. Für Mausereignisse kannst du folgende Konstanten verwenden:

ev_left_button
ev_right_button
ev_middle_button
ev_no_button
ev_left_press
ev_right_press
ev_middle_press
ev_left_release
ev_right_release
ev_middle_release
ev_mouse_enter
ev_mouse_leave
ev_global_press
ev_global_release
ev_joystick1_left
ev_joystick1_right
ev_joystick1_up
ev_joystick1_down
ev_joystick1_button1
ev_joystick1_button2
ev_joystick1_button3
ev_joystick1_button4
ev_joystick1_button5
ev_joystick1_button6
ev_joystick1_button7
ev_joystick1_button8
ev_joystick2_left
ev_joystick2_right
ev_joystick2_up
ev_joystick2_down
ev_joystick2_button1
ev_joystick2_button2
ev_joystick2_button3
ev_joystick2_button4
ev_joystick2_button5
ev_joystick2_button6
ev_joystick2_button7
ev_joystick2_button8

Für das "collision event" gib den Index des anderen Objektes an. Schliesslich kannst du für die übrigen Ereignisse folgende Konstanten nutzen:

ev_outside
ev_boundary
ev_game_start
ev_game_end
ev_room_start
ev_room_end

ev_no_more_lives
ev_no_more_health
ev_animation_end
ev_end_of_path
ev_user0
ev_user1
ev_user2
ev_user3
ev_user4
ev_user5
ev_user6
ev_user7

Beim "step event" kannst du folgende Konstanten benutzen:

ev_step_normal
ev_step_begin
ev_step_end

event_perform_object(obj,type,numb) Diese Funktion arbeitet genau wie voranstehende, mit der Ausnahme, dass du "events" in anderen Objekten auslösen kannst. Beachte, dass die "actions" in diesen Ereignissen für die aktuelle Instanz gilt, nicht für das der Instanz zugeordnete Objekt.

event_user(numb) Beim Menüpunkt "other events" kannst du 8 benutzerdefinierte "events" erstellen. Diese werden nur ausgelöst, wenn du diese Funktion aufrufst. numb liegt im Bereich 0-7.

event_inherited() Führt das ererbte Ereignis aus. Funktioniert nur, wenn die Instanz ein "parent objekt" hat.

Du kannst Informationen über das momentan ausgeführte "event" bekommen, indem du folgende "nurlesen" Variablen verwendest:
event_type* Type des momentan ausgeführten Ereignisses.

event_number* Nummer des momentan ausgeführten "event".

event_object* Der Objektindex, für den das aktuelle Ereignis ausgeführt wird.

event_action* Der Index der "action" die gerade ausgeführt wird (0 ist die erste innerhalb des Ereignisses, etc.).

31.10 Sonstige Funktionen und Variablen

Hier sind einige Variablen und Funktionen die sich mit Fehlern befassen.

error_occurred Gibt an, ob ein Fehler aufgetreten ist

error_last Zeichenkette, die die letzte Fehlermeldung angibt

show_debug_message(str) Zeigt die Zeichenkette im " debug mode" (Entwanzugsmodus/Fehlersuchmodus)

Die folgenden Funktionen gibt es, um zu prüfen, ob bestimmte Variablen definiert sind und mit ihnen kannst du auch Variablen setzen und ihre Werte auslesen. Bei allen Funktionen wird der Variablenname als "string" (Zeichenkette) übergeben!

variable_global_exists(name) Gibt an, ob eine globale Variable mit der Bezeichnung "name" existiert.

variable_local_exists(name) Gibt an, ob eine lokale Variable mit Bezeichnung "name" für die aktuelle Instanz existiert.

variable_global_get(name) Gibt den Wert der globalen Varaiblen wieder.

variable_local_get(name) Liefert den Wert der lokalen Variablen.

variable_global_array_get(name,ind) Gibt den Wert des Index ind des globalen Arrays name an (ein String).

variable_global_array2_get(name,ind1,ind2) Gibt den Wert der Indexa ind1,ind2 des globalen 2-dimensionalen Arrays mit dem Namen name an (ein string).

variable_local_array_get(name,ind) Gibt den Wert des Index ind des lokalen Arrays name an (ein String).

variable_local_array2_get(name,ind1,ind2) Gibt den Wert der Indexa ind1,ind2 des lokalen 2-dimensionalen Arrays mit dem Namen name an (ein string).

variable_global_set(name,value) Setzt den Wert(value) für die globale Variable.

variable_global_array_set(name,ind, value) Setzt den Wert value des Index ind des globalen Arrays name (ein String) fest.

variable_global_array2_set(name,ind1,ind2, value) Setzt den Wert value der Indexa ind1,ind2 des globalen 2-dimensionalen Arrays mit dem Namen name (ein string) fest.

variable_local_set(name,value) dasselbe - nur für die lokale Variable.

variable_local_array_set(name,ind, value) Setzt den Wert des Index ind des lokalen Arrays name (ein String) fest.

variable_local_array2_set(name,ind1,ind2) Setzt den Wert der Indexa ind1,ind2 des lokalen 2-dimensionalen Arrays mit dem Namen name (ein string) fest.

Beispielsweise kannst du schreiben:

```
{
if variable_global_exists('ammunition';)
global.ammunition += 1
else
global.ammunition = 0
}
```

Du kannst diese Funktionen auch verwenden, um Variablen an Skripte zu übergeben - indem du ihre Bezeichnungen als Zeichenkette (string) übergibst und obige Funktionen verwendest, um sie zu verändern.

Du kannst die Priorität mit den folgenden Funktionen verändern:

set_program_priority(priority) Setzt die Priorität für das Programm fest. Du kannst Werte zwischen -3 und +3 wählen. Bei -3 läuft das Spiel nur wenn kein anderes Programm Prozessorleistung beansprucht oder im Leerlauf sind. Die Werte -2 und -1 sind unter Normal, deshalb werden andere

Prozesse mehr Priorität bekommen.0 ist der Normalwert. +1 und +2 geben eine höhere Priorität, woraus eine höhere Spielgeschwindigkeit und ein flüssigerer Ablauf sichergestellt werden. Aber andere Prozesse bekommen weniger Prozessorleistung. +3 ist der Echtzeitmodus. Im Echtzeitmodus wird die gesamte Prozessorleistung dem Spiel zugeteilt.

Dies kann zu ernsthaften Problemen mit anderen Programmen führen. Auch Tastenanschläge und das drücken des "X" in der rechten oberen Ecke können von Windows nicht mehr wahrgenommen werden. Benutze es nur wenn das Spiel im Exklusivmodus läuft und die ganze Prozessorleistung benötigt. Teste es vorsichtig bevor du es festsetzt.

32. Spielerinteraktion

Kein Spiel ohne Interaktion mit dem Spieler. Üblicherweise wird das im Game Maker mit "actions" (Aktionen) realisiert, die in Maus- oder Tastatur-"events" (Ereignisse) gesetzt werden. Manchmal braucht man mehr Kontrolle. Aus einem GML-Segment heraus, kann man prüfen, ob bestimmte Tasten auf der Tastatur gedrückt sind, ob ein Mausknopf gedrückt ist und wo die Mausposition ist. Normalerweise stecken so Sachen im "step-event"(Schrittereignis) eines "controller-object" (Programmsteuerungsobjekt) und leiten dort die entsprechenden "actions" ein. Folgende Variablen und Funktionen gibt es dafür:

mouse_x* X-Koordinate des Mauszeigers im Raum. Kann nicht beeinflusst werden.

mouse_y* Y-Koordinate des Mauszeigers im Raum. Kann nicht beeinflusst werden.

mouse_button Momentan gedrückte Maustaste. Mit den Werten: mb_none(keine-), mb_any(eine-), mb_left (linke-), mb_middle(mittlere-), oder mb_right(rechte Maustaste).

keyboard_lastkey Tastaturcode der zuletzt gedrückten Taste (Weiter unten für Tastaturkonstanten). Veränderbar - beispielsweise auf 0.

keyboard_key Tastaturcode der momentan gedrückten Taste (siehe unten; 0 wenn keine Taste gedrückt!).

keyboard_lastchar Zuletzt eingegebenes Zeichen (als string).

keyboard_string Liefert einen String mit den letzen - maximal 1024 - eingegebenen Zeichen. Dieser string enthält nur druckbare Zeichen. Liefert auch ein Ergebnis, wenn "Backspace" gedrückt wurde.

Manchmal ist es nützlich einer Taste eine andere zuzuordnen. Wenn man zum Beispiel dem Spieler erlauben will beides zu benutzen: Die Pfeiltasten und den Ziffernblock. Anstatt alle "actions" zu duplizieren kann man die Ziffernblocktasten verweisen (map) auf die Pfeiltasten. Vielleicht möchte man auch so etwas wie ein Tastaturoptionsmenu machen, wo der Spieler seine Spieltasten auswählt. Um so etwas zu machen, gibt es diese Funktionen im Game Maker.

keyboard_set_map(key1,key2) Weist der Taste mit Tastaturcode key1 den von Tastaturcode key2 zu.

keyboard_get_map(key) Gibt den aktuellen Tastaturverweis für Taste key wieder.

keyboard_unset_map() Alle Verweise zurücksetzen.

Um zu prüfen, ob eine bestimmte Taste oder Maustaste, kann man nachstehende Funktionen benutzen. Sie sind besonders nützlich, wenn mehrere Tasten gleichzeitig betätigt werden.

keyboard_check_direct(key) Prüft über die Hardware, ob Taste betätigt wird. Das Ergebnis ist unabhängig von der momentan benutzten Anwendung. Die Funktion erlaubt ein paar Tests mehr. Benutze die Tastaturcodes vk_lshift, vk_lcontrol, vk_lalt, vk_rshift, vk_rcontrol und vk_ralt, um die linke bzw. rechte Shift-Taste, Control- or Alt-Taste abzufragen. (Geht nicht unter Win95).

mouse_check_button(numb) Gibt an, ob ne Maustaste gedrückt wurde (benutze als Werte: mb_none(keine Maustaste), mb_left(link MT), mb_middle(mittlere MT), oder mb_right(rechte MT).)

Nachstehende Funktionen können den Tastaturstatus verändern:

keyboard_get_numlock() Gibt an, ob NUM-Lock gesetzt ist.

keyboard_set_numlock(on) Setzt (true) oder rücksetzt (false) den Numlock. (Nicht mit Win95.)

keyboard_key_press(key) Simuliert den Tastendruck der angegebenen Taste key.

keyboard_key_release(key) Simuliert das Loslassen der angegebenen Taste.

Folgende Konstanten für künstliche/virtuelle Tastaturcodes gibt es:
vk_nokey Gibt an, dass keine Taste gedrückt ist.

vk_anykey Gibt an, dass eine Taste gedrückt ist.

vk_left Tastencode für linke Pfeiltaste.

vk_right Tastencode für rechte Pfeiltaste.

vk_up Tastencode für hoch Pfeiltaste

vk_down Tastencode für runter Pfeiltaste.

vk_enter Tastencode für Enter-Taste.

vk_escape Tastencode für die linke Escape-Taste.

vk_space Tastencode für die space-Taste.

vk_shift Tastencode für die shift-Taste.

vk_control Tastencode für die linke control-Taste.

vk_alt Tastencode für die alt-Taste.

vk_backspace Tastencode für die backspace-Taste.

vk_tab Tastencode für die tab-Taste.

vk_home Tastencode für die home-Taste.

vk_end Tastencode für die end-Taste.

vk_delete Tastencode für die delete-Taste.

vk_insert Tastencode für die insert-Taste.

vk_pageup Tastencode für die pageup-Taste.

vk_pagedown Tastencode für die pagedown-Taste.

vk_pause Tastencode für die pause/break-Taste.

vk_printscreen Tastencode für die printscreen/sysrq-Taste.

vk_f1 bis vk_f12 Tastencodes für die Funktionstasten F1 bis F12

vk_numpad0 bis vk_numpad9 Zahlen auf dem Ziffernblock.

vk_multiply Multiplikationstaste auf dem Ziffernblock.

vk_divide Divisionstaste auf dem Ziffernblock.

vk_add Additionstaste auf dem Ziffernblock.

vk_subtract Subtraktionstaste auf dem Ziffernblock.

vk_decimal Dezimalzeichen auf dem Ziffernblock.

Für die Buchstabentasten benutze zum Beispiel ord('A') (Für Grossbuchstaben.) Folgende Konstanten können nur in keyboard_check_direct verwendet werden:

vk_lshift linke Shift-Taste

vk_lcontrol linke Control-Taste

vk_lalt linke Alt-Taste

vk_rshift rechte Shift-Taste

vk_rcontrol rechte Control-Taste

vk_ralt rechte Alt-Taste
Diese funktionieren nicht bei alten Windowsversionen (Win95)!!

Zum Beispiel: Angenommen du hast ein Objekt, welches der Spieler mit den Pfeiltasten lenken soll. Nimm dafür folgendes GML-Segment:

```
{
if (keyboard_check(vk_left)) x -= 4;
if (keyboard_check(vk_right)) x += 4;
if (keyboard_check(vk_up)) y -= 4;
if (keyboard_check(vk_down)) y += 4;
}
```

Natürlich ist es viel einfacher so etwas in Tastaturereignisse zu packen.

Es gibt noch einige weitere Funktionen:
keyboard_clear(key) Löscht den Tastenstatus. Kein Tastaturereignis bis Tastenwiederholfunktion einsetzt.

mouse_clear(button) Löscht den Status der Maustaste. Keine Ereignisse bis zum wiederholten Drücken der Taste.

io_clear() Löscht sämtliche Tastenzustände.

io_handle() Benutzereingabe bearbeiten, auffrischen von Tastatur und Mausstatus.

keyboard_wait() Warte auf Tastendruck.

Steuerknüppelunterstützung(Joystick Support)

Ein paar Funktionen für Joysticks gibt es. Um aber volle Kontrolle zu haben, gibt es einen ganzen Satz an Funktionen für Joysticks.
Game Maker unterstützt bis zu 2 Joysticks. Alle Funktionen brauchen die Joystick id als Argument.

joystick_exists(id) Gibt an, ob Joystick id (1 oder 2) existiert.

joystick_name(id) Gibt den Namen des Joysticks an.

joystick_axes(id) Achsen des Joysticks.

joystick_buttons(id) Anzahl der Knöpfe.

joystick_has_pov(id) Coolie-Hat vorhanden.

joystick_direction(id) Gibt den Tastencode (vk_numpad1 to vk_numpad9) entsprechenden der Richtung von joystick id (1 oder 2).

joystick_check_button(id,numb) Gibt an, ob Knopf gedrückt ist (numb Wertebereich 1-32).

joystick_xpos(id) Gibt die Position auf der Joystick X-Achse an (-1 to 1) der angegebenen Joystick id.

joystick_ypos(id) Hier für die Y-Position.

joystick_zpos(id) Wenn eine da ist, auch für die Z-Position.

joystick_rpos(id) Ruderstellung (oder vierte Achse).

joystick_upos(id) Gibt die u-Position (oder fünfte Achse) an.

joystick_vpos(id) Gibt die v-Position (oder sechste Achse) an.

joystick_pov(id) Coolie-Hat Position. Ein Winkel zwischen 0 and 360 Grad. 0 ist vorwärts, 90 ist nach rechts, 180 rückwärts und 270 nach links. Wenn unbetätigt liefert die Funktion -1 als Rückgabewert.

33. Spielegrafiken

Ein wichtiger Bestandteil von Spielen sind die Grafiken. Normalerweise achtet der Game Maker auf das meiste und bei einfachen Spielen, braucht man sich um nichts zu sorgen. Aber manchmal braucht man mehr Kontrolle. Für manche Fälle gibt es "actions" aber mit GML kann man mehr beeinflussen. Dieses Kapitel beschreibt alle Variablen und Funktionen, die man dafür braucht und gibt etwas mehr Informationen darüber, was wirklich passiert.

33.1 Window and Cursor (Fenster und Mauszeiger)

Voreingestellt läuft das Spiel in einem zentriertem Fenster. Der Spieler kann mit <F4> auf Vollbilddarstellung umschalten, wenn es nicht deaktiviert wurde. Man kann es auch vom Programm aus bewerkstelligen mit folgenden Variablen:

full_screen Bei Vollbilddarstellung ist der Wert dieser Variablen "true" (wahr). Du kannst den Modus wechseln, indem du die Variable "true" oder "false" setzt.

Beachte, dass im Vollbildmodus die Titelleiste und der Punktestand auf dem Bildschirm angezeigt werden. (Dies kann in den "game options" unterdrückt werden.) Im Vollbildmodus ist das Bild zentriert oder skaliert.
Du kannst es mit diesen Variablen steuern:

scale_window Diese Variable gibt den Skalierungsprozentsatz im Fenster-Modus an. (100 bedeutet keine Skalierung).

scale_full Diese Variable gibt den Skalierungsprozentsatz im Vollbild-Modus an. (100 entspricht keiner Skalierung - 0 zeigt maximal mögliche Skalierung an).

Skalierte Modi können auf Rechnern mit schwacher CPU bzw. Grafik-Karten langsam sein. Es gibt auch eine allgemeinere Funktion um den Grafikmodus während des Spiels zu ändern:

set_graphics_mode(exclusive,horres,coldepth,freq,fullscreen,winscale,fullscale) Diese Funktion ändert den Grafikmodus während der Ausführung. exclusive setzt fest, ob der Exklusive Modus genutzt wird. horres setzt die horizontale Auflösung fest (einer der folgenden Werte: 320, 640, 800, 1024, 1280, 1600). Benutze 0 um die Auflösung nicht zu ändern. Die korrekte vertikale Auflösung wird errechnet. coldepth bestimmt die Farbtiefe (16 oder 32). frequency legt die Bildwiederholfreqenz fest (einer der folgenden Werte: 60, 70, 75, 80, 100, 120). fullscreen legt fest, ob in den Vollbildmodus geschaltet wird. winscale gibt die Fensterskalierung an (1=keine Skalierung). fullscale gibt die Vollbildskalierung (1 = keine Skalierung, 0 = maximale Skalierung). Das ändern des Grafikmodus führt normalerweise zu kurzem Flackern. Benutze es nur wenn unbedingt nötig. Du kannst dies dazu nutzen um den Benutzer über die Auflösung entscheiden zu lassen.
Nur in der registrierten Version!

Voreingestellt läuft jedes Spiel mit sichtbarer Positionsmarke (cursor). Für viele Spiele möchtest du das nicht. Um die Positionsmarke zu entfernen, benutze die Variable:

show_cursor Wenn der Wert "false" ist, ist die Positionsmarke unterdrückt, sonst sichtbar (true).

Du kannst die Positionsmarke auch durch eine der vielen vordefinierten von Windows ersetzen, indem du folgende Funktion verwendest:

set_cursor(cur) setzt die Positionsmarke auf den angegebenen Wert. Du kannst folgende Konstanten verwenden: cr_default
cr_none
cr_arrow
cr_cross
cr_beam
cr_size_nesw
cr_size_ns
cr_size_nwse
cr_size_we
cr_uparrow
cr_hourglass
cr_drag
cr_nodrop
cr_hsplit
cr_vsplit
cr_multidrag
cr_sqlwait
cr_no
cr_appstart
cr_help
cr_handpoint
cr_size_all

Kurz angemerkt: Beachte, dass es sehr einfach ist, deine eigenen Mauszeiger zu erstellen, nutze dafür folgende Funktion:

cursor_sprite Gibt das Sprite das den Mauszeiger darstellen soll an. Der Wert -1 (default) gibt an das kein Sprite gezeichnet werden soll. Wenn das Sprite animiert ist wird es auch animiert gezeichnet.

Um die Position des Mauszeigers zu verändern kannst du folgende Aktion nutzen:

mouse_set_screen_position(x,y) Setzt den Mauszeiger auf eine bestimmte Position auf dem Bildschirm. (Beachte das du eine Position auf dem Bildschirm angeben musst, nicht eine im Raum. Wenn du es vom Raum abhängig machen willst musst du die Position selbst berechnen.)

Um die Auflösung des Monitors herauszufinden kannst du diese nur auslesbaren Variablen verwenden:
monitor_width Die Breite des Monitors in Pixeln

monitor_height Die Höhe des Monitors in Pixeln.

Um Informationen über die Position des Fensters auf dem Bildschirm zu bekommen, nutze diese Funktionen:

window_left Die Position des linken Fensterrandes auf dem Monitor. Du kannst damit (nur im Fenstermodus) die Position des linken Fensterrandes verändern.

window_top Die Position des oberen Fensterrandes auf dem Monitor. Du kannst damit (nur im Fenstermodus) die Position des oberen Fensterrandes verändern.

Folgende Funktionen können nur ausgelesen werden:

window_width Die Breite des Spieles auf dem Monitor.

window_height Die Höhe des Spieles auf dem Monitor

window_client_left Die Position des linken Client-Fensterrandes auf dem Monitor
window_client_top Die Position des oberen Client-Fensterrandes auf dem Monitor

window_client_width Die Breite des Client-Fensters auf dem Monitor

window_client_height Die Höhe des Client-Fensters auf dem Monitor

33.2 Sprites und Images (Bildfolgen und Bilder)

Jedes Objekt kann mit einem "sprite" verbunden werden. Es kann ein Einzelbild oder eine Bildfolge sein. Für jede Instanz des Objektes zeichnet das Programm das korrespondierende Bild auf den Bildschirm, wobei der Bezugspunkt (origin) (wie in den "sprite properties" definiert) auf die X,Y-Position der Instanz gelegt wird.
Wenn eine Bildfolge vorhanden ist, werden alle Animationsschritte der Reihe nach gezeichnet, um einen Animationseffekt zu erhalten. Es gibt eine Vielzahl von Variablen, welche die Darstellung der Bilder beeinflussen. Sie können verwendet werden, um unterschiedliche Effekte zu erzielen. Jede Instanz besitzt folgende Variablen:

visible Wenn "visible" (sichtbar) auf 1 (true/wahr) gesetzt wird, wird das Bild dargestellt, sonst nicht. Unsichtbare Instanzen sind trotzdem aktiv und verursachen "collision events"; Du siehst sie nur nicht! Die Sichtbarkeit auf Null zu setzen ist zum Beispiel für "controller objects" (Steuerungsobjekte) (mach sie "nonsolid" (nicht massiv) um "collision events" zu vermeiden) oder versteckte Schalter sehr nützlich.

sprite_index Dies ist der Index (lat. Anzeiger) des momentanen sprites der Instanz. Du kannst ihn ändern, um der Instanz ein anderes sprite zuzuweisen. Als Wert kannst du die Bezeichnungen/Namen der verschiedenen sprites verwenden, welche du schon erstellt hast. Das Wechseln des sprites ändert nicht den Index des angezeigten "sub-image" (Bild einer Bildfolge).
sprite_width* Gibt die Breite des sprites an. (Dieser Wert ist unveränderbar aber möglicherweise willst du ihn trotzdem benutzen).

sprite_height* Gibt die Höhe des sprites an. (Auch dieser Wert ist unveränderbar.)
sprite_xoffset* Gibt den horizontalen Versatz (offset) des sprites an, wie in den "sprite properties" festgelegt. (Auch unveränderbar.)

sprite_yoffset* Vertikaler Versatz. (unveränderbar.)

image_number* Bildnummer eines Bildes einer Bildfolge der Instanz (unveränderbar).

image_index Wenn ein sprite aus einer Bildfolge besteht, wird diese zyklisch wiederholt. Diese Variable zeigt das momentan dargestellte Bild der Bildfolge an (Nummerierung beginnt bei 0). Du kannst das momentan angezeigte Bild ändern, indem du den Wert dieser Variablen änderst. Das Programm setzt, beginnend mit dem angegebenem Bild, die zyklische Wiederholung fort.

image_single Manchmal willst du nur ein spezielles Bild einer Bildfolge darstellen und nicht immer den ganzen Zyklus. Das kannst du erreichen, indem du diese Variable auf den Index des Bildes setzt, welches dargestellt werden soll (Das erste Bild der Bildfolge hat den Index 0). Vergebe den Wert -1 um zyklisch die Bildfolge wiederzugeben. Dies ist nützlich, wenn ein Objekt unterschiedliche Erscheinungsbilder hat.

Angenommen du hast ein Objekt, welches sich drehen kann und du hast eine Bildfolge worin alle Bilder, je nach Ausrichtung, enthalten sind (entgegen dem Uhrzeigersinn). Dann kannst du im "step event" des Objekts folgendes setzen:

```
{
image_single = direction * image_number/360;
}
```

image_speed Die Zyklusgeschwindigkeit. Ein Wert von 1 bedeutet pro "step" ein Bild weiter. Kleinere Werte bedeuten eine langsamere Zyklusgeschwindigkeit, wobei jedes Bild der Bildfolge mehrfach gezeichnet wird.
Grössere Werte bedeuten, dass einzelne Bilder der Bildfolge übersprungen werden, um die Bewegung schneller zu gestalten.

depth Normalerweise werden Bilder dargestellt in der Reihenfolge, wie die Instanzen erstellt werden. Du kannst dies ändern, indem du eine "image depth" (Zeichenebene) festlegst. Voreingestellt ist 0, sofern du keinen anderen Wert in den "object properties" angegeben hast. Je höher der Wert, desto weiter ist die Instanz "entfernt". (Du kannst auch negative Werte einsetzen.) Instanzen mit höherer "depth" werden hinter Instanzen mit niedrigerer "depth" dargestellt. Durch festlegen der "depth" wird sichergestellt, dass die Instanzen in der von dir gewünschten Anordnung dargestellt werden (z.B. Ein Flugzeug vor den Wolken). Hintergrundinstanzen sollten eine hohe (positive) "depth" haben und Vordergrundinstanzen eine niedrige (negative) "depth".

image_xscale Ein Skalierungsfaktor, um die Bilder grösser oder kleiner darzustellen. Ein Wert von 1 gibt die Normalgrösse an. Ein Ändern der Skalierung beeinflusst auch die Bildbreite und die "collision events", wie du vermutlich schon erwartet hast. Beachte das skalierte Bilder (besonders kleinergemachte) mehr Zeichnungszeit brauchen. Ändern der Skalierung kann verwendet werden, um 3D-Effekte zu simulieren.

image_xscale Dasselbe nur für die Höhe.

image_alpha Transparenzwert(alpha) der bei der Darstellung verwendet wird. 1 ist der normale Wert; ein Wert von 0 ist total transparent. Verwende es mit Vorsicht. Teilweise transparente Bilder brauchen sehr viel Zeit und verlangsamen das Spiel.

bbox_left* Linke Seite des Begrenzungsrahmens (bounding box) vom Bild der Instanz (Skalierung berücksichtigt).

bbox_right* Rechte Seite des Begrenzungsrahmens.

bbox_top* Oberkante des Begrenzungsrahmens.

bbox_bottom* Unterkante des Begrenzungsrahmens.

33.3 Backgrounds (Hintergrundbilder)

Jeder Raum kann bis zu acht Hintergründe haben und eine Hintergrundfarbe. Alle diese Aspekte von Hintergründen kannst du mit einem Codeschnipsel (piece of code) und mittels folgender Variablen ändern (Manche sind Arrays mit Wertebereich 0-7, die den Hintergrund angeben):

background_color Hintergrundfarbe des Raumes.

background_showcolor Gibt an, das Fenster in der Hintergrundfarbe zu füllen.

background_visible[0..7] Gibt an, ob dieses Hintergrundbild sichtbar ist.

background_foreground[0..7] Gibt an, ob das Hintergrundbild im Vordergrund ist.

background_index[0..7] Hintergrundbildindex für den Hintergrund.

background_x[0..7] X Position des Hintergrundbilds.

background_y[0..7] Y Position des Hintergrundbilds.

background_width[0..7]* Breite des Hintergrundbilds.

background_height[0..7]* Höhe des Hintergrundbilds.

background_htiled[0..7] Gibt an, ob das Hintergrundbild horizontal gekachelt(tiled) ist.

background_vtiled[0..7] Gibt an, ob das Hintergrundbild vertikal gekachelt(tiled) ist.

background_xscale[0..7] Horizontaler Skalierungsfaktor für den Hintergrund.

background_yscale[0..7] Vertikaler Skalierungsfaktor für den Hintergrund.

background_hspeed[0..7] Horizontaler "scrolling speed" (Verschiebung) des Hintergrundes (pixels pro step).

background_vspeed[0..7] Vertikaler "scrolling speed" (Verschiebung) des Hintergrundes (pixels pro step).

background_alpha[0..7] Transparenzwert (alpha) der zum Zeichnen des Hintergrundes benutzt wird. Ein Wert von 1 entspricht den normalen Einstellungen; ein Wert von 0 bedeutet totale Transparenz. Vorsichtiger Einsatz empfohlen, da das Zeichnen von teilweise transparenten Hintergründen zeitintensiv ist und das Spiel verlangsamt.

33.4 Tiles (Kacheln)

Wie du wissen solltest, kannst du den Räumen Kacheln zufügen. Eine Kachel(tile) ist Teil der Hintergrundbestandteile. Kacheln sind einfach sichtbare Bilder. Sie reagieren nicht auf Ereignisse (events) und erzeugen keine Kollisionen. Deshalb werden sie wesentlich schneller bearbeitet als Objekte. Alles, was keine Ereignisse oder Kollisionen verwendet, kann am Besten durch Kacheln dargestellt werden. Auch ist es oft besser eine Kachel (tile) zu verwenden, um die ansprechende Grafik darzustellen und ein einfaches Objekt, um zum Beispielsweise Kollisionen zu erzeugen.

Du hast mehr Kontrolle über "tiles" (Kacheln) als du möglicherweise denkst. Du kannst sie hinzufügen, wenn du den Raum gestaltest oder aber auch während des Spiels. Du kannst ihre Position ändern und sie skalieren oder transparent machen. Eine Kachel (tile) hat folgende Eigenschaften:

background der Hintergrundbestandteil, von dem die Kachel entnommen wurde.

left, top, width, height (links, oben, Breite, Höhe). Der Teil des Hintergrundes, der benutzt wird.
x,y Die Position der linken oberen Ecke der Kachel in dem Raum.

depth Die Zeichenebene der Kachel. Du kannst jede beliebige "depth" angeben, um den Effekt zu erzielen, dass "tiles" zwischen Objektinstanzen erscheinen.

visible Gibt an, ob Kachel sichtbar ist.

xscale, yscale Jede Kachel kann auch skaliert werden (vorgegeben ist 1).

alpha Ein alpha-Wert gibt die Transparenz. 1 = nicht transparent, 0 = total transparent. Du solltest dies mit Vorsicht benutzen, weil transparente Kacheln langsam zu zeichnen sind und auf manchen Systemen zu Problemen führen.

Um die Eigenschaften eines "tiles" zu ändern, brauchst du seine id. Bei der Raumerstellung wird die

id in der Info-Leiste (unten) angezeigt. Es gibt auch eine Funktion, um die id einer Kachel an einer bestimmten Position zu ermitteln.
Folgende Funktionen befassen sich mit "tiles":

tile_add(background,left,top,width,height,x,y,depth) Fügt eine Kachel mit den angegebenen Werten dem Raum hinzu (Erklärung siehe oben). Die Funktion gibt die id an, welche später verwendet werden kann.

tile_delete(id) Löscht die Kachel mit der angegebenen id.

tile_exists(id) Gibt an, ob eine Kachel mit der id vorhanden ist.

tile_get_x(id) Gibt die x-Position der Kachel mit der angegebenen id an.

tile_get_y(id) Das gleiche für die y-Position.

tile_get_left(id) Hier für die linken Wert.

tile_get_top(id) Und hier für den oberen Wert.

tile_get_width(id) Gibt die Kachelbreite der genannten (id) an.

tile_get_height(id) Gibt die Kachelhöhe an.

tile_get_depth(id) Liefert die Zeichenebene (depth) der Kachel mit der angegebenen id.

tile_get_visible(id) Gibt an, ob die Kachel mit id sichtbar bzw. unsichtbar ist.

tile_get_xscale(id) Gibt die X-Skalierung an.

tile_get_yscale(id) Gibt die Y-Skalierung an.

tile_get_background(id) Gibt den "background" der Kachel mit der id an.

tile_get_alpha(id) Liefert den alpha-Wert der Kachel.

tile_set_position(id,x,y) Setzt die x,y-Position.

tile_set_region(id,left,right,width,height) Legt den Bereich (region) der Kachel mit der angegebenen id in dessen "background".

tile_set_background(id,background) Setzt den Hintergrund (background) für die Kachel.

tile_set_visible(id,visible) Legt fest, ob die Kachel sichtbar ist.

tile_set_depth(id,depth) Legt die Zeichenebene fest.

tile_set_scale(id,xscale,yscale) Setzt die Skalierung der Kachel.

tile_set_alpha(id,alpha) Legt die Transparenz fest.

Die folgenden Funktionen behandeln Ebenen (Layer), welche Sammlungen von Tiles der selben Tiefe darstellen.

tile_layer_hide(depth) Versteckt alle Tiles in dieser Ebene.

tile_layer_show(depth) Zeigt alle Tiles in dieser Ebene.

tile_layer_delete(depth) Löscht alle Tiles in dieser Ebene.

tile_layer_shift(depth,x,y) Verschiebt alle Tiles in dieser Ebene um die Vektoren x und y. Kann genutzt werden um sich bewegende Hintergründe zu erschaffen.

tile_layer_find(depth,x,y) Gibt die id des Tiles an Position x,y in der Ebene an. Falls dort keines ist wird -1 zurückgegeben. Wenn mehrere auf dem selben Punkt in der selben Ebene existieren wird

die id des ersten zurückgegeben.

tile_layer_delete_at(depth,x,y) Löscht das Tile an Position x,y in der Ebene. Wenn meherer Tiles an der selben Position in der selben Ebene sind, werden alle gelöscht.

tile_layer_depth(depth,newdepth) Ordnet alle Tiles der Ebene einer neuen Ebene zu. Mit dieser Funtkion kannst du ganzen Ebenen eine neue Tiefe geben.

33.5 Drawing functions (Zeichenfunktionen)

Es ist möglich Objekte ganz anders als ihr Abbild aussehen zu lassen. Es gibt eine ganze Sammlung von Funktionen, um unterschiedliche Formen darzustellen. Unter anderem gibt es auch Funktionen, um Text darzustellen. Du kannst sie nur im "draw event" eines Objektes verwenden; diese Funktionen ergeben woanders keinen Sinn. Beachte, dass nur die Grafikkarte eines PCs die Darstellung von Bildern schnell macht. Folglich sind andere Zeichenfunktionen relativ langsam. Game Maker ist optimiert für Bilddarstellungen. Vermeide also andere Zeichenfunktionen wann immer möglich. (Wenn es irgendwie geht, mach ein Bitmap stattdessen). Beachte auch, dass Kollisionen zwischen Instanzen mit Hilfe ihrer "sprites" bzw. "masks" ermittelt werden und nicht anhand dessen, was du gemalt hast. Folgende "image"-bezogene Funktionen gibt es:

draw_sprite(sprite,subimg,x,y) Zeichnet das Bild subimg(-1 entspricht dem aktuellen) der Bildfolge(sprite) mit Index sprite , mit dem Bezugspunkt (x,y).

draw_sprite_ext(sprite,subimg,x,y,xscale,yscale,alpha) Zeichnet das Sprite skaliert mit den Faktoren x,y. alpha gibt die Dursichtigkeit des Sprites an. Ein Wert von 0 macht es komplett transparent. Ein Wert von 1 komplett undurchsichtig. Diese Funktion kann grossartige Effekte erzeugen (z.B. transparente Explosionen). Aber es ist langsam bei einem Wert <1 da es per Software realisiert wird. Benutze es mit Bedacht.

draw_sprite_stretched(sprite,subimg,x,y,w,h) Zeichnet das "sprite" gestreckt, so dass es den angegeben Bereich ausfüllt (x,y = linke obere Ecke; w= "width" (Breite) und h= Höhe).

draw_sprite_stretched_ext(sprite,subimg,x,y,w,h,alpha) Zeichnet das "sprite" gestreckt, so dass es den angegeben Bereich ausfüllt (x,y = linke obere Ecke; w= "width" (Breite) und h= Höhe). alpha legt die Transparenz fest.

draw_sprite_tiled(sprite,subimg,x,y) Zeichnet das "sprite" gekachelt, so dass es den ganzen Raum ausfüllt. (x,y) ist die Stelle an der ein "sprite" gezeichnet wird.

draw_sprite_tiled_ext(sprite,subimg,x,y,xscale,yscale,alpha) Zeichnet das "sprite" gekachelt, so dass es den ganzen Raum ausfüllt aber nun mit Skalierung und Transparenz. Aber es ist langsam bei einem Wert <1 da es per Software realisiert wird. Benutze es mit Bedacht.

draw_sprite_part(sprite,subimg,left,top,right,bottom,x,y) Zeichnet den Teil des Subimages subimg (-1=aktuelles) des Sprites sprite an Position x,y.

draw_sprite_part_ext(sprite,subimg,left,top,right,bottom,x,y,xscale,yscale,alpha) Zeichnet den Teil des Subimages subimg (-1=aktuelles) des Sprites sprite an Position x,y aber nun mit Skalierung und Transparenz.

draw_sprite_alpha(sprite,subimg,x,y,xscale,yscale,alphaspr,ind) Zeichnet das Sprite mit xscale und yscale skaliert. Der alpha Faktor wird nun per Pixelbasis über die Intensität des Sprites alphaspr ermittelt. Von dem können wir Subimage img nehmen. Ein schwarzer Pixel macht es komplett transparent, ein Weißer komplett undurchsichtig. Diese Funktion kann grossartige Effekte erzeugen (z.B. transparente Explosionen). Aber es ist langsam bei einem Wert <1 da es per Software realisiert wird. Benutze es mit Bedacht. Nur in der registrierten Version.

draw_sprite_part_alpha(sprite,subimg,left,top,right,bottom,x,y,xscale,yscale,alphaspr,ind) Zeichnet den Teil des Subimages subimg (-1=aktuelles) des Sprites sprite an Position x,y aber nun mit Skalierung und Transparenz per alphaspr.
Nur in der registrierten Version.

draw_background(back,x,y) Stellt den Hintergrund mit Index back an Position x,y dar.

draw_background_ext(back,x,y,xscale,yscale,alpha) Zeichnet den Hintergrund skaliert mit alpha (0-1) (langsam wenn alpha < 1!).

draw_background_stretched(back,x,y,w,h) Zeichnet den Hintergrund gestreckt im angegebenen Bereich.

draw_background_stretched_ext(bacl,x,y,w,h,alpha) Zeichnet den Hintergrund gestreckt im angegebenen Bereich. alpha gibt die Transparenz an.

draw_background_tiled(back,x,y) Kachelt den Hintergrund, damit er den ganzen Raum ausfüllt.

draw_background_tiled_ext(back,x,y,xscale,yscale,alpha) Kachelt den Hintergrund, damit er den ganzen Raum ausfüllt aber mit Skalierung und Transparenz.

draw_background_part(back,left,top,right,bottom,x,y) Zeichnet den Teil des Hintergrundes n an Position x,y.

draw_background_part_ext(back,left,top,right,bottom,x,y,xscale,yscale,alpha) Zeichnet den Teil des Hintergrundes n an Position x,y aber mit Skalierung und Transparenz.

draw_background_alpha(back,x,y,xscale,yscale,alphaback) Zeichnet den Hintergrund mit xscale und yscale skaliert. Der alpha Faktor wird nun per Pixelbasis über die Intensität des Hintergrundes alphaback ermittelt. Ein schwarzer Pixel macht es komplett transparent, ein Weißer komplett undurchsichtig. Es ist langsam bei einem Wert <1 da es per Software realisiert wird. Benutze es mit Bedacht. Nur in der registrierten Version.

draw_background_part_alpha(back,left,top,right,bottom,x,y,xscale,yscale,alphaback) Zeichnet den Teil des Hintergrundes n an Position x,y, aber mit Skalierung und Transparenz per alphaback.
Nur in der registrierten Version.

Die folgenden Funktionen zeichnen Grundformen. Sie benutzen eine Anzahl an Einstellungen, speziell "brush" (Pinsel) und "pen" (Stift) Farbe, welche durch entsprechende Variablen gekennzeichnet sind.

draw_pixel(x,y) Zeichnet ein Pixel in der Pinselfarbe an der Stelle x,y.

draw_getpixel(x,y) Gibt den Farbwert des Pixels an Position x,y wieder.

draw_fill(x,y) Ausfüllen (Flood fill) in Pinselfarbe ab Position x,y.

draw_line(x1,y1,x2,y2) Zeichnet eine Linie von(x1,y1) nach (x2,y2).

draw_arrow(x1,y1,x2,y2,size) Zeichnet einen Pfeil von (x1,y1) nach (x2,y2). size gibt die grösse des Pfeils in Pixeln an.

draw_circle(x,y,r) Zeichnet einen Kreis mit Mittelpunkt (x,y) und Radius r.

draw_ellipse(x1,y1,x2,y2) Zeichnet eine Ellipse.

draw_rectangle(x1,y1,x2,y2) Zeichnet ein Rechteck.

draw_roundrect(x1,y1,x2,y2) Zeichnet ein Rechteck mit abgerundeten Ecken.
draw_triangle(x1,y1,x2,y2,x3,y3) Zeichnet ein Dreieck.

draw_arc(x1,y1,x2,y2,x3,y3,x4,y4) Zeichnet den Bogen einer Ellipse.

draw_chord(x1,y1,x2,y2,x3,y3,x4,y4) Zeichnet eine Sehne(chord) in der Ellipse.

draw_pie(x1,y1,x2,y2,x3,y3,x4,y4) Zeichnet einen Ausschnitt einer Ellipse.

draw_button(x1,y1,x2,y2,up) Zeichnet einen Knopf, up gibt an, ob nach oben (1) oder nach unten (0).

draw_text(x,y,string) Zeichnet den "string" (Zeichenkette) an Position x,y. Ein # Symbol oder "carriage return" chr(13) oder "linefeed" chr(10) werden als Zeilenende gewertet; so kannst du mehrzeilige Texte zeichnen. (Verwende \# um das # Symbol darzustellen.)

draw_text_ext(x,y,string,sep,w) Ähnlich wie die vorangehende Routine aber du kannst 2 Dinge mehr festlegen. sep gibt den Zeilenabstand an. Benutze -1 für den Standardabstand. Verwende w, um die Textbreite in Pixeln anzugeben. Zeilen werden bei Leerzeichen und Bindestrichen umgebrochen. Verwende -1, wenn du keinen Zeilenumbruch willst.

draw_text_sprite(x,y,string,sep,w,sprite,firstchar,scale) Text mit der obigen Funktion darzustellen ist relativ kostspielig. Diese Funktion hier arbeitet genauso, nimmt aber die einzelnen Zeichen aus einem "sprite".
Dieses "sprite" muss jedes Zeichen in seiner Bildfolge enthalten. Das erste Zeichen entspricht "firstchar". Ab diesem Zeichen folgen die restlichen Zeichen der ASCI-Reihenfolge. Du kannst die Zeichentabbelle von Windows verwenden, um die korrekte Abfolge zu bestimmen. Wenn du nur einige der ersten brauchst (z.B. die Ziffern oder Grossbuchstaben) können die restlichen entfallen. scale gibt den Skalierungsfaktor an (1 ist normale Grösse). Auch hier gilt wieder: Skalierung ist langsam. Beachte, dass diese "sprites" recht gross sind und du offensichtlich keine Kollisionsabfrage benötigst.

draw_polygon_begin() Start der Polygonzeichnungsbeschreibung.

draw_polygon_vertex(x,y) Fügt den Eckpunkt(x,y)zum Polygon.

draw_polygon_end() Ende der Polygonzeichnungsbeschreibung. Diese Funktion zeichnet es.

draw_path(path,x,y,absolute) Mit dieser Funktion kannst du den Pfad im Raum ab seiner Startposition (x,y) zeichnen. Wenn absolute true ist wird der Pfad dort gezeichnet wo er erzeugt wurde und x und y werden ignoriert.

draw_healthbar(x1,y1,x2,y2,amount,backcol,mincol,maxcol,direction,showback,showborder) Mit dieser Funktion kannst die Lebensleiste zeichnen (oder irgeneine andere Leiste die einen Wert darstellt, beispielsweise Schaden). Die Argumente x1, y1, x2 und y2 geben den Platz für die Leiste an. amount gibt in Prozent den Wert der gezeichnet werden soll an (zwischen 0 und 100). backcol ist die Hintergrundfarbe der Leiste. mincol und maxcol geben den Farbwert für 0 und 100 an. Für einen Wert dazwischen wird die Farbe berechnet. So kannst du beispielsweise leicht eine Leiste von Rot nach Grün erzeugen. direction gitb die Richtung an, in die die Leiste gezeichnet wird. 0 beudeutet links, 1 rechts, 2 oben und 3 unten. Schlielich gibt showback an ob eine Hintergrundbox und showborder ob ein schwarzer Rahmen gezeichnet werden soll.

Du kannst eine Menge Einstellungen ändern, wie beispielsweise die Linienfarbe (pen), Bereich (brush) und font (Schriftart), und viele ander Schrifteinstellungen. Der Effekt dieser Variablen ist global! Wenn du sie in der Zeichenroutine eines Objektes änderst, werden sie so auch bei später gezeichneten Objekten wirksam. Du kannst diese Variablen auch in anderen "events" (Ereignissen) verwenden. Zum Beispiel, wenn sie nicht verändert werden, kannst du sie zu Beginn des Spiels festlegen(was wohl effektiver ist).
brush_color Füllfarbe für Formen. Eine ganze Anzahl von vordefinierten Farben gibt es: c_aqua c_black

c_blue
c_dkgray
c_fuchsia
c_gray
c_green
c_lime
c_ltgray
c_maroon
c_navy
c_olive
c_purple
c_red
c_silver
c_teal
c_white
c_yellow
Andere Farben erhält man unter Verwenden folgender Routine:

make_color(red,green,blue), wobei red(rot), green(grün) and blue(blau) Werte von 0 bis 255 haben können.

brush_style Verwendete Pinselart für Farbfüllungen. Folgende Arten gibt es: bs_hollow
bs_solid
bs_bdiagonal
bs_fdiagonal
bs_cross
bs_diagcross
bs_horizontal
bs_vertical
pen_color Farbe des Stiftes für Rahmen.
pen_size Grösse des Stiftes in Pixel.
font_color Schriftfarbe.
font_size Schriftgrösse (in points).
font_name Schriftart (ein "string").
font_style Art der Schrift. Folgende Arten gibt es (kombinierbar): fs_normal
fs_bold
fs_italic
fs_underline
fs_strikeout
font_angle Drehwinkel der Schrift (0-360 Grad) Zum Beispiel, Für vertikalen Text benutze einen Wert von 90.
font_align Ausrichtung des Textes (Links, Mittig oder Rechts) an der gegebenen Position. Folgende Werte können verwendet werden: fa_left
fa_center
fa_right

font_pixels_per_inch* Der Benutzer eines PCs kann die Anzahl der Pixel pro Inch verändern um z.B. grössere Schriftarten zu bekommen. Dies kann zu einem Problem in Spielen werden, da die Schriftart skaliert wird und nicht mehr passt. Diese nur lesbare Variable gibt die aktuellen Pixel pro Inch zurück. Die normale Windows Einstellung ist 96. (Eine grosse Einstellung ist 120.) Du kannst diese Funtkion nutzen um zu prüfen ob es sich verändert hat und, wenn benötigt, die Schriftgrösse im Spiel daran anpassen.

Die folgenden Routinen arbeiten mit Farben:

make_color_rgb(red,green,blue) Gibt den Farbwert mit den angegebenen Rot-, Grün- und Blaukomponenten wieder, wobei red, green und blue Werte zwischen 0 und 255 haben sollen.

make_color_hsv(hue,saturation,value) Gibt den Farbwert mit den angegebenen hue, saturation and value Komponenten wieder (jeder zwischen 0 und 255).

color_get_red(col) Gibt den Rotanteil des Farbwertes wieder.

color_get_green(col) Gibt den Grünanteil des Farbwertes wieder.

color_get_blue(col) Gibt den Blauanteil des Farbwertes wieder.

color_get_hue(col) Gibt die Hue Komponente des Farbwertes wieder.

color_get_saturation(col) Gibt die Saturation Koponente des Farbwertes wieder.

color_get_value(col) Gibt die Value Komponente des Farbwertes wieder.

merge_color(col1,col2,amount) Gibt die Mischung aus col1 und col2 wieder. Die Mischung wird durch amount bestimmt. Ein Wert von 0 tendiert zu col1, ein Wert von 1 zu col2 und Werte dazwischen werden gemischt.

Manchmal willst du temporär etwas ändern, etwas zeichnen und dann zu den originalen Einstellungen zurückkehren (speziell wenn du etwas Allgemeines in Skripts machst). Dafür existieren folgende Funktionen:

push_graphics_settings() Speichert die aktuellen Einstellungen Intern. Du kannst die Funktion mehrmals aufrufen. (Wenn du sie zu oft speicherst ohne sie wiederherzustellen tritt eine Fehlermeldung auf. Dies ist normalerweise das Ergebniss eines Fehlers in deinem Code.)

pop_graphics_settings() Stellt die Einstellungen wieder her. Jeder pop Aufruf braucht vorher einen push Aufruf. Resultiert in einem Fehler wenn keine Einstellungen gespeichert sind.
Einige weitere Funktionen existieren:
string_width(string) Breite der Zeichenkette (string), wenn sie mit der aktuellen Schriftart und der "draw_text()" Funktion gezeichnet würde. Kann benutzt werden, für präzise Grafikpositionierung.

string_height(string) Das gleiche, wie oben, nur ist es hier die Höhe der Zeichenkette.

string_width_ext(string,sep,w) Breite des "string", wenn er mit dem aktuellen Font und der "draw_text_ext()" Funktion gezeichnet würde. Kann für präzise Grafikpositionierung verwendet werden.

string_height_ext(string,sep,w) Das gleiche wie oben, nur wird hier die Höhe des string angegeben.

screen_gamma(r,g,b) Setzt den Gamma-Korrekturwert. r,g,b liegen in den Grenzen -1 und 1. Voreingestellt ist 0. Wenn du einen Wert kleiner 0 verwendest, wird die Farbe dunkler. Wenn du einen Wert grösser 0 einsetzt, wird die Farbe aufgehellt. Die meiste Zeit über wirst du die drei Werte gleich halten. Um beispielsweise einen Blitzeffekt zu erzielen, kannst du die Werte kurzzeitig nahe an 1 setzen. Diese Funktion kann nur im Exklusiven Grafikmodus verwendet werden!

screen_save(fname) Speichert ein Abbild des Bildschirms im bmp-Format mit dem angegebenen Dateinamen (fname). Dies ist nützlich, wenn du Bildschirmschnappschüsse benötigst.

screen_save_part(fname,left,top,right,bottom) Speichert den angegebenen Teil des Bildschirms in der Datei fname. (Beachte das die rechte und untere Pixelzeile enthalten ist.)

33.6 Views (Ansichten)

Wie du wissen solltest, kannst du bis zu acht verschiede "views" definieren, wenn du einen Raum erstellst. Auf diese Weise kannst du verschiedene Teile eines Raumes an unterschiedlichen Bildschirmpositionen darstellen. Du kannst damit auch sicherstellen, dass ein Objekt immer gesehen wird. Du kannst "views" aus dem Programmcode heraus steuern. Du kannst "views" sichtbar und unsichtbar machen, sie an einer anderen Bildschirmposition darstellen und die Grösse ändern (was sehr nützlich ist, wenn du kein Objekt anzeigen willst), du kannst die Grösse des horizontalen und vertikalen Rahmens um ein sichtbares Objekt herum bestimmen und du kannst angebeben, welches Objekt in den Ansichten zu sehen ist. Letzteres ist besonders wichtig, wenn das wichtige Objekt während des Spiels wechselt. Beispielsweise willst du die Hauptfigur deines Spiels ändern, basierend auf seinem momentanen Status. Unglücklicherweise bedeutet dies, das es nicht mehr das Objekt ist, welches sichtbar bleiben soll. Dem kann abgeholfen werden, indem du eine Programmzeile Code im "creation-Event" aller möglichen Hauptobjekte setzt (angenommen es geschieht im ersten "view"):

```
{
view_object[0] = object_index;
}
```

Folgende Variablen beeinflussen die Ansicht (view). Alle, ausser den ersten Beiden, sind "arrays" (Datenfelder), welche von 0 (erster "view") bis 7 (letzter "view") reichen.

view_enabled Gibt an, ob "views" aktiviert sind oder nicht.

view_current* Der momentan gezeichnete "view" (0-7). Verwende dies nur im "draw event". Du kannst diese Variable prüfen, um bestimmte Objekte in nur einem "view" darzustellen. Variable kann nicht geändert werden.

view_visible[0..7] Gibt an, ob der angegebene "view" dargestellt wird.

view_left[0..7] Linke Position der Ansicht (view) im Raum.
view_top[0..7] Obere Position der Ansicht im Raum.

view_width[0..7] Breite der Ansicht in Pixel.

view_height[0..7] Höhe der Ansicht in Pixel.

view_x[0..7] X-Position der Ansicht auf dem Bildschirm.

view_y[0..7] Y-Position der Ansicht.

view_hborder[0..7] Grösse des horizontalen Rahmens um das sichtbare Objekt (in Pixel).

view_vborder[0..7] Grösse des vertikalen Rahmens (in Pixel).

view_hspeed[0..7] Maximale horizontale Verschiebungsgeschwindigkeit des "view".

view_vspeed[0..7] Maximale vertikale Verschiebungsgeschwindigkeit des "view".

view_object[0..7] Objekt dessen Instanz im "view" angezeigt wird. Wenn mehrere Instanzen vorhanden sind, folgt die Ansicht nur der ersten Instanz des Objektes.
Du kannst der Variablen auch eine Instanz-id zuweisen. In diesem Fall folgt die Ansicht der angegebenen Instanz.
Beachte, dass die Grösse des Bilds auf dem Bildschirm abhängt von den sichtbaren "views" zu beginn des Raumes. Wenn du "views" während des Spiels änderst, können sie nicht mehr in den Bildschirm "passen".

Die Bildschirmgrösse wird nicht automatisch angepasst. Wenn du also so etwas benutzt, musst du selbst dafür sorgen, dass es passt, indem du folgende Variablen verwendest:

screen_width Breite des Bilds auf dem Schirm, das ist der Bereich in dem gezeichnet wird. Wenn kein "view" definiert ist entspricht die Breite der Raumbreite (room_width).

screen_height Höhe des Bilds auf dem Bildschirm.

33.7 Transitions (Übergänge/Überblendungen/Raumwechsel)

Wie du weisst, kannst du, wenn du von Raum zu Raum wechselst, einen Übergang festlegen. Du kannst auch einen Übergang für das nächste "Frame" bestimmen, ohne den Raum zu wechseln, indem du die Variable "transition_kind" verwendest. Wenn du ihr einen Wert zwischen 1 und 17 zuweist, wird die korrespondierende "transition" verwendet (es sind die gleichen Übergänge, wie bei den Räumen). Ein Wert von 0 bedeutet keinen Übergangseffekt. Es hat nur einen Effekt, wenn das nächste "Frame" gezeichnet wird.
Du kannst diese Variablen auch setzen, bevor du den Raum vom Programmcode aus wechselst.

transition_kind Gibt den nächsten Bildübergang an (0-17).

transition_time Gibt die Zeit für die Überblendung an (in millisekunden).

transition_steps Anzahl der "steps" für den Übergang.

33.8 Repainting the screen (Neuzeichnen des Bildschirms)

Normalerweise wird der Raum am Ende jedes "step" neu auf den Schirm gezeichnet. Aber unter seltenen Umständen muss der Raum zu anderen Momenten neu dargestellt werden. Das geschieht, wenn dein Programm die Kontrolle übernimmt. Beispielsweise wenn das Spiel lange "geschlafen/pausiert" (sleep) hat, kann ein auffrischen des Bildschirms notwendig sein. Auch wenn du aus dem Programmcode heraus ein Nachrichtfenster einblendest, wo der Spieler eine Eingabe machen soll, brauchst du es. es gibt zwei unterschiedliche Funktionen dafür.

screen_redraw() zeichnet den Raum neu, indem es alle "draw-events" aufruft.

screen_refresh() zeichnet nur das aktuelle Abbild des Raumes neu (ohne die "draw-events" auszuführen).

Um die zweite Funktion zu verstehen, musst du wissen, wie intern (im GM) gezeichnet wird. Im Speicher wird ein Bild erstellt, auf dem alles gezeichnet wird. Dieses Bild wird nicht auf dem Schirm angezeigt. Nur am Ende eines "step", nachdem alles gezeichnet wurde, wird das Bildschirmbild durch dieses "Speicherbild" ersetzt. (Das nennt man "double buffering".) Die erste Funktion zeichnet das "Speicherbild" neu und aktualisiert dann das Bildschirmbild, die zweite Funktion ersetzt nur das Bildschirmbild durch das Speicherbild. Jetzt sollte dir auch einleuchten, warum du keine "draw"-Funktionen in anderen Ereignissen ausser "drawevents" verwenden kannst.

Sie zeichnen die Dinge auf dem Speicherbild aber sie werden nicht auf dem Bildschirmbild dargestellt. Und wenn die "draw-events" ausgeführt werden, wird zuerst das Hintergrundbild gezeichnet, dadurch wird alles was vorher auf das Speicherbild gezeichnet wurde überschrieben. Wenn du aber "screen_refresh()" nach deinem Zeichnen aufrufst, wird das aktualisierte Bild gezeichnet.
Beispielsweise schreibst du via script einen Text auf den Bildschirm, rufst die "refresh"-Funktion

auf und wartest auf die Eingabe des Spielers, so wie hier veranschaulicht:

```
{
draw_text(screen_width/2,100,'Press any key to continue.');
screen_refresh();
keyboard_wait();
}
```

Beachte bitte, wenn du in einem anderen Event als dem "draw"-Event zeichnest, zeichnest du einfach auf das Bild, nicht innerhalb eines "views"! Demnach sind die Koordinaten so, als ob kein "view" definiert ist. Sei behutsam beim Anwenden dieser Technik. Versichere dich, das du alles begriffen hast und beachte, dass das Auffrischen (refresh) auch etwas Zeit benötigt.

34. Soundeffekte und Musik

Geräusche spielen eine entscheidende Rolle in Computerspielen. Es gibt zwei Arten: Hintergrundmusik und Soundeffekte (SFX). Hintergrundmusik besteht normalerweise aus einer langen MIDI-Datei, die endlos wiederholt wird. SFX sind kleine/kurze WAV-Dateien. Um sofortige Effekte zuhaben, werden diese Stücke im PC-Speicher bereitgehalten. Mache sie also nicht arg so lang/gross. Geräusche kommen ins Spiel, in Form von "sound resources"(~Geräuschquelle). Prüfe immer, ob die Bezeichnungen, die du vergibst, auch gültige Variablenbezeichnungen sind. Einen Punkt gibt es bei SFX, die Anzahl der "buffer", wo man anfangs etwas probieren sollte. Das System kann immer nur eine WAV-Datei abspielen. Das bedeutet, wenn du einen sound wieder benutzt, bevor die vorherige WAV-Datei beendet ist, wird der vorherige Sound abgebrochen. Sowas ist nicht reizvoll. Wenn du also eine WAV-Datei x-mal gleichzeitig brauchst (Ein Schussgeräusch beispielsweise), benötigst du x "buffer"(~Pufferspeicher). Je höher die Anzahl der "buffer", desto öfter kann diese WAV-Datei abgespielt werden; aber um so höher ist auch der Speicherbedarf. Benutze es also mit Behutsamkeit. Game Maker wählt automatisch den ersten verfügbaren "buffer". Wenn du also einmal die Anzahl der "buffer festgelegt hast, brauchst du dich um nix mehr kümmern.

Es gibt fünf grundlegende Funktionen für Geräusche:
Zwei, um eine WAV-Datei abzuspielen, eine zum Prüfen, ob ein SFX abgespielt wird und zwei um sie anzuhalten. Die meisten nehmen den Index als Argument. Der Name des SFX stellt diesen Zeiger dar. Du kannst aber auch den "index"in einer Variablen speichern und damit arbeiten.

sound_play(index) Spielt den SFX(index entspricht Name)einmal.

sound_loop(index) Spielt den SFX(index entspricht Name)als Endlosschleife.

sound_stop(index) Stoppt den SFX(index entspricht Name),wenn mehrere "buffer"gespielt werden,werden alle angehalten.

sound_stop_all() Stoppt die gesamten SFX.

sound_isplaying(index) Gibt an,ob der angegebene (index)SFX gespielt wird.

Es ist möglich weitere Effekte zu benutzen. Dies geht aber nur mit WAV-Dateien - nicht mit MIDI-Dateien. Wenn du besondere Effekte benutzen willst, musst du die entsprechende "checkbox" in den "sound properties" ("Allow for sound effects") markieren. Beachte dabei, dass SFX mit aktivierten Effekten mehr Speicherbedarf haben. Mach da also nur einen Haken, wenn du untenstehende Funktionen benutzt! Es gibt drei Arten von Effekten.
Zuerst die Lautstärke.
Ein Wert von 0 bedeutet Stille. Ein Wert von 1 bedeutet, dass die WAV-Datei in originaler Lautstärke wiedergegeben wird -lauter geht nicht!
Zweitens das Panorama (akustische Position der Schallquelle).
Ein Wert von 0 bedeutet ganz links, einer von 1 ganz rechts und 0.5 markiert die Mitte (dies ist auch die Voreinstellung). Damit kannst du eine bewegte Schallquelle simulieren.
Drittens die Frequenz.
Dies kann benutzt werden, um beispielsweise Antriebe(Motorgeräusche) zu simulieren. Ein Wert von 0 ist die niedrigste Wiedergabefrequenz, einer von 1 die höchste Wiedergabefrequenz.

sound_volume(index,value) Ändert die Lautstärke von index (Name der WAV-Datei)(0 =leise,1 =laut).

sound_pan(index,value) Ändert das Panorama von index (0 =links,1 =rechts).

sound_frequency(index,value) Ändert die Wiedergabefrequenz von index (0 =langsamste,1 =schnellste).

SFX sind eine komplizierte Angelegenheit. MIDI-Dateien werden mit der Standard Multimedia-Anwendung abgespielt. Es kann nur eine einzige MIDI-Datei gleichzeitig abgespielt werden und irgendwelche Effekte werden nicht unterstützt.
Die WAV-Dateien verarbeitet der Game Maker mit DirectSound. Sie werden im Speicher bereitgehalten und können deshalb auch Effekte haben. Game Maker versucht auch andere Formate abzuspielen, wenn du sie beschreibst, insbesondere MP3-Files. Auch MP3s werden mit der Standard Multimedia-Anwendung wiedergegeben. Sei deshalb vorsichtig! Ob das funktioniert ist Systemabhängig oder manchmal auch davon, welche andere Software läuft bzw.installiert ist. Es ist nicht empfehlenswert MP3s zu verwenden, wenn du dein Spiel weitergeben möchtest.
Es gibt auch viele Funktionen zum Abspielen von Musik-CDs:

cd_init() Muss ausgeführt werden bevor die anderen Funktionen benutzt werden.Sollte auch nach einem Cd- Wechsel aufgerufen werden (oder einfach von Zeit zu Zeit).

cd_present() Gibt an, ob eine Cd im Laufwerk ist.

cd_number() Anzahl der Tracks auf der CD.

cd_playing() Gibt an, ob die CD momentan abgespielt wird.

cd_paused() Gibt an, ob die CD angehalten oder pausiert ist.

cd_track() Gibt die Nummer des momentan gespielten Track an (1=ist der erste).

cd_length() Gibt die totale Wiedergabedauer an in Millisekunden.

cd_track_length(n) Gibt die Wiedergabedauer von Lied n an in Millisekunden.

cd_position() Momentane Wiedergabeposition in Millisekunden bezogen auf gesamte CD.

cd_track_position() Momentane Wiedergabeposition in Millisekunden bezogen auf aktuelles Lied.

cd_play(first,last) Abspielreihenfolge vom ersten Lied bis zum Letzten. Wenn du die ganze CD spielen willst nimm 1 bis 1000 als Argumente.

cd_stop() Wiedergabe stoppen.

cd_pause() Pause.

cd_resume() Fortsetzen.

cd_set_position(pos) Setzen der Leseposition in Millisekunden bezogen auf die gesamte CD.

cd_set_trac _position(pos) Das gleiche, nur im momentanen Lied.

cd_open_door() Ausfahren der CD-Schublade.

cd_close_door() Schliessen der CD-Schublade.

Es gibt eine sehr allgemeine Funktion, um auf die Multimediamöglichkeiten von Windows zuzugreifen.

MCI_command(str) Diese Funktion sendet eine Befehlszeichenkette an das Windows Multimedia System mit Hilfe des Media Control Interface (MCI). Es liefert einen return-string. Du kannst es benutzten, um
verschiedene Multimediageräte anzusprechen. Schau in der Windows-Dokumentation nach, wie es verwendet wird.
Beispiel:

MCI_command('play cdaudio from 1') spielt eine CD an. (Wenn du vorher alles richtig mit weiteren Funktionen initialisiert hast.)

Diese Funktion ist nur was für Fortgeschrittene!

35. Splash-Screens, Highscores und andere Pop-ups

Viele Spiele verwenden sogenannte "splash-screens" (Einblendung). Auf diesen "screens" werden Videos, Bilder oder ein wenig Text dargestellt. Oft werden sie zu Beginn eines Spieles oder Levels als Intro - oder aber am Ende als "Credits" eingesetzt. Im Game Maker können diese "splash-screens", mit Text, Bildern oder Videos, jederzeit während des Spiels gezeigt werden. Das Spiel wird so lange angehalten, wie der "splash-screen" angezeigt wird. Diese Funktionen werden dafür benutzt:

show_text(fname,full,backcol,delay) Zeigt einen "text splash screen". "fname" ist der Name der Textdatei (.txt oder .rtf). Diese Datei muss im selben Ordner/Verzeichnis sein, wie das Spiel auch. Auch wenn du eine "stand-alone"-Version deines Spiels machst, darfst du nicht versäumen, diese Dateien mitzugeben. "full" zeigt an, ob es im Vollbildmodus dargestellt wird. "backcol" ist die "background color" (Hintergrundfarbe) und "delay" gibt die zu verstreichende Zeitspanne an, bevor es wieder zurück zum Spiel geht. (Der Spieler kann immer mit der Maus innerhalb des "screens" klicken, um zum Spiel zurückzukehren.)

show_image(fname,full,delay) Zeigt einen "image splash screen". "fname" ist der Name der Bilddatei (nur:.bmp,.jpg und .wmf Dateien). Du musst diese Dateien im Spielverzeichnis bereithalten. "full" zeigt an, ob es im Vollbildmodus dargestellt wird. "delay" Anzeigedauer bis das Spiel fortgesetzt wird.

show_video(fname,full,loop) Zeigt einen "video splash screen". "fname" Ist der Name der Video-Datei (.avi,.mpg). Sie muss im Spielverzeichnis enthalten sein. "full" zeigt an, ob es im Vollbildmodus angezeigt wird. "loop" gibt an, dass es wiederholt werden soll.

show_info() Zeigt die "game information"an. load_info(fname) Lade "game information" aus der Datei: fname. Es sollte eine .rtf-Datei sein. Dadurch ist es möglich, verschiedene Hilfe-Dateien, zu unterschiedlichen Situationen darzustellen. Du kannst auch die "data file resource" nutzen, um diese Dateien einzubinden.

Eine Anzahl von Funktionen gibt es, um Nachrichten, Fragen und Auswahlmenüs einzublenden oder Dialoge, wo der Spieler Zahlen, Zeichen, seine Farbauswahl oder einen Dateinamen angibt:

show_message(str) Zeigt eine "dialog box" mit der Zeichenkette str als Meldungstext.

show_message_ext(str,but1,but2,but3) Zeigt eine "dialog box" mit der Zeichenkette str als Meldungstext und bis zu drei "buttons" (Auswahlschalter), wobei but1, but2 und but3 den "button text" enthalten. Ein leerer string bedeutet, dass kein "button"angezeigt wird. Im Text kannst du das "&"-Symbol dazu benutzen, um darzustellen, dass das nächste Zeichen als Tastaturkürzel für diesen "button" verwendet wird. Diese Funktion gibt den Wert des Zeichen wieder (0 wenn Esc-Taste gedrückt wird).

show_question(str) Zeigt eine Frage an; gibt "true" zurück, wenn der Benutzer "yes" wählt, sonst "false".

get_integer(str,def) Fragt den Spieler, in einer dialog-box, nach einer ganzen Zahl. str ist der Fragetext. def ist die voreingestellte, anzuzeigende Zahl.

get_string(str,def) Fragt den Spieler per Dialog-box nach einem string (Zeichenkette). str ist der Nachrichtentext. def ist der voreingestellte, anzuzeigende Wert.

message_background(back) Legt das Hintergrundbild für die "pop-up box" der obengenannten Funktion fest. back muss ein im Spiel definierter "background" sein (er muss im resource-tree vorhanden sein).

message_button(spr) Setzt ein "sprite" anstelle des "buttons" in die "pop-up box". spr muss aus drei (3) Einzelbildern bestehen, das erste stellt den unbetätigten "button dar, das zweite, wie der "button"aussieht, wenn der Mauszeiger darüber ist und das dritte, wenn der "button"gedrückt wird.

message_text_font(name,size,color,style) Legt den "font"(Schriftart) für das "pop-up"-Fenster fest.

message_button_font(name,size,color,style) "font" für die Schaltflächen der "pop-up box".

message_input_font(name,size,color,style) Schriftart für das Eingabefeld.

message_mouse_color(col) Schriftfarbe der "buttons", wenn Mauszeiger darüber ist.

message_input_color(col) Hintergrundfarbe des Eingabefeldes der "pop-up box".

message_caption(show,str) Titelzeile der "pop-up box". "show" zeigt an, ob ein Rahmen gezeigt wird (0 nein bzw.1 ja) und str stellt die Titelzeile dar.

message_position(x,y) Bildschirmposition der "pop-up box".

message_size(w,h) Legt die Fenstergrösse des "pop-up"-Fensters fest. Wenn du 0 für die Breite w einsetzt, wird die Originalbreite der Bilddatei benutzt. Wenn du 0 für die Höhe h verwendest, wird die Höhe aus der Anzahl der darzustellenden Textzeilen berechnet.

show_menu(str,def) Zeigt ein "popup menu". str enthält den "menu text". Dieses besteht aus verschiedenen Menüteilen, die durch einen Querbalken getrennt sind. Zum Beispiel:
str ='menu0|menu1|menu2'.
Wenn nun das erste ausgewählt wird, wird eine 0 zurückgeggeben, usw. Wenn der Spieler keinen Punkt wählt, wird der vordefinierte Wert def zurückgegeben.

show_menu_pos(x,y,str,def) zeigt ein menu (wie oben erläutert) an der Bildschirmposition x,y.

get_color(defcol) Fragt den Spieler nach einer Farbe. "defcol" ist die voreingestellte color(Farbe). Wenn der Benutzer "Cancel" (Abbrechen) drückt, wird der Wert -1 zurückgegeben.

get_open_filename(filter,fname) Fragt den Spieler nach einer zu öffnenden Datei mit angebenem Filter. Der Filter hat die Form "name1|mask1|name2|mask2|...". Eine "mask" (Maske) enthält verschiedene Optionen, getrennt durch ein Semikolon. "*" steht für irgendeinen string. Zum Beispiel:
bitmaps|*.bmp;*.wmf
Wenn der Nutzer "Cancel" betätigt, wird ein inhaltsloser string zurückgegeben.

get_save_filename(filter,fname) Fragt nach dem Namen, unter dem die Datei mit angegeben Filter gespeichert werden soll. Wenn der Benuzter "Cancel" drückt, wird ein leerer string wiedergegeben.

get_directory(dname) Fragt nach dem Verzeichnisnamen. "dname" ist der voreingestellte Name. Wenn der
Benutzer abbricht, wird ein leerer string zurückgegeben.

get_directory_alt(capt,root) Eine Alternativmöglichkeit nach dem Verzeichnis zu fragen. "capt" ist

der anzuzeigende Titelleistentext. "root" ist der Ursprung des anzuzeigenden Verzeichnisbaumes. Benutze einen leeren string, damit du den kompletten Baum anzeigen kannst. Wenn der Benutzer "Cancel" drückt, wird ein leerer string wiedergegeben.

show_error(str,abort) Stellt eine Standardfehlermeldung dar (oder schreibt ins log-file). "abort" zeigt an, ob das Spiel abgebrochen werden soll.

Eine besondere Highscoreliste wird für jedes Spiel bereitgehalten. Folgende Funktionen gibt es dafür:

highscore_show(numb) Zeigt die Highscoreliste. "numb" entspricht dem neuen "score". Wenn der Punktestand gut genug ist, um in die Tabelle aufgenommen zu werden, kann der Spieler seinen Namen eingeben. Verwende -1, um die aktuelle Liste nur anzuzeigen.

highscore_show_ext(numb,back,border,col1,col2,name,size) stellt die Highscortabelle dar. "numb" ist der neue Punktestand. Wenn der Punktestand gut genug ist, um in die Tabelle aufgenommen zu werden, kann der Spieler seinen Namen eingeben. Verwende -1,um die aktuelle Liste nur anzuzeigen. "back" entspricht dem
zu nutzenden Hintergrundbild, "border" gibt an, ob ein Rahmen gezeichnet wird. "col1" entspricht der Farbe des neuen Eintrags, "col2" der Farbe der übrigen Einträge. "name" ist der Name des zu benutzenden "fonts" (Schriftart), und "size" die Schriftgrösse.

highscore_clear() Löscht alle Einträge der Highscoretabelle!

highscore_add(str,numb) Fügt einen Spielernamen nebst Punktestand der Tabelle hinzu (str:Spielername und numb:Punktestand).

highscore_add_current() Fügt den aktuellen "score" zur Tabelle. Der Spieler wird nach einem Namen gefragt.

highscore_value(place) Gibt den Punktestand vom Inhaber des mit "place" markierten Tabellenplatzes an (1- 10). Damit kannst du deine eigene Tabelle machen.

highscore_name(place) Gibt den Namen von Tabellenplatz "place" an (1-10).

draw_highscore(x1,y1,x2,y2) Zeichnet die Highscoretabelle im "room" in der angegebenen Dimension mit dem momentan gewählten "font" (Schriftart).

Beachte, dass keines dieser "pop-ups" dargestellt werden kann, wenn das Spiel im exklusiven Grafikmodus läuft!!

36. Ressourcen

Im Game Maker kannst du verschiedene Arten von Komponenten definieren, wie zum Beispiel Sprites (Bildfolgen), Geräusche, Dateien, Objekte, etc.. In diesem Kapitel findest du eine Anzahl von Funktionen, die sich mit den Komponenten befassen. Bevor du sie benutzt, vergewissere dich das du folgendes begriffen hast: Wann immer du eine Spielkomponente änderst, ist das ursprüngliche Datenmaterial verschwunden! Das heisst, diese Komponente ändert sich für alle Instanzen, von denen sie verwendet wird. Beispiel: Wenn du ein Sprite änderst, veränderst du es bei allen Instanzen, welche dieses "sprite" verwenden, für den Rest des Spiels. Neustarten eines Raumes oder Neustarten des Spiels bringt eine Spielkomponente nicht wieder in die ursprüngliche Form! Auch wenn der momentane Spielstand gespeichert wird, werden veränderte Komponenten NICHT gespeichert. Wenn du einen Spielstand einliest, obliegt es deinem Verantwortungsbereich, dafür Sorge zu tragen, dass alle Komponeneten entsprechend wieder vorhanden sind.

36.1 Sprites

Folgende Funktionen liefern Informationen über ein Sprite:

sprite_exists(ind) Gibt an, ob ein Sprite mit angegebenem Index vorhanden ist.

sprite_get_name(ind) Gibt den Namen des Sprite an, dessen Index angegeben ist.

sprite_get_number(ind) Gibt die Anzahl der Einzelbilder für das indizierte Sprite.

sprite_get_width(ind) Gibt die Breite des indizierten Sprite an.

sprite_get_height(ind) Gibt die Höhe des indizierten Sprite an.

sprite_get_transparent(ind) Gibt an, ob das indizierte Sprite transparent ist.

sprite_get_xoffset(ind) Gibt den X-Versatz des indizierten Sprite an.

sprite_get_yoffset(ind) Gibt den Y-Versatz des indizierten Sprite an.

sprite_get_bbox_left(ind) Gibt die linke Seite des Begrenzungsrahmen des indizierten Sprite an.

sprite_get_bbox_right(ind) Gibt die rechte Seite des Begrenzungsrahmen des indizierten Sprite an.

sprite_get_bbox_top(ind) Gibt die obere Seite des Begrenzungsrahmen des indizierten Sprite an.

sprite_get_bbox_bottom(ind) Gibt die untere Seite des Begrenzungsrahmen des indizierten Sprite an.

sprite_get_precise(ind) Gibt an, ob das indizierte Sprite präzise Kollisionserkennung verwendet.

sprite_get_videomem(ind) Gibt an, ob das inditierte Sprite VIDEO-Speicher benötigt.

sprite_get_loadonuse(ind) Gibt an, ob das indizierte Sprite nur bei Gebrauch geladen wird.

Sprites belegen eine Menge Speicherplatz. Um sie schnell darstellen zu können, ist es wichtig sie im VIDEO-Speicher zu speichern. Wie im Kapitel 15 gezeigt, kannst du festlegen, welche Sprites im VIDEO-Speicher gespeichert werden sollen. Auch kannst du angeben, dass bestimmte Sprites nur geladen werden, wenn sie benötigt werden. Diese Sprites werden am Ende eines Levels aussortiert. Du kannst diesen Vorgang teilweise aus dem Programmcode heraus steuern.

Folgende Funktionen gibt es:

sprite_discard(numb) Gibt den Speicherbereich, der von dem Sprite verwendet wurde, wieder frei.

Wenn bei dem Sprite die load-on-use (Nur bei Bedarf laden)
Eigenschaft eingeschaltet ist, wird es ganz entfernt. Ansonsten wird eine Kopie im normalen Speichen bereitgehalten (wovon normalerweise ausreichend vorhanden ist), so dass das Sprite wiederhergestellt werden kann, wenn es benötigt wird.

sprite_restore(numb) Bringt ein Sprite in den (VIDEO-)Speicher. Das geschieht eigentlich automatisch, wenn das Sprite beötigt wird. Weil es aber manchmal einen
kleinen Rucker verursacht, besonders wenn load-on-use gesetzt und das Sprite gross ist. Möglicherweise möchtest du es beschleunigen, indem du es zu Beginn des Raumes lädst, wo es benötigt wird.

discard_all() Sortiert alle Sprites, Bildschirmhintergründe und Sounds aus, die load-on-use gesetzt haben.

36.2 Sound

Folgende Funktionen geben dir Auskunft über einen Sound:

sound_exists(ind) Gibt an, ob der indizierte Sound vorhanden ist.

sound_get_name(ind) Gibt den Namen des indizierten Sound an.

sound_get_kind(ind) Gibt die Art des indizierten Sound an (0=wave, 1=midi, 2=mp3, 10=unbekannt).

sound_get_buffers(ind) Gibt die Anzahl der buffers (Pufferspeicher) des indizierten Sound an.

sound_get_effect(ind) Gibt an, ob der indizierte Sound Spezialeffekte erlaubt.

sound_get_loadonuse(ind) Gibt an, ob der indizierte Sound nur bei Bedarf geladen wird.

Sounds brauchen eine Menge Reserven und die meisten Systeme können nur eine begrenzte Anzahl von Geräuschen wiedergeben und verarbeiten. Wenn du ein grosses Spielprojekt machst, möchtest du wahrscheinlich mehr Einfluss darauf haben, welcher Sound jetzt in den Speicher geladen werden soll und wie oft. Du kannst dafür die load-on-use-Option verwenden, um sicherzustellen, dass der Sound nur bei Bedarf geladen wird. Das kann auch wieder zu einer kleinen Verzögerung führen, wenn der Sound das erste mal verwendet wird. Es hilft auch nicht viel, wenn du nur einen grossen Raum hast. Für zusätzliche Kontrolle verwende folgende
Funktionen:

sound_discard(index) Gibt den vom indizierten Sound belegten Speicher frei.

sound_restore(index) Stellt den indizierten Sound im Speicher wieder her.

discard_all() Sortiert alles aus, was nur bei Bedarf geladen werden soll (Sprites, Bildschirmhintergründe und Sounds).

36.3 Hintergründe

Untenstehende Funktionen informieren über den Hintergrund:

background_exists(ind) Gibt an, ob der indizierte Hintergrund vorhanden ist.

background_get_name(ind) Liefert den Namen des indizierten Hintergrund.

background_get_width(ind) Breite des indizierten Hintergrund.

background_get_height(ind) Höhe des indizierten Hintergrund.

background_get_transparent(ind) Gibt an, ob der indizierte Hintergrund transparent ist.

background_get_videomem(ind) Gibt an, ob VIDEO-Speicher vom indizierten Hintergrund verwendet werden soll.

background_get_loadonuse(ind) Gibt an, ob der indizierte Hintergrund bei Bedarf geladen werden soll.

Hintergrundbilder belegen eine Menge Speicher. Um sie schnell genug darstellen zu können, kann es hilfreich sein, sie im VIDEO-Speicher zu haben. Wie vorhin schon erwähnt kannst du angeben, welcher Hintergrund sich im VIDEO-Speicher befinden darf und ob er nur bei Bedarf geladen wird. Solche Hintergründe werden am Levelende wieder aussortiert. Du kannst es teilweise via Programmcode lenken.
Diese Funktionen gibt es:

background_discard(numb) Gibt den vom Hintergrundbild belegten Speicher frei. Wenn das Hintergrundbild nur bei Bedarf geladen werden sol, wird es ganz entfernt. Ansonsten ist im normalen Speicher eine Kopie vorhanden, damit man es wiederherstellen kann, wenn man es benötigt.

background_restore(numb) Stellt das Hintergrundbild im Speicher wieder her. Normalerweise geschieht das automatisch, wenn das Bild benötigt wird. Auch hier
kanns ruckeln, wenn es ein fettes Bild ist. Lade es dann zu Beginn des Raumes, wo es verwendet werden soll.

discard_all() Sortiert alle Sprites, Hintergründe und Sounds aus, die bei Bedarf geladen werden sollen.

36.4 Pfade

Folgende Funktionen geben Informationen über einen Pfad:

path_exists(ind) Gibt an, ob ein Pfad mit dem angegebenen Index existiert.

path_get_name(ind) Liefert den Namen des indizierten Pfades.

path_get_length(ind) Länge des indizierten Pfades.

path_get_kind(ind) Punktverbindungsart des indizierten Pfades (0=direkt, 1=sanft).

path_get_closed(ind) Gibt an ob der Pfad geschlossen ist oder nicht.

path_get_precision(ind) Gibt die Präzision für die Erzeugung flüssiger paths wieder.

path_get_number(ind) Gibt die Anzahl der Definitionspunkte des Pfades wieder.

path_get_point_x(ind,n) Gibt die x-Koordinate des n'ten Definitionspunktes des Pfades wieder. 0 ist der erste Punkt.

path_get_point_y(ind,n) Gibt die y-Koordinate des n'ten Definitionspunktes des Pfades wieder. 0 ist der erste Punkt.

path_get_point_speed(ind,n) Gibt den Geschwindigkeitsfaktor am n'ten Definitionspunkt des Pfades wieder. 0 ist der erste Punkt.

path_get_x(ind,pos) Gibt die x-Koordinate der Position pos des Pfades wieder. pos muss zwischen 0 und 1 liegen.

path_get_y(ind,pos) Gibt die x-Koordinate der Position pos des Pfades wieder. pos muss zwischen 0 und 1 liegen.

path_get_speed(ind,pos) Gibt den Geschwindigkeitsfaktor der Position pos des Pfades wieder. pos muss zwischen 0 und 1 liegen.

36.5 Skripte

Nachstehende Funktionen liefern Infos über Skripte:

script_exists(ind) Gibt an, ob das indizierte Skript existiert.

script_get_name(ind) Gibt den Namen des indizierten Skript an.

script_get_text(ind) Gibt den Inhalt des indizierten Skript wieder.

36.6 Daten-Dateien

Hier die nötigen Funktionen:

datafile_exists(ind) Gibt an, ob indiziertes "data file" existiert.

datafile_get_name(ind) Gibt den Namen des indizierten "data file" an.

datafile_get_filename(ind) Gibt den Dateinamen des indizierten "data file" wieder.

Folgende Funktionen können verwendet werden, wenn man nicht zu Spielbeginn alle Daten-Dateien automatisch exportiert:

datafile_export(ind,fname) Exportiert die Daten-Datei in die angegebene Datei (wenn du es nicht als Voreinstellung von Beginn an machst).

datafile_discard(ind) Gibt alle intern gespeicherten Daten der Daten-Datei frei.

36.7 Zeitleisten

Die folgenden Funktionen gibt die Informatione über Zeitleisten:

timeline_exists(ind) Gibt zurück ob die Zeitleiste mit dem Index existiert.

timeline_get_name(ind) Gibt den Namen der Zeitleiste mit dem angegeben Index zurück.

36.8 Objekte

Information über ein Objekt:

object_exists(ind) Gibt an, ob das indizierte Objekt vorhanden ist.

object_get_name(ind) Gibt den Namen des indizierten Objektes an.

object_get_sprite(ind) Liefert den Index des vorbestimmten Sprite des indizierten Objektes.

object_get_solid(ind) Gibt an, ob das indizierte Objekt solide/massiv ist (vorher festgelegt).

object_get_visible(ind) Gibt an, ob das indizierte Objekt per Voreinstellung sichtbar ist.

object_get_depth(ind) Liefert die Zeichenebene des indizierten Objektes.

object_get_persistent(ind) Gibt an, ob das indizierte Objekt beständig (persistent) ist.

object_get_mask(ind) Liefert den Index der Mask des indizierten Objektes (-1 wenn kein besondere angegeben).

object_get_parent(ind) Liefert den Index des Parent-Objektes vom indizierten Objekt (-1 wenn Elternlos).

object_is_ancestor(ind1,ind2) Gibt an, ob Objekt ind2 ein Vorfahre von Objekt ind1 ist.

36.9 Räume

Informationen über einen Raum:

room_exists(ind) Gibt an, ob der indizierte Raum vorhanden ist.

room_get_name(ind) Liefert den Namen des indizierten Raumes.

Beachte! Weil Räume sich während des Spiels verändern können, gibt es andere Funktionen um den Inhalt von Räumen zu verändern.

37. Ressourcen verändern

Diese Funktionen sind nur in der registrierten Version vom Game Maker verfügbar!

Es ist möglich neue Ressourcen während des Spieles zu erstellen. Ausserdem kann man bestehende Ressourcen verändern. Dieses Kapitel beschreibt diese Möglichkeiten.
Sei gewarnt! Das Verändern von Ressourcen führt leicht zu schweren Fehlern in deinem Spiel!!!
Du musst folgende Regeln befolgen:

· Verändere keine Ressourcen die gerade benutzt werden. Dies führt zu Fehlern! Z. b. verändere keine Sprites welche gerade von einer Instanz benutzt werden.
· Wenn du das Spiel während des Spiels speicherst, werden neue und geändertem Ressourcen NICHT mit diesem Spiel gespeichert. Also, wenn du das gespeicherte Spiel später wieder lädst, sind sie nicht mehr da. Generell, wenn du Ressourcen veränderst, solltest du nicht länger das eingebaute System für das Laden und Speichern von Spielen benutzen.
· Wenn du das Spiel während des Spielens neu startest, werden die geänderten Ressourcen nicht in ihrer originalen Form wiederhergestellt. Generell, wenn du Ressourcen veränderst, solltest du nicht mehr die Aktion oder Funktion für das Neu Starten (Restart) des Spieles verwenden.
· Verändern der Ressourcen macht alles langsamer. Verändern des Sprites oder Hintergründe ist sehr langsam. Also benutze es nicht während des Spielablaufes.
· Das Erstellen von Ressourcen während des Spieles (hauptsächlich Sprites und Hintergründe) benötigt einen Menge Speicher. Also gehe sorgfältig damit um. Wenn du z. B. ein 32 Frame 128x128 animiertes Sprite hast und du beschliesst 36 drehende Kopien davon zu machen, benutzt du 36x32x128x128x4 = 36 MB des Speichers!
· Gehe sicher dass du die Ressourcen löscht, wenn sie nicht mehr gebraucht werden. Sonst geht dem System der Speicher aus.

Generell solltest du keine Ressourcen während des Spielablaufes ändern. Besser erstelle und verändere Ressourcen am Beginn des Spieles oder vielleicht am Beginn eines Raumes.

37.1 Sprites

Die folgenden Routinen sind für das Manipulieren von Sprite-Eigenschaften vorhanden.

sprite_set_transparent(ind,transp) Setzt die Transparenz (true oder false) des Sprites mit dem angegebenen Index.

sprite_set_offset(ind,xoff,yoff) Setzt das Offset des Sprites mit dem angegebenen Index.

sprite_set_bbox_mode(ind,mode) Setzt den Begrenzungsrahmen des Sprites (0=automatisch, 1=gesamtes Bild, 2=manuell).

sprite_set_bbox(ind,left,top,right,bottom) Setzt den Begrenzungs-rahmen des Sprites mit dem angegebenen Index.

sprite_set_precise(ind,mode) Soll das Sprite mit dem angegebenen Index präzise Kollisionserkennung benützen (wahr oder falsch).

sprite_set_videomem(ind,mode) Soll das Sprite mit dem angegebenen Index den Videospeicher benutzen (true oder false).

sprite_set_loadonuse(ind,mode) Soll das Sprite mit dem angegebenen Index nur beim Starten geladen werden oder nicht (true oder false)

Die folgenden Routinen können für das Erstellen oder Entfernen neuer Sprites verwendet werden.

sprite_duplicate(ind) Erstellt ein Duplikat des Sprites mit dem angegebenen Index. Es gibt den Index des neuen Sprites zurück. Wenn ein Fehler passiert, wird –1 zurückgegeben.

sprite_merge(ind1,ind2) Vermischt die Bilder vom Sprite ind2 in den Sprite ind1, fügt ihm am Ende hinzu. Wenn die Grösse nicht mit dem Sprite übereinstimmt, wird dieser verändert.

sprite_add(fname,imgnumb,precise,transparent,videomem,loadonuse,xorig,yorig) Fügt ein Bild welches in der Datei fname gespeichert ist zu den Sprite-Ressourcen hinzu. Nur bmp, jpg und gif Bilder können damit umgeben. Wenn das Bild eine bmp oder jpg Bild ist kann es ein Strip mit einigen Einzelbilden für das Sprite enthalten. Benutze imgnumb um die Anzahl (1 für ein Einzelbild) anzugeben. Für (animierte) gif Bilder, wird dieses Argument nicht benötigt, die Anzahl der Bilder in einer Gif-Datei wird benützt. precise gibt an ob die präzise Kollisionserkennung eingeschaltet werden soll. transparent gibt an ob das Bild teilweise transparent sein soll, videomem gibt an ob das Sprite im Grafikspeicher gespeichert werden soll. xorig und yorig geben die Position des Bezugspunkte des Sprites an. Diese Funktion gibt den Index des neuen Sprites zurück, welcher zum Zeichnen verwendet werden kann, oder für das Zuteilen des Sprites zu einem Objekt. –1 wird bei einem Fehler zurückgegeben.

sprite_replace(ind,fname,imgnumb,precise,transparent,videomem,loadonuse,xorig,yorig) Das gleiche wie vorher aber in diesem Fall wird der Spriteindex ind ersetzt. Diese Funktion gibt zurück ob es erfolgreich war.

sprite_create_from_screen(left,top,right,bottom,precise,transparent,videomem,loadonuse,xorig,yorig) Erstellt ein Sprite durch Kopieren des angegebenen Feldes des Bildschirms. Das macht es möglich irgendein gewünschtes Sprite zu erstellen. Zeichne das Bild am Bildschirm mit den Zeitenfunktionen und danach erstelle ein Sprite davon. Die anderen Parameter sind wie die oben. Die Funktion gibt den Index des neuen Sprites zurück.

sprite_add_from_screen(ind,left,top,right,bottom) Fügt ein Feld am Bildschirm als nächstes Einzelbild eine Sprites mit dem index ind.

sprite_delete(ind) Löscht das Sprites aus dem Speicher, gibt den Speicher frei (er kann nicht wieder hergestellt werden.)

Die folgenden Routinen können benützt werden um ein Sprite zu manipulieren genauso wie die Befehle im Sprite-Editor. Diese Routinen sind nicht sehr schnell. Du kannst sie schneller machen indem du Precise collision checking der Sprites ausschaltetest und Load on use einschaltest und die bounding box auf manuell setzt. Wenn die manipulierten Sprites fertig sind, kannst du die Eigenschaften wieder auf die gewünschten Werte setzten.

sprite_mirror(ind) Spiegelt die Bilder des Sprites horizontal.

sprite_flip(ind) Dreht die Bilder des Sprites vertikal.

sprite_shift(ind,x,y) Verschiebt die Bilder des Sprites über den angegebenen Vektor.

sprite_rotate180(ind) Dreht die Bilder des Sprites um 180 Grad.

sprite_rotate90(ind,clockwise,resize) Dreht die Bilder des Sprites 90 Grad, clockwise gibt an ob im Uhrzeigesinn gedreht werden soll, resize gibt an ob das Bild vergrössert werden soll (dass die Breite und Höhe umgedreht wird).

sprite_rotate(ind,angle,quality) Dreht die Bilder des Sprites mit dem Winkel (angle), quality gibt die Qualität des Bildes (1-9) an. Je höher der Wert, umso besser die Qualität aber desto länger dauert es.

sprite_resize(ind,w,h,corner) Vergrössert die Bilder des Sprites mit w x h Pixel. Das Bild wird nicht vergrössert. corner gibt den Platz des originalen Bildes an (1-9 wie am Ziffernblock).

sprite_stretch(ind,w,h,quality) Dehnt die Bilder des Sprites mit der Grösse w x h und der angegebenen Qualität (1-9).

sprite_scale(ind,xscale,yscale,quality,corner,resize) Verändert den Umfang der Bilder des Sprites mit dem Factor xscale und yscale. quality gibt die Qualität an(1-9). resize gibt an ob das Bild vergrössert werden muss. corner gibt die Position des Originalbildes (1-9 wie am Ziffernblock) an, wenn die Bilder nicht vergrössert werden.

sprite_black_white(ind) Verändert die Bilder des Sprites in Schwarz/Weiß.

sprite_set_hue(ind,amount) Setzt den Farbton (0-255) für die Bilder des Sprites.

sprite_set_hue_partial(ind,from,till,newhue) Alle Pixel mit einem Farbton zwischen from und till werden mit dem neuen newhue ersetzt.

sprite_shift_hue(ind,amount) Versetzt den Farbton für das Sprite um den angegebenen Wert.

sprite_change_value(ind,amount) Ändert den Farbwert (0-255) der Bilder des Sprites.

sprite_change_saturation(ind,amount) Ändert die Farbsättigung (0-255) der Bilder des Sprites.

sprite_fade(ind,col,amount) Blendet die Bilder des Sprites in Richtung col (Farbe) mit der Anzahl amount (0-255).

sprite_screendoor(ind,amount) Gibt ein Anzahl amount (0-255) der Türtransparenz der Bilder des Sprites an..

sprite_blur(ind,amount) Verwischt die Bilder im Sprites mit der angegeben Menge (1-9).

37.2 Sounds

Die folgenden Routinen können benützt werden für das Erstellen neuer Sounds und das Zerstören eben jener.

sound_add(fname,buffers,effects,loadonuse) Fügt eine Soundressource dem Spiel hinzu. fname ist der Name der Sounddatei. buffers gibt die Menge des Buffers an,
und effects und loadonuse gibt an ob Soundeffekte erlaubt sind und ob der Sound im internen Speicher gespeichert (wahr oder falsch) werden soll. Die Funktion gibt den Index des neuen Sounds zurück, was für das Spielen eines Sounds benützt werden kann (- 1 wenn ein Fehler passiert, z. B. wenn die Datei nicht vorhanden ist).

sound_replace(index,fname,buffers,effects,loadonuse) Gleiche wie die vorherige Funktion aber diesmal wird kein neuer Sound erstellt sondern ein vorhandener
Soundindex wird ersetzt. Der alte Sound wird freigegeben. Gibt wieder zurück ob es funktioniert hat.

sound_delete(index) Löscht den angegebenen Sound, gibt den benutzten Speicher frei. Das kann nicht mehr rückgängig gemacht werden.

37.3 Hintergründe

Die folgenden Routinen sind für das Manipulieren von Hintergründen vorhanden:

background_set_transparent(ind,transp) Setzt die Transparenzfarbe (true oder false) des Hintergrundes mit dem angegebenen Index.

background_set_videomem(ind,mode) Soll der Hintergrund mit dem angegebenen Index den Videospeicher benützen (true oder false).

background_set_loadonuse(ind,mode) Soll der Hintergrund mit dem angegebenen Index nur beim Laden benützt werden (true oder false).

Die folgenden Routinen können zum Erstellen neuer Hintergründe oder zum Zerstören dieser benutzt werden.
background_duplicate(ind) Erstellt eine Duplikat des Hintergrundes mit dem angegebenen Index. Es gibt den Index des neuen Hintergrundes zurück. Wenn ein Fehler passiert, gibt es – 1 zurück.

background_create_color(w,h,col,transparent,videomem,loadonuse) Erstellt einen Hintergrund mit der angegebenen Grösse und Farbe und mit den gegebenen Einstellungen. Es gibt den Index des neuen Hintergrunds zurück. Wenn ein Fehler passiert wird -1 zurückgegeben.

background_create_gradient(w,h,col1,col2,kind,transparent,videomem,loadonuse) Erstellt einen Hintergrund mit der angegebenen Grösse und Farbverlauf und mit den gegebenen Einstellungen. col1 und col2 geben die Start- und Endfarbe an. kind ist eine Zahl zwischen 0 und 5 welche die Art des Verlaufs angibt: 0=horizontal 1=vertikal, 2= rechteckig, 3=ellipse, 4=doppelt horizontal, 5=doppelt vertical. Es gibt den Index des neuen Hintergrunds zurück. Wenn ein Fehler passiert wird -1 zurückgegeben.

background_add(fname,transparent,videomem,loadonuse) Fügt das Bild (in fname gespeichert) den Hingergrund –Ressourcen hinzu. Nur bmp und jpg Bilder können damit umgehen. transparent gibt an ob das Bild teilweise transparent ist, videomem gibt an ob der Hintergrund im Videospeicher geladen werden soll und loadonuse gibt an ob der Hintergrund nur geladen werden soll, wenn er benützt wird. Diese Funktion gibt den Index des neuen Hintergrundes zurück, du kannst dies zum Zeichnen benützen oder Verbinden mit der Variable background_index[0] um es sichtbar im aktuellen Raum zu machen. –1 wird bei einem Fehler zurückgegeben.

background_replace(ind,fname,transparent,videomem,loadonuse) Gleiche wie vorher aber in diesem Fall wird der Hintergrund mit Index überschrieben.
Die Funktion gibt zurück ob sie erfolgreich war. Wenn der Hintergrund zurzeit sichtbar im Raum ist, wird er auch ersetzt.

background_create_from_screen(left,top,right,bottom,transparent,videomem,loadonuse)
Erstellt einen Hintergrund durch Kopieren des angegebenen Feldes des Bildschirms. Das macht es möglich jeden gewünschten Hintergrund zu erstellen. Zeichne das Bild am Bildschirm mit den Zeichenfunktion und erstelle einen Hintergrund davon (Wenn du nicht den drawing Event benützt, kannst du es so machen, dass es nicht sichtbar ist dadurch dass der Bildschirm nicht neu aufgebaut wird). Die anderen Parameter sind wie oben. Die Funktion gibt den Index des neuen Hintergrundes zurück.

background_delete(ind) Löscht den Hintergrund aus dem Speicher, gibt den benutzen Speicher frei (Das kann nicht rückgängig gemacht werden).

Die folgenden Routinen können für das Manipulieren des Hintergrundes genauso wie die Befehle im Hintergrund-Editor verwendet werden.

background_mirror(ind) Spiegelt das Bild im Hintergrund waagrecht.

background_flip(ind) Dreht das Bild im Hintergrund senkrecht.

background_shift(ind,x,y) Verschiebt das Bild im Hintergrund über dem angegebenen Vektor.

background_rotate180(ind) Dreht das Bild im Hintergrund 180 Grad.

background_rotate90(ind,clockwise,resize) Dreht das Bild im Hitnergrund 90 Grad. clockwise gibt an ob das Bild im Uhrzeigersinn gedreht wird. resize gibt an ob das Bild vergrössert werden soll (dass die Breite und Höhe umgedreht wird).

background_rotate(ind,angle,quality) Dreht das Bild des Hintergrundes mit dem Winkel (angle), quality gibt die Qualität des Bildes (1-9) an. Je höher der Wert, umso besser die Qualität aber desto länger dauert es.

background_resize(ind,w,h,corner) Vergrössert das Bild des Hintergrundes mit w x h Pixel. Das Bild wird nicht vergrössert. corner gibt den Platz des originalen Bildes an (1-9 wie am Ziffernblock).

background_stretch(ind,w,h,quality) Dehnt das Bild des Hintergrundes mit der Grösse w x h und der angegebenen Qualität (1-9).

background_scale(ind,xscale,yscale,quality,corner,resize) Verändert den Umfang des Bilder des Hintergrundes mit dem Faktor xscale und yscale. quality gibt die Qualität an(1-9). resize gibt an ob das Bild vergrössert werden muss. corner gibt die Position des Originalbildes (1-9 wie am Ziffernblock) an, wenn die Bilder nicht vergrössert werden.

background_black_white(ind) Verändert das Bild des Hintergrundes in Schwarz/Weiß.

background_set_hue(ind,amount) Setzt den Farbton (0-255) für das Bild des Hintergrundes.

background_set_hue_partial(ind,from,till,newhue) Alle Pixel mit einem Farbton zwischen from und till werden mit dem neuen newhue ersetzt.

background_shift_hue(ind,amount) Versetzt den Farbton für den Hintergrund um den angegebenen Wert.

background_change_value(ind,amount) Ändert den Farbwert (0-255) des Bilder des Hintergrundes.

background_change_saturation(ind,amount) Ändert die Farbsättigung (0-255) des Bildes des Hintergrundes.

background_fade(ind,col,amount) Blendet das Bilder des Hintergrundes in Richtung col (Farbe) mit der Anzahl amount (0-255).

background_screendoor(ind,amount) Gibt ein Anzahl amount (0-255) der Türtransparenz der Bildes des Hintergrundes an.

background_blur(ind,amount) Verwischt das Bild des Hintergrundes mit der angegeben Menge (1-9).

37.4 Pfade

Es ist möglich Pfade zu erstellen und Punkte den Pfaden zuzuordnen. Verändere nie einen Pfad welcher von einer Instanz benutzt wird.

path_set_kind(ind,val) Setzt die Art der Verbindung des Pfades mit dem angegebenen Index (0 = gerade, 1 = sanft).

path_set_closed(ind,val) Gibt an ob der Pfad geöffnet (false) oder geschlossen sein muss (true).

path_set_precision(ind,prec) Gibt die Präzision an mit der die „Flüssigkeit“ des Pfades berechnet werden soll (sollte zwischen 1 und 8 liegen).

path_add() Fügt einen neuen leeren Pfad hinzu. Der Index des Pfades wird zurückgegeben.

path_delete(ind) Löscht den Pfad mit dem angegebenen Index.

path_duplicate(ind) Erstellt eine Kopie des Pfades mit dem gegebenen Index her. Gibt den Index der Kopie wieder.

path_assign(ind,path) Verbindet den angegebenen Pfad mit dem Pfad ind. So erstellt die eine Kopie von path. Damit kannst du einen existierenden Pfad einen neuen Pfad zuweisen.

path_add_point(ind,x,y,speed) Fügt eine Punkt zu dem Pfad mit dem angegebenen Index in die Position (x, y) mit der angegebenen Geschwindigkeitsfaktor. Denke daran das ein Faktor von 100 die aktuelle Geschwindigkeit darstellt. Niedrigere Werte verlangsamen und höhere beschleunigen.

path_insert_point(ind,n,x,y,speed) Fügt dem gegebenen Pfad einen Punkt vor dem Punkt n an, an der Position (x,y) und mit dem gegebenen Geschwindigkeitsfaktor.

path_change_point(ind,n,x,y,speed) Verändert den Punkt n im Pfad mit dem gegebenen Index zur Position (x,y) und dem gegebenen Geschwindigkeitsfaktor.

path_delete_point(ind,n) Löscht den Punkt n aus dem Pfad ind.

path_clear_points(ind) Löscht alle Punkte im Pfad, wird in einen leeren Pfad umgewandelt.

path_reverse(ind) Dreht den Pfad um.

path_mirror(ind) Spiegelt den Pfad horizontal (an seinem Zentrum).

path_flip(ind) Spiegelt den Pfad vertikal (an seinem Zentrum).

path_rotate(ind,angle) Rotiert den Pfad im Uhrzeigersinn ind angle Grad (um sein Zentrum).

path_scale(ind,xscale,yscale) Skaliert den Pfad mit den gegebenen Faktoren (aus seinem Zentrum).

path_shift(ind,xshift,yshift) Versetzt den Pfad mit den gegebenen Werten.

37.5 Skripte

Skripte können nicht während des Ausführung eines Spieles geändert werden. Skripte sind Teil der Spiellogik. Verändern von Skripten würde leicht zu Fehlern führen. Aber es gibt andere Wege dies zu bewerkstelligen. Wenn du wirklich ein Stück Code ausführen willst, welcher noch nicht beim Design des Spieles vorhanden ist (z. b. von einer Datei) kannst du folgende Funktionen benutzen:

execute_string(str) Führt das Stück Code des Strings aus.

execute_file(fname) Führt das Stück Code in der Datei aus.
Manmal willst du einen Skriptindex in einer Variable speichern und ausführen. Dafür kannst du folgende Funktion verwenden

script_execute(scr,arg0,arg1,…) Führt das Skript mit dem Index scr und den angegebenen Argumenten aus.

37.6 Daten-Dateien

Für das Manipulieren von Daten-Dateien sind keine Routinen verfügbar.

37.7 Time Lines

Die folgenden Routinen sind für das Erstellen und Verändern von Zeitleisten verfügbar. Verändere keine Zeitleisten die gerade benutzt werden!

timeline_add() Erstellt eine neue Zeitleiste. Es gibt den Index der Zeitleiste zurück.

timeline_delete(ind) Löscht die Zeitleiste mit dem angegebenen Index. Gehe sicher dass keine Instanz dieser Zeitleiste in irgendeinem Raum benutzt.

timeline_moment_add(ind,step,codestr) Fügt eine Code Aktion in die Zeitleiste um angegebenen Zeitpunkt. codestr beinhaltet den Code der Aktion. Wenn der Schritt (step) nicht vorhanden ist, wird er erstellt. Also kannst du mehre Code Aktionen zum gleichen Zeitpunkt hinzufügen.

timeline_moment_clear(ind,step) Diese Funktion löscht die Aktionen zu einem gewissen Zeitpunkt.

37.8 Objects

Auch Objekte können manipuliert werden und während des Spielablaufes erstellt werden. Ändere oder lösche NIE ein Objekt wo Instanzen vorhanden sind. Das kann zu unerwarteten Effekten führen, da die bestimmten Objekteigenschaften mit der Instanz gespeichert werden, daher hat das Verändern des Objekts nicht den gewünschten Erfolg.

object_set_sprite(ind,spr) Setzt den Sprites des Objektes mit dem angegebenen Index. Benütze –1 um den aktuellen Sprite vom Objekt zu entfernen.

object_set_solid(ind,solid) Sollen die Instanzen die vom Objekt erstellt werden fest sein oder nicht (true oder false).

object_set_visible(ind,vis) Sollen die Instanzen die vom Objekt erstellt werdesichtbar sein oder nicht (true oder false).

object_set_depth(ind,depth) Setzt die voreingestellte Zeichentiefe der Instanzen der erstellten Objektes.

object_set_persistent(ind,pers) Sollen die Instanzen die vom Objekt erstellt werden, dauernd sein oder nicht (true oder false).

object_set_mask(ind,spr) Setzt die Sprite Maske des Objektes mit dem angegebenen Index. Benutze – 1 um die Maske zu dem Sprites des Objektes zu setzen.

object_set_parent(ind,obj) Setzt das Elternobjekt. Benutze –1 um kein Elternobjekt zu haben. Das Ändern des Elternobjektes ändert das Verhalten der Instanzen des Objektes.

Die folgenden Routinen sind nützlich um neue Objekte on the fly zu erstellen. Wie beim Verändern allen anderen Routinen, sei vorsichtig, dass du nicht die ganze Zeit neue Objekte erstellst.

object_add() Fügt ein neues Objekt ein. Der Index des Objekte wird zurückgegeben. Du kannst den Index benützen um bestimmte Eigenschaften des Objektes zu setzen und für das Erstellen von

Instanzen des Objektes.
object_delete(ind) Löscht das Objekt mit dem angegeben Index. Gehe sicher das keine Instanz des Objektes in irgendeinem Raum existiert.

object_event_add(ind,evtype,evnumb,codestr) Um dem Objekte ein Verhalten zu geben, müssen wir dem Objekt Eigenschaften definieren. Du kannst nur Code-Aktionen für die Events hinzufügen. Du musst das Objekt nennen, den Event Typ. die Event Nummer (benütze die Konstanten welche für die event_erform() Funktion angeben ist). Zuletzt gib den Code Zeichenkette welche ausgeführt werden soll, an. Du kannst mehrer Code Aktionen zu jedem Event hinzufügen.

object_event_clear(ind,evtype,evnumb) Du kannst diese Funktion benützen umalle Aktionen für ein bestimmtes Event zu löschen. Das Erstellen von Objekten ist teilweise nützlich wenn du ein Skript oder Action Bibliothek erstellst. Ein Initialisierungsskript kann Objekte erstellen, die Texte anzeigen und ein anderes Skript kann solche Objekt mit einem bestimmten Text hinzufügen. In dieser Art hast du einen einfachen Mechanismus um Text anzuzeigen oder Objekte zu erstellen, die das Standartinterface benutzen.

37.9 Rooms

Manipulieren von Räumen ist gefährlich. Du solltest bedenken dass Räume sich die ganze Zeit ändern, abhängig davon was im Spiel passiert. Normal ist hier nur der momentan aktuelle Raum eingeschlossen und es gibt eine Menge Routinen, wie vorher beschrieben, die Instanzen, Hintergründe und Tiles im aktiven Raum ändern. Aber das Ändern eines aktiven Raumes sollte das Letzte sein, wenn der Raum dauernd ist. Daher sollst du nie Teile des aktuellen Raumes oder irgendeinen Raum, welcher dauernd ist oder schon besucht wurde, manipulieren. Solche Änderungen werden generell nicht wahrgenommen aber manchmal passieren dann unerwartete Fehler. Da Räume in einem komplizierten Verfahren verlinkt sind, gibt es keine Routine um einen Raum zu löschen.
Die folgenden Routinen sind verfügbar.

room_set_width(ind,w) Setzt die Breite des Raumes mit dem angegeben Index.

room_set_height(ind,w) Setzt die Höhe des Raumes mit dem angegeben Index.

room_set_caption(ind,str) Setzt die Überschrift des Raumes mit dem angegeben Index.

room_set_persistent(ind,val) Gibt an ob der Raum mit dem angegeben Index dauernd ist oder nicht.

room_set_code(ind,str) Setzt den Initialisierungscode (str) für den Raum mit dem angegeben Index.

room_set_background_color(ind,col,show) Setzt die Farbeigenschaften für einen Raum mit dem angegeben Index. Wenn es keinen Hintergrund hat. gibt col die Farbe an und show gibt an ob die Farbe angezeigt werden soll oder nicht.

room_set_background(ind,bind,vis,fore,back,x,y,htiled,vtiled,hspeed,vspeed,alpha) Setzt den Hintergrund mit Index (0-7) für den Raum mit dem angegebenen Index. vis gibt an ob der Hintergrund sichtbar ist und fore ob es ein Vordergrund ist. back ist der Index der Hintergrund-Datei. x, y geben die Position des Bildes an und htiled und vtiled gibt an ob das Bild geteilt wird. hspeed und vspeed geben die Geschwindigkeit mit welcher der Hintergrund sich bewegt und alpha gibt die Alphatranzparenz an (1 = fest und am schnellsten).

room_set_view(ind,vind,vis,left,top,width,height,x,y,hborder,vborder,hspeed,vspeed,obj) Setzt die View mit dem angegeben Index vind (0-7) im Raum mit dem angegeben Index. vis gibt an ob die

View sichtbar ist, left, top, width und height geben die Position des Viewes im Raum an, x,y geben die Position am Bildschirm an. Wenn die View einem Objekt folgen soll, geben hborder und vborder den geringsten sichtbaren Rand an, der um das Objekt bleiben soll. hspeed und vspeed geben die maximale Geschwindigkeit an, mit welches die View sich bewegen kann. obj ist der Index des Objektes oder der Index der Instanz.

room_set_view_enabled(ind,val) Gibt an ob die View für den Raum mit dem angegeben Index eingeschaltet ist oder nicht.

room_add() Fügt einen neuen Raum hinzu. Es gibt den Index den Raumes zurück. Bedenke dass der Raum nicht Teil der Raumreihenfolge ist. So hat der neue Raum keinen vorhergehenden oder nachfolgenden Raum. Wenn du in einen hinzugefügten Raum gehen willst muss du den Index des Raumes angeben.

room_duplicate(ind) Fügt eine Kopie des Raumes mit dem angegeben Index. Es gibt den Index des Raumes zurück.

room_assign(ind,room) Verbindet den Raum mit dem Raum ind. Somit wird der Raum kopiert.

room_instance_add(ind,x,y,obj) Fügt eine neue Instanz des Objektes obj zu dem Raum mit dem angegeben Index. Es gibt den Index der Instanz zurück.

room_instance_clear(ind) Zerstört alle Instanzen des Raumes mit dem angegeben Index.

room_tile_add(ind,back,left,top,width,height,x,y,depth) Fügt eine neues Tile im Raum auf der angegebenen Position ein. Es gibt den Index des Tiles
zurück. back ist der Hintergrund wo der Tile genommen wird, left, top, widh und height geben den Teil des Hintergrundes an, welcher das Tile formt. x, y ist die Position des Tiles im Raum und depth ist die Zeichentiefe des Tiles an.

room_tile_add_ext(ind,back,left,top,width,height,x,y,depth,xscale,yscale,alpha) Gleiche wie vorher aber diesmal kannst du einen Skalierungs-faktor in x und y Richtung und eine Alphatransparenz für das Tile angeben. room_tile_clear(ind) Zerstört alle Tiles vom angegebenen Raum.

38. GML: Dateien, Registry und Programme

In weiter fortgeschrittenen Spielen willst du vielleicht Daten aus einer Datei lesen, die du dem Spiel beilegst. Z.B. könntest du eine Datei machen, die beschreibt, in welchen Momenten bestimmte Dinge passieren sollen. Oder du willst Informationen speichern für das nächste Mal, wenn das Spiel gestartet wird (etwa den aktuellen Raum). Die folgenden Funktionen existieren um Daten in Textdateien zu lesen und zu schreiben: file_text_open_read(fname) Öffnet die angegebene Datei zum lesen. Die Funktion gibt die ID zurück, welche für andere Funktionen benötigt wird. Du kannst mehrere Dateien auf einmal öffnen (maximal 32). Vergesse nicht sie zu schliessen wenn du mit ihnen fertig bist!

file_text_open_write(fname) Öffnet die angegebene Datei zum Schreiben, sie wird erstellt falls sie nicht existiert. Die Funktion gibt die ID zurück, welche für andere Funktionen benötigt wird.

file_text_open_append(fname) Öffnet die angegebene Datei um Daten anzufügen, sie wird erstellt falls sie nicht existiert. Die Funktion gibt die ID zurück, welche für andere Funktionen benötigt wird.

file_text_close(fileid) Schliesst die Datei mit der angegebenen ID.

file_text_write_string(fileid,str) Schreibt den string in die Datei mit der angegebenen ID.

file_text_write_real(fileid,x) Schreibt einen reelen Zahlenwert in die Datei mit der angegebenen ID.

file_text_writeln(fileid) Schreibt einen "neuezeile" Wert in die Datei mit der angegebenen ID.

file_text_read_string(fileid) Liest einen string von der Datei mit der angegebenen ID und gibt ihn zurück. Ein string endet mit dem Zeilenende ("neuezeile" Wert).

file_text_read_real(fileid) Liest einen reelen Zahlenwert aus und gibt diesen zurück.

file_text_readln(fileid) Überspringt den Rest der Zeile und beginnt in der nächsten Zeile.

file_text_eof(fileid) Gibt zurück ob das Dateiende erreicht wurde.

Um Dateien im Dateisystem zu manipulieren können die folgenden Funktionen genutzt werden:

file_exists(fname) Gibt zurück ob die Datei mit dem angegebenen Namen existiert (true) oder nicht (false).

file_delete(fname) Löscht die Datei mit dem angegebenen Namen.

file_rename(oldname,newname) Ändert den Dateinamen von oldname nach newname.

file_copy(fname,newname) Kopiert die Datei fname in die Datei newname.

directory_exists(dname) Gibt zurück, ob das angegebene Verzeichnis existiert.

directory_create(dname) Erstellt das angegebene Verzeichnis, falls es nicht existiert.

file_find_first(maske,attribut) Gibt den Namen der ersten Datei zurück, die mit der Maske und den Attributen übereinstimmt. Existiert keine solche Datei, wird ein Leerstring zurückgegeben. Die Maske kann einen Pfad und Auslassungszeichen enthalten, z.B. "C:\Temp*.doc". Die Attribute geben die zusätzlichen Dateien an. (Die normalen Dateien werden also immer zurückgegeben, wenn sie mit der Maske übereinstimmen).
Folgende Konstanten können als Attribute verwendet werden:

fa_readonly	schreibgeschützte
fa_hidden	Dateien
fa_sysfile	versteckte Dateien

fa_volumeid	Systemdateien Volume-ID Dateien
fa_directory	Verzeichnisse
fa_archive	archivierte Dateien

(die Konstanten können mit dem |-Operator [bitweises Oder] verknüpft werden)

file_find_next() Gibt den Namen der nächsten Datei zurück, die mit der vorher angegebenen Maske und den Attributen übereinstimmt. Existiert keine solche Datei, wird ein Leerstring zurückgegeben.

file_find_close() Muss nach allen file_find-Funktionen aufgerufen werden, um Speicher freizugeben.

file_attributes(fname,attr) Gibt zurück, ob die Datei alle Attribute hat, die in attr angegeben sind. Wenn der Spieler secure mode in den Voreinstellungen angegeben hat, darfst du in einigen Funktionen keinen Pfad angeben, und nur Dateien im Anwendungsverzeichnis können z.B. geschrieben werden.

Die folgenden Funktionen können genutzt werden um Dateinamen zu verarbeiten. Beachte das sie nicht mit den Dateien sondern nur mit dem Namen arbeiten.

filename_name(fname) Gibt den Namen der Datei mit Erweiterung aber ohne Pfad wieder.

filename_path(fname) Gibt den Pfad der Datei inklusive des finalen Backslashs zurück.

filename_dir(fname) Gibt das Verzeichniss der Datei zurück, welches normalerweise der Pfad ohne den finalen Backslash ist.

filename_drive(fname) Gibt das Laufwerk der Datei zurück.

filename_ext(fname) Gibt die Dateierweiterung wieder, inklusive des Punktes.

filename_change_ext(fname,newext) Gibt den angegebenen Dateinamen, mit der Erweiterung an (inklusive des Punktes) welche zu einer neuen Erweiterung verändert wurde. Indem ein leerer string angegeben wird kann die Erweiterung gelöscht werden.

In seltenen Situation müssen Daten von binären Dateien gelesen werden. Die folgenden low-level Routinen existieren hierfür:

file_bin_open(fname,mod) Öffnet die Datei mit dem angegebenen Namen. Der Modus gibt an, was mit der Datei getan werden kann: 0 = lesen, 1 = schreiben, 2 = beides). Die Funktion gibt die ID zurück, welche für andere Funktionen benötigt wird. Du kannst mehrere Dateien auf einmal öffnen (maximal 32). Vergesse nicht sie zu schliessen wenn du mit ihnen fertig bist!

file_bin_rewrite(fileid) Schreibt die Datei mit der angegebenen ID neu, d.h. löscht sie und beginnt am Anfang.

file_bin_close(fileid) Schliesst die Datei mit der angegebenen ID.

file_bin_size(fileid) Gibt die Dateigrösse (in Byte) der Datei mit der angegebenen ID wieder.

file_bin_position(fileid) Gibt die aktuelle Position (in Byte; 0 ist die erste Position) in der Datei mit der angegebenen ID zurück.

file_bin_seek(fileid,pos) Bewegt die aktuelle Position zur angegebenen. Um es möglich zu machen die Position zur Dateigrösse zu bewegen, bevor geschrieben wird.

file_bin_write_byte(fileid,byte) Schreibt ein Byte an die aktuelle Position der angegebenen Datei

mit der ID.

file_bin_read_byte(fileid) Liest ein Byte der Datei und gibt es zurück.

Wenn der Spieler den Sicherheitsmodus betreibt, ist es für eine Anzahl dieser Funktionen nicht erlaubt einen Pfad anzugeben und es können nur Dateien im Anwendungsordner geschrieben werden.

Die folgenden drei (schreibgeschützten) Variablen können nützlich sein:

game_id* ID für das Spiel. Du kannst diese Nummer verwenden, wenn du einen einmaligen Dateinamen brauchst.

working_directory* Anwendungsverzeichnis (enthält nicht den abschliessenden Schrägstrich)

temp_directory* Temporäres Verzeichnis.(enthält nicht den abschliessenden Schrägstrich) Alle Dateien darinnen werden beim Ende des Spiels gelöscht.
Manchmal möchtest du vielleicht dem Spieler die Möglichkeit geben, dem Spiel Kommandozeilenargumente zu übergeben (z.B. für Cheats). Um diese Argumente zu bekommen, kannst du diese zwei Funktionen verwenden:

parameter_count() Gibt die Anzahl der Kommandozeilenparameter zurück (Achtung: der Name des Programm ist auch einer davon)

parameter_string(n) Gibt Kommandozeilenparameter n zurück. Der erste Parameter (der Programmname) hat den Index 0.

Du kannst Umgebungsvariablen mit folgender Funktion lesen.

environment_get_variable(name) Gibt den Wert der Umgebungsvariablen mit dem gegebenen Namen zurück (als String).

Wenn du nur wenige Daten zwischen verschiedenen Spielstarts speichern willst, existiert ein einfacherer Mechanismus als eine Datei zu nutzen. Du kannst die Registry verwenden. Die Registry ist eine grosse Datenbank, die Windows verwendet, um alle Einstellungen für Programme zu speichern. Ein Eintrag hat einen Namen und einen Wert. Du kannst Strings und Zahlen verwenden. Folgende Funktionen existieren dazu:

registry_write_string(name,str) Erstellt einen Eintrag in der Registry mit demgegebenen Namen und einem String-Wert.

registry_write_real(name,x) Erstellt einen Eintrag in der Registry mit dem gegebenen Namen und einem Zahlenwert.

registry_read_string(name) Gibt den String zurück, der im Eintrag name steht. (Falls der Eintrag nicht existiert, wird ein Leerstring zurückgegeben)

registry_read_real(name) Gibt den Zahlenwert zurück, der im Eintrag name steht.
(Falls der Eintrag nicht existiert, wird 0 zurückgegeben)

registry_exists(name) Gibt zurück, ob ein Schlüssel mit dem angegebenen Namen existiert.

Eigentlich sind die Registry-Einträge in Schlüssel gruppiert. Die obigen Funktionen arbeiten alle mit einem Schlüssel, der speziell für dein Spiel erstellt wird. Dein Programm kann das verwenden, um Informationen über das System zu bekommen, auf dem das Spiel läuft. Du kannst auch Werte in

anderen Schlüsseln lesen. Du kannst sie auch lesen, aber sei vorsichtig: DU KÖNNTEST SO LEICHT DEIN BETRIEBSSYSTEM UNFÄHIG MACHEN! (Im "secure mode" ist schreiben nicht erlaubt) Beachte, dass Schlüssel in Gruppen plaziert sind. Alle folgenden Funktionen arbeiten standardmässig mit HKEY_CURRENT_USER. (das kannst du aber auch verändern). Wenn du z.B. das aktuelle Temp-Verzeichnis herausfinden willst, schreibe

```
path = registry_read_string_ext('\Environment','TEMP');
```

Folgende Funktionen existieren:

registry_write_string_ext(key,name,str) Erstellt einen Eintrag im Schlüssel key mit dem gegebenen Namen und einem String-Wert.
registry_write_real_ext(key,name,x) Erstellt einen Eintrag im Schlüssel key mit dem gegebenen Namen und einem Zahlenwert.
registry_read_string_ext(key,name) Gibt den String zurück, der im Schlüssel key im Eintrag name steht. (Falls der Eintrag nicht existiert, wird ein Leerstring zurückgegeben)

registry_read_real_ext(key,name) Gibt den Zahlenwert zurück, der im Schlüssel key im Eintrag name steht. (Falls der Eintrag nicht existiert, wird 0 zurückgegeben)
registry_exists_ext(key,name) Gibt zurück, ob der gegebene Name im Schlüssel key existiert.
registry_set_root(root) Ändert das Stammverzeichnis für die anderen Funktionen. Verwende folgende Werte:
0 = HKEY_CURRENT_USER
1 = HKEY_LOCAL_MACHINE
2 = HKEY_CLASSES_ROOT
3 = HKEY_USERS

Um bestimmte Programmeinstellungen zu speichern, wird oft eine INI-Datei verwendet. INI-Dateien enthalten verschiedene Bereiche. Jeder Bereich enthält einige Name-Wert-Paare. Eine typische INI-Datei sieht etwa so aus:

```
[Fenster]
Oben=100
Links=100
Titel=Das beste
Spiel

[Spiel]
MaxScore=21321
```

Diese Datei enthält zwei Bereiche: Einen mit Namen "Fenster" und einen Bereich "Spiel". Der erste Bereich enthält drei Paare. Die ersten beiden sind Zahlen, während das dritte ein String ist. Solche INI-Dateien sind einfach zu erstellen und zu verändern. Die folgenden Funktionen existieren, um Daten zu lesen und zu schreiben:

ini_open(name) Öffnet die INI-Datei mit dem gegebenen Namen. Die Datei muss sich im selben Ordner wie das Spiel befinden!
ini_close() Schliesst die aktuell geöffnete INI-Datei.

ini_read_string(section,key,default) Liest den String mit dem Namen key im Bereich section. Existiert der String nicht, wird default zurückgegeben.

ini_read_real(section,key,default) Liest die Zahl mit dem Namen key im Bereich section. Existiert die Zahl nicht, wird default zurückgegeben.

ini_write_string(section,key,value) Schreibt einen String mit dem Namen key in den Bereich section.

ini_write_real(section,key,value) Schreibt eine Zahl mit dem Namen key in den Bereich section.

ini_key_exists(section,key) Gibt zurück, ob der Name key in dem Bereich section existiert.
ini_section_exists(section) Gibt zurück, ob der Bereich section existiert.

ini_key_delete(section,key) Löscht den Schlüssel key aus dem Bereich section.

ini_section_delete(section) Löscht den Bereich section.

Game Maker hat auch die Möglichkeit, externe Programme aufzurufen. Es gibt zwei Funktionen dafür: execute_program startet ein Programm, möglicherweise mit Argumenten. Die Funktion kann auf das Programm warten (das Spiel wird unterbrochen) oder nicht. execute_shell öffnet eine Datei. Das kann irgendeine Datei sein, mit der ein Programm verknüpft ist (z.B. ein *.txt- Datei - hier wird Notepad gestartet). Es kann auch eine Datei sein. Die Funktion kann nicht warten, also fährt das Spiel fort.
execute_program(prog,arg,wait) Führt das Programm prog mit den Argumenten arg aus. wait legt fest, ob das Spiel warten soll.

execute_shell(prog,arg) Führt das Programm (oder die Datei) in der Shell aus.

Beide Funktionen funktionieren nicht, wenn "secure mode" in den Voreinstellungen aktiviert ist. Das kannst du mit der
folgenden schreibgeschützten Variable überprüfen:

secure_mode* Ob das Spiel im "secure mode" läuft oder nicht.

39. Partikeleffekte

Diese Funktionen sind nur in der registrierten Version vom Game Maker verfügbar.

Partikelsysteme sind für das Erstellen von Spezialeffekten geeignet. Partikeln sind kleine Elemente (repräsentiert von einem kleinen Sprite, einem Pixel oder einer kleinen Form). Solche Partikeln bewegen sich herum, sie befolgen vorher bestimmte Regeln und können die Farbe, während sie sich bewegen, ändern. Viele solcher Partikel untereinander können z.B. ein Feuerwerk, Regen,Schnee, Sternenfelder, fliegende Trümmer, usw. erzeugen.

Game Maker beinhaltet ein umfassendes Partikelsystem, das zum Erstellen von großartigen Effekten benutzt werden kann. Es ist wirklich nicht einfach zu benutzen, deshalb lies vorher dieses Kapitel sorgfältig durch. Ein Partikelsystem hat viele Parameter und es ist nicht immer einfach zu verstehen, wie der gewünschte Effekt erreicht werden kann.

Zuerst gibt es Partikeltypen. Eine Partikeltyp definiert ein besonderen Teil des Partikels. Solche Typen haben viele Parameter die die Form, Grösse, Farbe und Bewegung des Partikels beschreiben.

Zweitens gibt es Partikelsysteme. Es können verschiedene Partikelsysteme im Spiel sein. Ein Partikelsystem kann Partikel verschiedenen Types haben. Ein Partikelsystem hat "Emitter", die Partikel erstellen, entweder kontinuierlich oder in Salven. Es kann auch "Attractors" haben die Partikel anziehen. Zuletzt gibt es "Destroyers" die Partikel zerstören.

Bedenke, dass dein Spiel für die Aktualisierung des Partikelsystem und das Zeichnen verantwortlich ist. Also wird ein typisches Partikelsystem mit einem Objekt im Spiel verbunden. Im step event ruft es die Funktion zur Aktualisierung des Partikelsystems und im drawing event ruft es die Funktion zum Zeichnen des Partikelsystems auf.

39.1 Partikeltypen

Ein Partikeltyp beschreibt die Form, Farbe und Bewegung einer bestimmten Art eines Partikels. Es besitzt viele Parameter. Die folgenden Parameter beschreiben die Form des Partikels:

shape Gibt die Form des Partikels an. Das ist nur relevant wenn das Partikelsystem keine Sprites als Partikel verwendet. Es kann folgende Form sein:
pt_shape_pixel
pt_shape_disk
pt_shape_square
pt_shape_diamond
pt_shape_star
pt_shape_plus
pt_shape_cross

sprite Gibt den Sprite für das Partikelsystem an, nur wenn das Partikelssystem auf Sprites basiert.

sprite_animate Soll das Sprite animiert (wahr) oder nicht (falsch) sein.

sprite_stretch Soll die Spriteanimation über die ganze Lebenszeit des Partikels (wahr) gedehnt werden oder soll die Animation wiederholt werden (falsch). (Der vorherige Wert muss auf wahr gesetzt sein.)

sprite_random Soll die Spriteanimation bei einem zufälligen Einzelbild (wahr) oder beim ersten Einzelbild (falsch) starten. Wenn das Sprite nicht animiert ist benütze es um ein zufälliges Einzelbild anstelle des ersten zu bekommen.

size_min, size_max Grösse der Partikel. Die Grösse wird zufällig zwischen den gegebenen Grenzen gewählt. Diese Werte sind für Partikel aus Pixelformen nicht relevant. Für spritebasierende Partikel gibt des den benutzten Skalierungsfaktor an.

size_incr Steigung der Grösse in jedem Schritt. Es lässt die Partikel schrumpfen oder wachsen (ein Grösse unter 0 wird als 0 gewertet) über die Zeit.

size_rand Eine zufälliger Ausgleich der Grösse in jedem Schritt. Wenn du eine großen Wert angibst, werden die Partikel schnell die Grösse ändern, dass gibt einen Blinkeffekt.

color_start, color_middle, color_end Die Farbe des (nicht spritebasierenden) Partikels. Es startet mit der ersten Farbe und über seine Lebenszeit ändert es langsam die Farbe zur mittleren Farbe und letztendlich zur letzten Farbe.

Ein Partikel lebt für eine begrenzte Zeit. Das wird durch folgende Parameter angegeben:

life_min, life_max Die Lebenszeit des Partikels. Ein zufälliger Wert zwischen den gegebenen Grenzen wird gewählt wenn das Partikel erstellt wird. Wenn das Leben zu Ende ist, stirbt das Partikel.

Die folgenden Parameter beeinflussen die Bewegung der Partikel:

speed_min, speed_max Geschwindigkeit des Partikels (in Pixel pro Schritt). Ein zufälliger Wert zwischen den gegebenen Grenzen wird gewählt wenn das Partikel erstellt wird.

speed_incr Geschwindigkeitssteigerung pro Schritt. Benutze einen negativen Wert um den Partikel zu bremsen (die Geschwindigkeit wird nie kleiner als 0).

speed_rand Maximal zufällige Geschwindigkeit, die der Geschwindigkeit pro Schritt in jedem Schritt hinzugefügt wird. Es kann benutzt werden um den Partikeln einen zufälligeren Weg zu geben.

dir_min, dir_max Richtung der Bewegung der Partikel (im Uhrzeigersinn; 0 ist eine Bewegung nach rechts). Ein zufälliger Wert zwischen den gegebenen Werten wird gewählt wenn das Partikel erstellt wird. Wenn du also z.B. 0 und 360 als Werte wählst, geht das Partikel in eine zufällige Richtung.

dir_incr Richtungsänderung in jedem Schritt. Das bewegt die Partikel in einer Kreisform.

dir_rand Maximal zufällige Menge der Richtung die pro Schritt der Richtung in jedem Schritt hinzugefügt wird. Es kann benützt werden um de Partikeln einen zufälligeren Weg zu geben.

grav_amount Menge der Anziehungskraft in jedem Schritt.

grav_dir Richtung der Anziehungskraft; verwende 270 für eine Abwärtsbewegung.

Partikel können selbst Partikel erstellen. Dafür gibt es folgende Parameter:

step_type Typ des neuen Partikels der in jedem Schritt erzeugt wird.

step_number Nummer solcher Partikel die in jedem Schritt erzeugt werden. Diese zwei Parameter können benützt werden um Partikels einen Schweif beim Fliegen zu geben. Sei vorsichtig, dass die gesamte Anzahl der Partikels nicht zu hoch wird.

death_type Gibt den Typ der Partikel an, die erstellt werden, wenn das Partikel stirbt.

death_number Die Anzahl der Partikel die erstellt werden wenn ein Partikel stirbt. Diese zwei Parameter können benützt werden um Sekundäreffekte zu erzeugen, z. B. ein Partikel explodiert nach eine Weile in einige andere Partikeln. Sei vorsichtig damit. Das kann leicht eine hohe Anzahl von Partikel erzeugen. Bedenke, dass diese Partikel nur erstellt werden, wenn die Partikel am Ende ihres Lebens sterben. Nie wenn sie aufgrund eines Destroyers sterben (siehe unten).

Die folgenden Funktionen sind verfügbar um Partikeltypen zu bestimmen. Bedenke dass jede Funktion eine id vom Typ des Arguments bekommt. Diese muss zwischen 0 und 1000 sein. Du solltest kleine positive ganze Zahlen bevorzugen (z.B. 1, 2, 3, 4).

part_type_create() Erstellt einen neuen Partikeltyp. Die Funktion gibt den Index des Typs zurück. Dieser Index muss in alle Aufrufen benützt werden um die Eigenschaften des Partikeltyps zu setzen.

part_type_destroy(ind) Zerstört den Partikeltyp ind. Rufe diese Funktion auf wenn du es nicht mehr brauchst um Platz zu sparen.

part_type_destroy_all() Zerstört alle Partikeltypen die erstellt wurden.

part_type_exists(ind) Gibt zurück, ob der angegebene Partikeltyp existiert.

part_type_clear(ind) Setzt den Partikeltyp ind auf seine voreingestellten Einstellungen zurück.

part_type_shape(ind,shape) Setzt die Form des Partikeltyps. Benütze einer der folgenden Konstanten (voreingestellt pt_shape_pixel).
pt_shape_pixel
pt_shape_disk
pt_shape_square
pt_shape_diamond
pt_shape_star
pt_shape_plus
pt_shape_cross

part_type_sprite(ind,sprite,animate,stretch,random) Setzt den Sprite für den Partikeltyp (wenn das Partikelsystem spritebasiert ist; siehe unten). Mit animate gibst du an ob das Sprite animiert sein soll. Mit stretch ob die Animation über seine Lebenszeit gestreckt werden soll. Und mit random gibst du an ob ein zufälliges Einzelbild als Startbild gewählt werden soll.

part_type_size(ind,size_min,size_max,size_incr,size_rand) Setzt die Grössenparameter des Partikeltyps. (Die Grösse 1 ist eingestellt und die Grösse ändert sich nicht.)

part_type_color(ind,color_start,color_middle,color_end) Setzt die Farbparameter des Partikeltyps. (Eingestellt: alle Farben sind weiss.)

part_type_color2(ind,color_start,color_end) Wie oben, aber diesmal wird nur die Start- und Endfarbe gesetzt. Die mittlere Farbe wird dazwischen genommen.

part_type_life(ind,life_min,life_max) Setzt die Lebenszeitgrenze des Partikeltyps. (Eingestellt: beide auf 100.)

part_type_speed(ind,speed_min,speed_max,speed_incr,speed_rand) Setzt die Geschwindigkeitsparameter des Partikeltyps. (Eingestellt: alle Werte sind 0.)

part_type_direction(ind,dir_min,dir_max,dir_incr,dir_rand) Setzt die Richtungsparameter des Partikeltyps. (Eingestellt: alle Werte sind 0.)

part_type_gravity(ind,grav_amount,grav_dir) Setzt die Gravitationsparameter des Partikeltyps. (Eingestellt: es gibt keine Gravitation.)

part_type_step(ind,step_number,step_type) Setzt die Anzahl und den Typ der Partikel die in jedem Schritt für den angegebenen Partikeltyp erzeugt werden. (Eingestellt: kein Partikel wird erzeugt.)

part_type_death(ind,death_number,death_type) Setzt die Anzahl und den Typ der Partikel die erstellt werden, wenn der Partikel des angegebenen Typ stirbt. (Eingestellt: keine Partikel werden erstellt.)

39.2 Partikelsysteme

Partikel leben in Partikelsystemen. Um also Partikel in deinem Spiel zu haben musst du ein oder mehrere Partikelsysteme erstellen. Es können verschiedene Partikelsysteme sein. Beispielsweise, wenn dein Spiele einige Bälle hat und jeder Ball soll einen Schweif Partikel haben, hat jeder Ball sein eigenes Partikelsystem. Der einfachste Weg mit Partikelsystemen umzugehen ist, zuerst eines zu erstellen und dann die Partikel darin zu erstellen, und die vorher angegeben Partikeltypen zu verwenden. Aber wie wir später sehen werden, können Partikelsysteme Emitters enthalten, welche automatisch Partikeln produzieren, Attractors die sie anziehen und Destroyers die sie zerstören.

Partikelsysteme können spritebasiert oder nicht. In einem spritebasierten Partikelsystem wird jedes Partikel vor einem Sprite repräsentiert. In einem nichtspritebasierten Partikelsystem sind Partikel ein Pixel oder sie haben eine geometrische Form (wie ein Quadrat). Du kannst nicht spritebasierte und geometrische Partikel im selben System mixen (das ist aus Effenzienzgründen so) aber du kannst ja verschiedene Partikelsystem dafür definieren.

Dein Spiel ist verantwortlich für die Arbeit des Partikelsystemes. Du musst also das Partikelsystem erstellen (typischerweise im creation event eines Objekts), angeben dass es die Position aller Partikel aktualisieren soll (typischerweise im step event) und die Partikel zeichnen (im draw event des Objektes). Ausserdem solltest du das Partikelsystem zerstören wenn es nicht mehr benötigt wird.

Die folgenden Funktionen arbeiten mit Partikelsystemen:

part_system_create() Erstellt ein neues Partikelsystem. Es gibt den Index des Typs zurück. Der Index muss in jedem Aufruf darunter verwendet werden um die Eigenschaften des Partikelsystem zu setzen.

part_system_destroy(ind) Zerstört das Partikelsystem ind. Rufe die Funktion auf, wenn du das System nicht mehr brauchst, um Speicher zu sparen.

part_system_destroy_all() Zerstört alle Partikelsysteme die erstellt wurden.

part_system_exists(ind) Gibt zurück ob das angegebene Partikelsystem existiert.

part_system_clear(ind) Setzt das Partikelsystem ind auf seine voreingestellten Einstellungen zurück, zerstört alle Partikel, Emitters und Attractors darin.

part_system_sprite_based(ind,set) Ist das Partikelsystem spritebasiert (true) oder nicht (false). (Voreingestellt ist false.)

part_system_draw_order(ind,oldtonew) Setzt die Reihenfolge in welcher das Partikelsystem die Partikels zeichnet. Wenn oldtonew wahr ist, werden die ältesten Partikel zuerst gezeichnert und die neuen liegen über ihnen (voreingestellt). Sonst werden die neuen Partikel zuerst gezeichnet. Das kann einen anderen Effekt ergeben.

part_system_doastep(ind) Die Funktion aktualisiert die Postion aller Partikel im System und lässt die Emitters Partikel erstellen. Sie sollte exakt einmal in jedem Schritt des Spieles aufgerufen

werden.

part_system_draw(ind,x,y) Diese Funktion zeichnet die Partikel im System, relativ auf Position (x,y). Sie sollte im draw event des Objektes aufgerufen werden. Meistens wirst du 0,0 benutzen, da das Partikelsystem im Allgemeinen selbst auf die Position anpasst.

Die folgenden Funktion arbeiten mit Partikel des Partikelsystems:

part_particles_create(ind,x,y,parttype,number) Diese Funktion erstellt eine Anzahl von Partikel des angegeben Types auf Position (x,y) im System.

part_particles_clear(ind) Diese Funktion zerstört alle Partikel im System.

part_particles_count(ind) Diese Funktion gibt die Anzahl der Partikel im System zurück.

39.3 Emitter

Emitter erstellen Partikel. Sie können entweder ein kontinuierlichen Strom von Partikeln oder können eine Anzahl von Partikel herausstossen, wenn die jewelige Funktion aufgerufen wird. Ein Partikelsystem kann eine beliebige Anzahl an Emitters haben. Ein Emitter hat folgende Eigenschaften:

xmin, xmax, ymin, ymax Die Ausdehnung der Region, in der die Partikel erstellt werden.
shape Die Form der Region. Es kann folgende Werte haben:
ps_shape_rectangle
ps_shape_ellipse
ps_shape_diamond
ps_shape_line

distribution Die benütze Verteilung zum Generieren der Partikel. Es kann folgende Werte haben:
ps_distr_linear Lineare Verteilung, es ist überall in der Region möglich
ps_distr_gaussian Gauss-Verteilung, es werden mehr Partikel in der Mitte als auf der Seite der Region verteilt

particle type Der Typ der erstellten Partikel

number Die Anzahl der Partikel, die in jedem Schritt erzeugt werden. Wenn dieser Wert kleiner als 0 ist, wird jedem Schritt ein Partikel mit der Chance -1/number erzeugt. Beispielsweise ein Wert von -5 erzeugt durchschnittlich einen Partikel in jedem fünften Schritt.

Die folgenden Funktionen sind verfügbar um Emitters zu setzen und diese Partikel erstellen zu lassen. Bedenke, dass jede davon den Index des Partikelssystems bekommt welcher als erstes Argument übergeben werden muss.

part_emitter_create(ps) Erstellt einen neuen Emitter im angegebenen Partikelsystem. Die Funktion gibt den Index des Typs zurück. Dieser Index muss in allen Aufrufen unten verwendet werden um die Eigenschaften des Emitters zu setzen.

part_emitter_destroy(ps,ind) Zerstört den Emitter ind im Partikelsystem. Rufe die Funktion auf wenn du Speicher sparen willst.

part_emitter_destroy_all(ps) Zerstört alle Emitters im Partikelsystem die erstellt wurden.

part_emitter_exists(ps,ind) Existiert der angegebene Emitter im Partikelsystem oder nicht.

part_emitter_clear(ps,ind) Setzt den emitter ind auf seine Voreinstellungen zurück.

part_emitter_region(ps,ind,xmin,xmax,ymin,ymax,shape,distribution) Setzt die Region und Verteilung des Emitters.

part_emitter_burst(ps,ind,parttype,number) Stösst eine Anzahl von Partikeln des angegebenen Typs vom Emitter hinaus.

part_emitter_stream(ps,ind,parttype,number) Vom diesen Moment an werden number Partikel des gegebenen Typs in jedem Schritt von dem Emitter erzeugt. Wenn kleiner als 0, wird jedem Schritt ein Partikel mit der Chance -1/number erzeugt. Beispielsweise ein Wert von -5 erzeugt durchschnittlich einen Partikel in jedem fünften Schritt.

39.4 Attractors

Neben Emitters kann ein Partikelsystem auch Attractors enthalten. Ein Attractor zieht die Partikel an (oder drückt sie weg). Ein Partikelsystem kann beliebig viele Attractors haben. Es wird empfohlen nur wenige davon zu benützen, weil sie den Vorgang verlangsamen. Ein Attractor hat folgende Eigenschaften:

x,y Die Position des Attractor.

force Die Anziehungskraft des Attractor. Wie die Kraft wirkt, hängt von den folgenden Parametern ab.

dist Die maximale Entfernung, wo der Attractor noch einen Effekt hat. Nur Partikel, die näher als diese Entfernung sind, werden angezogen.

kind Die Art des Attractor.

Die folgenden Werte existieren:
ps_force_constant Die Kraft ist konstant und unabhängig von der Entfernung.
ps_force_linear Die Kraft steig linear an. Bei der maximalen Entfernung ist die Kraft 0, während auf der Position des Attractor der gegebene Wert erreicht wird.
ps_force_quadratic Die Kraft wächst quadratisch.

additive Wird die Kraft zur Geschwindigkeit und Richtung in jedem Schritt (wahr) addiert oder nur der Postion des Partikels (falsch) hinzugefügt. Wenn dieser Wert gesetzt ist, beschleunigt das Partikel zum Attractor, während es sich sonst mit konstanter Geschwindigkeit bewegt.

Die folgenden Funktionen existieren um Attractors zu definieren. Bedenke, dass jede den Index des Partikelsystems bekommen welches als erstes Argument übergeben werden muss:

part_attractor_create(ps) Erstellt einen neuen Attractor im angegebenen Partikelsystems. Es gibt den Index des Typs zurück. Der Index muss in allen Aufrufen zum Setzen der Eigenschaften des emitter angegeben werden.

part_attractor_destroy(ps,ind) Zerstört den Attractor ind im Partikelsystem. Wenn du sie nicht mehr brauchst, rufe das auf um Speicher zu sparen.

part_Attractor_destroy_all(ps) Zerstört alle Attractor im Partikelsystem welche erstellt wurden.

part_attractor_exists(ps,ind) Gibt zurück ober den angegebene Attractor im Partikelsystem existiert.

part_attractor_clear(ps,ind) Setzt den Attractor ind auf seine Voreinstellung zurück.

part_attractor_position(ps,ind,x,y) Setzt die Position des Attractor ind auf (x, y).

part_attractor_force(ps,ind,force,dist,kind,aditive) Setzt den Force Parameter des Attractor ind.

39.5 Destroyer

Destroyers zerstören Partikel wenn sie in ihrer Region erscheinen. Ein Partikelsystem kann eine beliebige Anzahl von Destroyers haben. Ein Destroyer hat die folgenden Eigenschaften:

xmin, xmax, ymin, ymax Die Ausdehnung der Region wo die Partikel zerstört werden.
shape Form der Region. Es kann folgende Werte haben:
ps_shape_rectangle
ps_shape_ellipse
ps_shape_diamond

Die folgenden Funktionen existieren um Destroyers zu definieren. Bedenke, dass jede den Index des Partikelsystems bekommen welches als erstes Argument übergeben werden muss.

part_destroyer_create(ps) Erstellt einen neuen destroyer im angegebenen Partikelsystem. Es gibt den Index des Typs zurück. Der Index muss in allen Aufrufen zum Setzen der Eigenschaften des destroyer angegeben werden.

part_destroyer_destroy(ps,ind) Zerstört den Destroyer ind im Partikelsystem. Wenn du den Destroyer nicht mehr brauchst, rufe das auf um Speicher zu sparen.

part_destroyer_destroy_all(ps) Zerstört alle Destroyer im Partikelsystem welche erstellt wurden.

part_destroyer_exists(ps,ind) Gibt zurück ob der angegebene Destroyer im Partikelsystem existiert.

part_destroyer_clear(ps,ind) Setzt den Attractor ind auf seine Voreinstellung zurück.

part_destroyer_region(ps,ind,xmin,xmax,ymin,ymax,shape) Setzt die Region des Destroyers.

39.6 Deflectors

Deflectors lenken Partikeln ab, wenn sie in diese Region kommen. Ein Partikelsystem kann eine beliebige Anzahl von Deflectors haben. Ein Deflector hat die folgenden Eigenschaften:

xmin, xmax, ymin, ymax Die Ausdehnung der Region wo die Partikeln abgelenkt werden.

kind Art des deflector. Es kann folgende Werte haben:
ps_deflect_horizontal lenkt die Partikel horizontal ab, typisch für vertikale Wände
ps_deflect_vertical lenkt die Partikel vertikal ab, typisch für horizontale Wände

friction Die Menge der Reibung als Folge des Einschlages mit dem Deflector. Je höher die Menge desto mehr wird der Partikel beim Einschlag eingebremst.

Die folgenden Funktionen existieren um Deflectors zu definieren. Bedenke, dass jede den Index des Partikelsystems bekommt, welches als ersters Argument übergeben werden muss.

part_deflector_create(ps) Erstellt einen neuen deflector im angegebenen Partikelsystems. Es gibt den Index des Typs zurück. Der Index muss in allen Aufrufen zum Setzen der Eigenschaften des Deflectors angegeben werden.

part_deflector_destroy(ps,ind) Zerstört den deflector ind im Partikelsystem. Wenn du sie nicht mehr brauchst, rufe das auf um Speicher zu sparen.

part_deflector_destroy_all(ps) Zerstört alle deflectors im Partikelsystem welche erstellt wurden.

part_deflector_exists(ps,ind) Gibt zurück ob der angegebene Deflector im Partikelsystem existiert.

part_deflector_clear(ps,ind) Setzt den Deflector ind auf seine Voreinstellung zurück.

part_deflector_region(ps,ind,xmin,xmax,ymin,ymax) Setzt die Region des Deflectors.

part_deflector_kind(ps,ind,kind) Setzt die Art des Deflectors.

part_deflector_friction(ps,ind, friction) Setzt die Reibung des Deflectors.

39.7 Changers

Changers ändern Partikel, wenn sie in diese Region kommen. Ein Partikelsystem kann eine beliebige Anzahl von Changers haben. Ein Changer hat die folgenden Eigenschaften:

xmin, xmax, ymin, ymax Die Ausdehnung der Region wo die Partikel geändert werden.
shape Die Art der Region. Es kann folgende Werte haben:
ps_shape_rectangle
ps_shape_ellipse
ps_shape_diamond

parttype1 Gibt an welcher Partikeltyp geändert wird.

parttype2 Gibt an in welchen Partikeltyp geändert wird.

kind Die Art des changer. Es kann folgende Werte haben:
ps_change_motion Ändert nur den Bewegungsparameter des Partikels, nicht die Farbe und Form der Lebenszeiteinstellung
ps_change_shape Ändert nur den Formparamter wie Grösse und Farbe und Form
ps_change_all Ändert alle Parameter, das bedeutet, dass das Partikel zerstört wird und ein neuer

Typ erstellt wird.

Die folgenden Funktionen existieren um Changer zu definieren. Bedenke, dass jeder den Index des Partikelsystems bekommen welches als erstes Argument übergeben werden muss.

part_changer_create(ps) Erstellt einen neuen Changer im angegebenen Partikelsystem. Es gibt den Index des Typs zurück. Der Index muss in allen Aufrufen zum Setzen der Eigenschaften des Deflector angegeben werden.

part_changer_destroy(ps,ind) Zerstört den Changer ind im Partikelsystem. Wenn du sie nicht mehr brauchst, rufe das auf um Speicher zu sparen.

part_changer_destroy_all(ps) Zerstört alle Changers im Partikelsystem welche erstellt wurden.

part_changer_exists(ps,ind) Gibt zurück ob der angegebene Changer im Partikelsystem existiert.

part_changer_clear(ps,ind) Löscht den Changer ind auf seine Voreinstellung.

part_changer_region(ps,ind,xmin,xmax,ymin,ymax,shape) Setzt die Region des Changers.

part_changer_types(ps,ind,parttype1,parttype2) Setzt die Einstellung, welcher Partikeltyp der Changer in welchen anderen Typ umwandeln soll.

part_changer_kind(ps,ind,kind) Setzt die Art des Changers.

39.8 Ein Beispiel

Hier ist ein Beispiel eines Partikelsystems, das ein Feuerwerk erstellt. Das Feuerwerk benützt zwei Partikeltypen: eines das die Raketen gestaltet und eines das das eigentliche Feuerwerk gestaltet. Die Rakete generiert die Feuerwerkpartikel wenn sie explodiert. Wir generieren auch einen Emitter im Partikelsystem, der regelmässig Raktetenpartikel entlang des unteren Bildschirmrandes ausstößt. Dafür brauchen wir ein Objekt. Im creation event schreiben wir folgenden Code, der Partikeltypen, Partikelsystem und Emitter erstellt:

```
{
// erstelle das Partikelsystem
ps = part_system_create();
// die Feuerwerk-Partikel
pt1 = part_type_create();
part_type_speed(pt1,0,3,0,0);
part_type_direction(pt1,0,360,0,0);
part_type_color(pt1,c_red,c_yellow,c_black);
part_type_life(pt1,30,40);
part_type_gravity(pt1,0.2,270);


// Die Rakete
pt2 = part_type_create();
part_type_shape(pt2,pt_shape_disk);
part_type_size(pt2,4,4,0,0);
part_type_speed(pt2,10,14,0,0);
part_type_direction(pt2,80,100,0,0);
part_type_color(pt2,c_white,c_ltgray,c_gray);
part_type_life(pt2,30,60);
part_type_gravity(pt2,0.2,270);
part_type_death(pt2,100,pt1); // erstellt das Feuerwerk
beim Tod
```

```
// erstellt den emitter
em = part_emitter_create(ps);
part_emitter_region(ps,em,100,540,480,490,
ps_shape_rectangle,ps_distr_linear);
part_emitter_stream(ps,em,pt2,-4); // erstelle eines jeden
vierten Schritt
}
```

Im step event des Objektes müssen wir dem Partikelsystem mitteilen einen Schritt zu machen. Dafür brauchen wir folgenden Code:

```
{
part_system_doastep(ps);
}
```

Nun geben wir im draw event des Objektes folgenden Code zum Zeichnen des Partikelsystems an:

```
{
part_system_draw(ps,0,0);
}
```

40. Datenstrukturen

Diese Funktionalität ist nur mit einer registrierten Version des Game Maker verfügbar.

In Spielen musst du oftmals Informationen speichern. Beispielsweise musst du eine Liste mit Gegenständen, welche von einer Person getragen werden, speichern oder du willst die Orte/Stellen speichern, die immer noch besucht werden müssen. Du kannst "arrays" (Datenfelder) dafür verwenden. Wenn du aber kompliziertere Operationen machen willst, wie Daten sortieren oder nach einem bestimmten Begriff (kann auch Feld bedeuten /item/) suchen, musst du große Teile in GML-Programmcode schreiben, der in der Ausführung sehr langsam sein kann.

Um dies zu vermeiden, gibts beim Game Maker eine Anzahl von eingebauten Datenstrukturen, auf die mittels Funktionen zugegriffen werden kann. Momentan sind fünf (5) verschiedene Datenstrukturen bereitgestellt: stacks (Stapel), queues (Schlangen/Reihen), lists (Listen), maps (Abbildungen) und priority queues (vorrangige Schlangen). Jede dieser Datenstrukturen ist für einen bestimmten Einsatzzweck optimiert (siehe weiter unten).

Alle Datenstrukturen funktionieren (zusammenfassend betrachtet) nach dem gleichen Prinzip. Du kannst eine Datenstruktur mit einer Funktion erstellen, welche die ID der Datenstruktur zurückmeldet. Du benutzt diese ID, um Operationen auf die Datenstruktur anzuwenden. Wenn alles erledigt ist, vernichtest du die Datenstruktur wieder, um Speicherplatz zu sparen. Du kannst soviele

Datenstrukturen gleichzeitig verwenden, wie du willst. Sämtliche Strukturen speichern sowohl strings (Zeichenketten), als auch real values (reelle Werte/Zahlen).

Beim Vergleichen von Werten, beispielsweise beim Suchen in einer map oder beim Sortieren einer Liste, muss der Game Maker entscheiden, wann zwei Werte gleich sind. Für strings und integer (ganzzahlige) Werte ist es ja klar, aber bei reellen Werten , verursacht durch Rundungsfehler, können gleiche Zahlen ganz leicht ungleich werden.
Ein Beispiel: (5/3)*3 ist nicht gleich 5. Um dies zu vermeiden, wird eine Genauigkeit verwendet. Wenn der Unterschied zwischen zwei Zahlen kleiner als die Genauigkeit ist, werden sie als gleich betrachtet. Voreingestellt ist eine Genauigkeit von 0.0000001. Du kannst diese Genauigkeit ändern mittels dieser Funktion:

ds_set_precision(prec) Gibt die Genauigkeit bei Vergleichen an.

Diese Genauigkeit wird in allen Datenstrukturen angewandt aber nicht in anderen Vergleichen in GML!

40.1 Stacks - Stapel

Eine Stapeldatenstruktur ist ein sogenannter LIFO (Last-In-First-Out was soviel heisst wie: zuletzt rein - zuerst raus). Du kannst Werte auf den Stapel schieben und sie wieder zurückbewegen, durch Ziehen vom Stapel. Der Wert, welcher zuletzt auf den Stapel gelegt wurde, ist derjenige, der zuerst wieder zurückgenommen wird. Stacks werden oft benutzt, um interrupts zu bearbeiten oder wenn man rekursive Funktionen verwendet. Folgende Funktionen gibt es für stacks:

ds_stack_create() Erstellt einen neuen stack. Diese Funktion gibt eine ganze Zahl als ID wieder, welche fortan in allen anderen Funktionen verwendet werden muss, um auf diesen stack zuzugreifen. Du kannst mehrfache stacks erzeugen.

ds_stack_destroy(id) Vernichtet den stack mit der angegebenen ID und gibt den Speicher frei. Versäume nicht diese Funktion aufzurufen, wenn du die Struktur nicht mehr benötigst.

ds_stack_clear(id) Löscht den stack mit der angegebenen ID, entfernt alle Daten vom stack, ohne ihn zu vernichten.

ds_stack_size(id) Gibt die Anzahl der im stack gelagerten Werte wieder.

ds_stack_empty(id) Gibt an, ob der stack leer ist. Es entspricht dem Testen auf Grösse gleich Null.

ds_stack_push(id,val) Schiebt einen Wert auf den stack.

ds_stack_pop(id) Gibt den obersten stack Wert zurück und entfernt diesen vom stack.

ds_stack_top(id) Gibt nur den obersten stack Wert an - ohne ihn zu entfernen.

40.2 Queues - Schlangen

Ein queue ist sowas ähnliches, wie ein stack nur das nach dem FIFO-Prinzip (First-In-First-Out was soviel heisst, wie zuerst rein - zuerst raus) arbeitet. Der Wert, der zuerst in den queue gestellt wird, ist auch der erste, der wieder zurückgegeben wird. Es funktioniert ähnlich einer Warteschlange an einer Kasse; die Person, welche als erstes in der Schlange steht, wird zuerst bedient. Queues werden normalerweise eingesetzt, um Dinge zu speichern, die auf ihre Abarbeitung warten aber es gibt viele andere Gebrauchsformen. Nachstehende Funktionen existieren (beachte, dass die ersten fünf (5)

gleich den Funktionen beim stack sind; alle Datenstrukturen besitzen diese fünf Funktionen.)

ds_queue_create() Erstellt einen neuen queue. Die Funktion liefert eine ganze Zahl als ID, die fortan verwendet werden muss, wenn man aus anderen Funktionen auf diese Struktur zugreifen möchte. Du kannst mehrfache queues erstellen.

ds_queue_destroy(id) Vernichtet den queue mit der angegebenen ID und gibt belegten Speicher frei. Vegiss nicht diese Funktion aufzurufen, wenn du die Struktur nicht mehr benötigst.

ds_queue_clear(id) Löscht den queue mit der angegebenen ID, alle Daten werden entfernt aber die Struktur bleibt erhalten.

ds_queue_size(id) Gibt die Anzahl der im queue gespeicherten Werte an.

ds_queue_empty(id) Gibt an, ob der queue leer ist. Es entspricht dem Testen auf Grösse gleich Null.

ds_queue_enqueue(id,val) Gibt den Wert in den queue.

ds_queue_dequeue(id) Liefert den Wert, der schon am längsten im queue verweilt und entfernt diesen aus dem queue.

ds_queue_head(id) Liefert den Wert am Kopf der Schlange, das ist der Wert, der am längsten im queue ist. (Er wird nicht aus dem queue entfernt.)

ds_queue_tail(id) Liefert den Wert am Ende der Schlange, das ist der zuletzt hinzugefügte Wert. (Er wird nicht entfernt aus dem queue.)

40.3 Lists - Listen

Eine Liste speichert eine Sammlung von Werten in einer bestimmten Reihenfolge. Du kannst Werte am Ende der Liste anhängen oder sie irgendwo mittig in die Liste setzen. Du kannst die Werte addressieren, indem du einen index (Zeiger) verwendest. Ferner kannst du die Listenelemente sortieren - sowohl in absteigender, als auch in aufsteigender Reihenfolge. Listen können vielfältig verwendet werden, beispielsweise um wechselnde Sammlungen von Werten zu speichern. Sie werden durch einfache arrays (Felder) realisiert, die aber mit kompiliertem Programmcode ausgeführt werden, was beträchtlich schneller ist, als wenn du selber ein array machst. Folgende Funktionen sind verfügbar:

ds_list_create() Erzeugt eine neue Liste. Die Funktion liefert eine ganze Zahl als ID, die fortan verwendet werden muss, wenn man aus anderen Funktionen auf diese list zugreifen möchte. Du kannst mehrfache lists erstellen.

ds_list_destroy(id) Vernichtet die list mit der angegebenen ID und gibt den Speicher frei. Vegiss nicht diese Funktion aufzurufen, wenn du die Struktur nicht mehr benötigst.

ds_list_clear(id) Löscht die list mit der angegebenen ID, alle Daten werden entfernt aber die Struktur bleibt erhalten.

ds_list_size(id) Gibt die Anzahl der in der list gespeicherten Werte an.

ds_list_empty(id) Gibt an, ob die list leer ist. Es entspricht dem Testen auf Grösse gleich Null.

ds_list_add(id,val) Fügt den Wert ans Ende der list.

ds_list_insert(id,pos,val) Fügt einen Wert an Position pos in die list. Die erste Position ist 0 (Null), die letzte entspricht der um eins verringerten Grösse (size) der Liste.

ds_list_replace(id,pos,val) Ersetzt den Wert an Position pos innerhalb der list mit dem neuen Wert (val).

ds_list_delete(id,pos) Löscht den Wert an Position pos aus der list. (Position 0 ist das erste Element.)

ds_list_find_index(id,val) Gibt die Position an, an der der Wert (val) in der list zu finden ist. Ist der Wert nicht enthalten, wird -1 zurückgegeben.

ds_list_find_value(id,pos) Gibt den Wert an, der an Position pos in der list gespeichert ist.

ds_list_sort(id,ascend) Sortiert die Werte der list. Wenn ascend (aufsteigend) true (wahr) ist, werden die Werte aufsteigend sortiert - sonst in absteigender Reihenfolge.

40.4 Maps - Abbildungen

In ziemlich vielen Situationen muss man Paare speichern, die aus einem key (Schlüssel) und einem value (Wert) bestehen. Zum Beispiel kann ein Charakter eine Anzahl verschiedener Gegenstände besitzen und von jedem dieser Gegenstände eine bestimmte Anzahl. In so einem Fall ist der Gegenstand der key und die jeweilige Anzahl der value. Maps unterstützen solche Paare, sortiert nach Schlüsseln. Du kannst Paare hinzufügen und nach dem entsprechenden Wert von bestimmten Schlüsseln suchen. Weil alle Schlüssel sortiert sind, kannst Du auch den vorherigen und den nächsten Schlüssel finden. Manchmal ist es auch sinnvoll eine map zu verwenden, um nur Schlüssel abzuspeichern - ohne entsprechende Werte. Verwende in so einem Fall einen Wert von 0 (Null). Die nachstehenden Funktionen gibt es:

ds_map_create() Erstellt eine neue map. Die Funktion liefert eine ganze Zahl als ID, die fortan verwendet werden muss, wenn man aus anderen Funktionen auf diese map zugreifen möchte.

ds_map_destroy(id) Vernichtet die map mit der angegebenen ID und gibt den Speicher frei. Vergiss nicht diese Funktion aufzurufen, wenn du die Struktur nicht mehr benötigst.

ds_map_clear(id) Löscht die map mit der angegebenen ID, alle Daten werden entfernt aber die Struktur bleibt erhalten.

ds_map_size(id) Gibt die Anzahl der key-value Paare an, die in der map gespeichert sind.

ds_map_empty(id) Gibt an, ob die map leer ist. Es entspricht dem Testen auf Grösse gleich Null.

ds_map_add(id,key,val) Fügt ein key-value Paar der map hinzu.

ds_map_replace(id,key,val) Ersetzt den dem key entsprechenden Wert durch einen neuen Wert.

ds_map_delete(id,key) Löscht den key und den entsprechenden value aus der map. (Wenn mehrfache Einträge mit demselben key vorhanden sind, wird nur einer entfernt.)

ds_map_exists(id,key) Gibt an, ob der key in der map enhalten ist.

ds_map_find_value(id,key) Liefert den Wert des entsprechenden key.

ds_map_find_previous(id,key) Gibt den grössten key der map an, der kleiner als der angegebene ist. (Beachte, dass der key zurückgegeben wird, nicht der value. Du kannst die vorherige Routine verwenden, um an den value zu gelangen.)

ds_map_find_next(id,key) Liefert den kleinsten key aus der map, der grösser als der angegebene key ist.

ds_map_find_first(id) Liefert den kleinsten key der map.

ds_map_find_last(id) Gibt den grössten key der map an.

40.5 Priority queues - Vorrangige Schlangen

In einer vorrangigen Schlange werden Werte gespeichert, jeder mit einer priority (Vorrangigkeit). Du kannst schnell die Werte mit der höchsten und niedrigsten priority ermitteln. Durch das Verwenden dieser Datenstruktur kannst du bestimmte Sachen nach ihrer Dringlichkeit abarbeiten. Diese Funktionen stehen bereit:

ds_priority_create() Erzeugt eine neue priority-queue. Die Funktion liefert eine ganze Zahl als ID, die fortan verwendet werden muss, wenn man aus anderen Funktionen auf diese priority-queue zugreifen möchte.

ds_priority_destroy(id) Vernichtet die priority-queue mit der angegebenen ID und gibt den Speicher frei. Vegiss nicht diese Funktion aufzurufen, wenn du die Struktur nicht mehr benötigst.

ds_priority_clear(id) Löscht die priority-queue mit der angegebenen ID, alle Daten werden entfernt aber die Struktur bleibt erhalten.

ds_priority_size(id) Gibt die Anzahl der in der priority-queue gespeicherten Werte an.

ds_priority_empty(id) Gibt an, ob die priority-queue leer ist. Es entspricht dem Testen auf Grösse gleich Null.

ds_priority_add(id,val,prio) Fügt den Wert mit der angegebenen priority in die priority-queue.
ds_priority_change_priority(id,val,prio) Ändert die priority des angegebenen value ind der priority-queue.
ds_priority_find_priority(id,val) Liefert die priority des angegebenen value in der priority-queue.

ds_priority_delete_value(id,val) Löscht den value (und seine priority) aus der priority-queue.

ds_priority_delete_min(id) Liefert den value mit der kleinsten priority und entfernt ihn aus der priority-queue.

ds_priority_find_min(id) Wie oben, nur wird der Wert nicht aus der priority-queue entfernt.

ds_priority_delete_max(id) Liefert den value mit der grössten priority und entfernt ihn aus der priority-queue.

ds_priority_find_max(id) Wie oben, nur wird der Wert nicht aus der priority-queue entfernt.

41. Mehrspieler Spiele

Diese Funktionen sind nur in der registrierten Version vom Game Maker verfügbar.

Gegen den Computer zu spielen macht Spass. Aber gegen andere menschliche Spieler zu spielen kann noch mehr Spass machen. Es ist ausserdem relativ einfach, solche Spiele zu erstellen, weil du keine komplizierte künstliche Intelligenz für den virtuellen Gegner erstellen musst. Du kannst natürlich mit 2 Spielern vor einem Monitor hocken und verschiedene Tasten bzw. Eingabegeräte benutzen, aber es ist viel interessanter, wenn jeder vor seinem eigenen PC sitzt. Game Maker unterstützt Multiplayer-Spiele. Beachte aber, dass es schwierig ist, gut synchronisierte stabile Netzspiele zu entwickeln. Dieses Kapitel gibt einen kurzen Überblick über die Netzwerk-Möglichkeiten des Game Maker. Auf der Internetseite gibt es ein eigenes Tutorial zu diesem Thema mit mehr Informationen.

41.1 Eine Verbindung aufbauen

Wenn mehrere PCs miteinander kommunizieren sollen, müssen sie sich auf ein Kommunikationsprotokoll (sowas wie eine gemeinsame Sprache) einigen. Wie die meisten Spiele bietet auch der Game Maker vier verschiedene Verbindungstypen: IPX, TCP/IP, Modem, serielle Verbindung.
Diese benutzen jeweils eigene Protokolle. IPX arbeitet fast vollständig transparent, und kann in lokalen Netzwerken verwendet werden. Es muss installiert werden, bevor man es benutzt. TCP/IP ist das Internet-Protokoll, wird aber auch im Netzwerk verwendet. Es kann benutzt werden, um mit Freunden übers Internet zu spielen, vorausgesetzt man kennt die IP-Adresse. In LANs brauchst du keine Adressen bereithalten. Modem-Verbindungen werden mit einem Modem hergestellt. Du musst einige Modem-Parameter bereithalten (einen Initialisierungsstring und die Telefonnummer), um es zu verwenden. Für die serielle Verbindung brauchst du einige Anschlussparameter. Um diese Verbindungstypen zu initialisieren, gibt es verschiedene Funktionen im Game Maker:

mplay_init_ipx() Initialisiert eine IPX-Verbindung.

mplay_init_tcpip(addr) Initialisiert eine TCP/IP Verbindung. addr ist ein string, welcher die Internet- oder IP-Adresse enthält, z.B. 'www.gameplay.com' oder '123.123.123.12', möglicherweise gefolgt von einer Port Nummer (z.B. ':12'). Nur wenn man eine Verbindung aufbaut (siehe unten) braucht man eine Addresse. Im LAN sind sie nicht nötig.

mplay_init_modem(initstr,phonenr) Initialisiert eine Modem-Verbindung. initstr ist der Initialisierungstring fur das Modem (kann leer sein). phonenr ist ein string der die anzurufende Telefon-Nummer enthält. (z.B. '0201234567'). Nur für den Verbindungsaufbau wird die Nummer benötigt (siehe unten).

mplay_init_serial(portno,baudrate,stopbits,parity,flow) Initialisiert eine serielle Verbindung (Com). portno ist die Port Nummer (1-4). baudrate ist die zu benutzende Baudrate (100-256K). stopbits zeigt die Anzahl der stopbits (0 = 1 bit, 1 = 1.5 bit, 2 = 2 bits). parity gibt die Parität an (0=keine, 1=ungerade, 2=gerade, 3=Markierung). Und flow gibt die Datenflusskontrolle an (0=keine, 1=xon/xoff, 2=rts, 3=dtr, 4=rts und dtr).

Ein typischer Aufruf wäre: mplay_init_serial(1,57600,0,0,4)
Gib 0 als erstes Argument an, um ein Dialogfenster zu öffnen, in dem der Benutzer Einstellungen ändern kann.

Dein Spiel sollte eine dieser Funktionen exakt einmal aufrufen. Alle Funktionen geben zurück, ob

sie erfolgreich waren. Sie sind nicht erfolgreich, wenn die Protokolle nicht installiert bzw. unterstützt sind. Um zu überprüfen, ob eine Verbindung verfügbar ist, verwende folgende Funktion:

mplay_connect_status() Gibt den Status der momentanen Verbindung an.
0 = keine Verbindung, 1 = IPX Verbindung, 2 = TCP/IP Verbindung, 3 = Modem-Verbindung, 4 = serielle Schnittstelle.

Um die Verbindung abzubrechen, nutze folgende Funktion:

mplay_end() beendet die momentane Verbindung.

Bei einer TCP/IP-Verbindung willst du vielleicht einem potentiellen Mitspieler deine IP-Adresse mitteilen. Folgende Funktion hilft dir dabei:

mplay_ipaddress() Gibt die IP-Adresse des eigenen PCs an (z.B. '123.123.123.12') als string. Kann man auf dem Bildschirm anzeigen lassen.
Anmerkung: Diese Routine ist langsam - ruf sie nicht permanent auf!

41.2 Erstellen und Teilnehmen an Sessions

Wenn du mit einem Netzwerk verbunden bist, können mehrere Spiele in diesem Netzwerk stattfinden. Hier werden sie "Sessions" (Sitzungen) genannt. Diese unterschiedlichen "Sessions" können verschiedenen oder einem Spiel entsprechen. Ein Spiel muss sich eindeutig im Netzwerk identifizieren. Glücklicherweise macht der Game Maker das für dich. Das einzige, was du wissen musst ist: Wenn du die "Game-ID" änderst, ändert sich auch diese Identifikation hier. Auf diese Weise kannst du verhindern, dass Leute mit alten Version gegen welche mit neuen Versionen spielen.
Wenn du ein Mehrspieler-Spiel beginnen willst, musst du erst ein neue "Session" erstellen, nutze diese Funktion:

mplay_session_create(sesname,playnumb,playername) Erstellt eine neue "Session" auf der momentanen Verbindung. sesname ist ein string, der den Namen der "session" angibt. playnumb ist die maximale Spieleranzahl in diesem Spiel (0=beliebig). playname dein Spielername. Mit Erfolgsmeldung.

Eine Instanz des Spiels muss die Session erstellen. Die anderen Instanzen sollten dieser Session beitreten. Das ist etwas schwieriger. Du musst zuerst schauen, welche Sessions verfügbar sind und dann an einer teilnehmen. Drei wichtige Routinen gibt es dafür:

mplay_session_find() Sucht Sessions ab, die noch Spieler akzeptieren und gibt die Anzahl der gefundenen zurück.

mplay_session_name(zahl) Gibt den Namen von Session zahl an (0 ist die erste session). Diese Funktion kann nur nach der voranstehenden aufgerufen werden.

mplay_session_join(zahl,playername) Teilnehmen an Session zahl (0 ist die erste Session). playername ist der von dir gewählte Spielername. Mir Erfolgsmeldung.

Es gibt eine weitere Funktion, die den Session-Modus ändert. Sie sollte vor dem Erstellen einer Session ausgeführt werden:

mplay_session_mode(move) Bestimmt, ob ein anderer Rechner die Session übernimmt, wenn der

momentane Server ausfällt. move sollte entweder "true"(wahr) oder false(falsch) (false ist voreingestellt).

Um den Verbindungsstatus zu prüfen verwende:

mplay_session_status() Gibt den Status der momentanen Session an.
0 = keine Session, 1 = erstellte Session, 2 = an Session teilgenommen.

Ein Spieler kann folgende Routine nutzen, um eine Session zu stoppen:

mplay_session_end() beendet die Session für diesen Spieler.

41.3 Spieler

Jede Instanz, die an einer "session" teilnimmt ist ein Spieler. Spieler haben Namen. 3 Funktionen befassen sich damit:

mplay_player_find() Gibt die Anzahl der Spieler innerhalb der aktuellen Session an

mplay_player_name(numb) Gibt den Namen vom Spieler numb an (numb=0 ist der erste Spieler, immer du selbst). Diese Funktion kann nur nach voranstehender Funktion ausgeführt werden.

mplay_player_id(numb) Gibt die ID von Spieler numb an (numb=0 ist der erste Spieler, du selbst). Kann nur ausgeführt werden, wenn die oberste Funktion aufgerufen wurde (vorher). Diese id wird benutzt, um Nachrichten von und zu einzelnen Spielern zu schicken.

41.4 Gemeinsame Daten

Gemeinsame Daten sind wahrscheinlich der einfachste Weg ein Spiel zu synchronisieren. Die gesamte Kommunikation wird dir vom Leib gehalten. Es gibt einen Satz von 1000000 Werten, welcher eine gemeinsame Basis für alle ist. Jedes Spiel kann jeden Wert lesen und ändern. Game Maker stellt sicher, dass jedes Spiel dieselben Werte sieht. Werte können reelle Zahlen oder Zeichenketten sein. Es gibt nur 2 Funktionen:

mplay_data_write(ind,val) Schreibt den Wert val (string oder real) in Postion ind (ind zwischen 0 und 10000).

mplay_data_read(ind) Gibt den Wert von Position ind wieder (ind zwischen 0 und 10000) zurück. Anfangs sind alle Werte auf 0.

Um die Daten auf den verschiedenen Rechnern zu synchronisieren kannst du entweder einen verlässlichen Modus verwenden, der sicherstellt, das die Änderungen auch auf dem anderen Rechner ankommen (aber es ist langsam) oder nicht verlässlich. Um das zu ändern, benutze:

mplay_data_mode(guar) Legt die verlässliche bzw. unsichere Übertragungsmethode für gemeinsame Daten fest. guar sollte true(wahr) bzw. false(falsch) sein. true ist vorgegeben.

41.5 Messages (Nachrichten)

Der zweite Kommunikationsmechanismus, den Game Maker unterstützt, ist das Senden und Empfangen von Nachrichten. Ein Spieler kann eine Nachricht an alle Mitspieler senden, oder an einzelne. Spieler bemerken eingehende Nachrichten und können entsprechend reagieren. Es gibt wie

oben ein sicheren und einen unsicheren Weg, mit den selben Vor- und Nachteilen.
Diese Funktionen gibt es dafür:

mplay_message_send(player,id,val) Sendet eine Nachricht an player (id oder Name; 0 sendet Nachricht an alle). id ist eine ganzzahliger Wert, der die Nachricht identifiziert ("message identifier") und val ist der Wert (real oder string). Die Nachricht wird im unsicheren Modus gesendet.

mplay_message_send_guaranteed(player,id,val) Sendet eine Nachricht an player. Diesmal im sicheren Modus.

mplay_message_receive(player) Empfängt nächste Nachricht aus der Warteschlange, welche vom angegebenen player kam (id oder Name). Nutze 0 für Nachrichten von jedem Spieler. Die Funktion gibt wieder, ob tatsächlich eine neue Nachricht da ist. Wenn dem so ist, geht es hiermit an den Inhalt:

mplay_message_id() Gibt den "message identifier" der letzten empfangenen Nachricht zurück

mplay_message_value() Gibt den Wert(Inhalt) der letzten empfangenen Nachricht zurück.

mplay_message_player() Gibt den Spieler, welcher die letzte empfangene Nachricht versendet hat, an.
mplay_message_name() Gibt den Namen des Spielers an, der die letzte empfangene Nachricht versandt hat an.

mplay_message_count(player) Gibt die Anzahl verbleibender Nachrichten von player in der Warteschlange an (0 um alle zu zählen).

mplay_message_clear(player) Löscht alle verbleibenden Nachrichten von player aus der Warteschlange (0 für alle Nachrichten).

Ein paar Anmerkungen hier: Zuerst, wenn du eine Nachricht an einen Spieler senden willst, musst du seine player-id kennen. Wie schon erwähnt bekommst du sie mit: mplay_player_id(). Diese player-id wird auch benutzt, um Nachrichten von einzelnen Spielern zu empfangen.

Alternativ kannst du den Spielernamen des Spielers angeben (als String). Wenn mehrere Spieler den gleichen Namen haben, bekommt nur der erste die Nachricht. Zweitens wunderst du dich vielleicht, dass jede Nachricht eine ganzzahlige Nachrichten-id hat. Der Grund ist, dass es hilft, unterschiedliche Arten von Nachrichten zu versenden. Der Empfänger kann die Art der Nachricht anhand der id prüfen und entsprechend reagieren. (Weil Nachrichten nicht sicher ankommen, ist es problematisch, wenn id und Wert nicht mit derselben Nachricht gesendet werden.)

42. DLLs benutzen

Diese Funktionen sind nur in der registrierten Version vom Game Maker verfügbar.

In Fällen, in denen dir die Funktionalität von GML nicht genug ist, kannst du die Möglichkeiten nutzen sie mit Plug-Ins erweitern. Ein Plug-In hat die Form einer DLL ("Dynamic Link Library"). Es kann mit allen Programmiersprachen, die DLLs unterstützen, programmiert werden. (z.B. C/C++, Delphi usw.) In einer solchen DLL kann man Funktionen definieren. Diese Funktionen können 0 bis 11 Parameter besitzen, die entweder eine reelle Zahl (real in Delphi, double in C/C++) oder einen null-terminierten String (PChar in Delphi, char* bzw. char[] in C/C++) als Typ sind. Bei mehr als 4 Argumenten müssen alle reelle Zahlen sein. Die Funktionen können entweder eine reelle Zahl oder einen String zurückgeben.
Ein Beispiel für eine mit Delphi erstellte DLL: (Bedenke dass dies kein GML Code ist!)

```
-------------------------------------------------------
library MyDLL;
uses SysUtils, Classes;

function MyMin(x,y:double):double; cdecl;
begin
if x<y then Result := x else Result := y;
end;

var res : array[0..1024] of char;

function DoubleString(str:PChar):PChar; cdecl;
begin
StrCopy(res,str);
StrCat(res,str);
Result := res;
end;

exports MyMin, DoubleString;
begin
end.

-------------------------------------------------------
Die entsprechende C/C++-Variante:
-------------------------------------------------------
#include <string.h>
#ifdef __cplusplus
//Exportmakro für C++
#define EXPORT extern "C" __declspec(dllexport)
#else
//Exportmakro für C
#define EXPORT __declspec(dllexport)
#endif

EXPORT double MyMin (double x, double y)
{
```

```
return (x<y) ? x : y;
}

char result[1024];

EXPORT char* DoubleString (char* str)
{
strcpy(result,str);
strcat(result,str);
return result;
}
```

Diese DLLs definieren zwei Funktionen: MyMin gibt die kleine von zwei reellen Zahlen zurück und DoubleString hängt einen String an sich selbst an. Beachte, dass du vorsichtig mit dem Speicher umgehen musst. (Darum ist der Ergebnis-String global). Ausserdem wird hier die cdecl-Aufrufkonvention verwendet. Du kannst aber auch stdcall verwenden. Wenn du den Code kompilierst, bekommst du eine Datei MyDLL.dll. Diese Datei muss sich im Arbeitsverzeichnis des Spiels befinden (oder in einem anderen Verzeichnis, wo Windows sie findet). Du kannst sie auch als Daten-Datei benutzen. Um diese DLL im Game Maker zu benutzen, musst du erst die externen Funktionen definieren.
Dafür gibt es die folgende GML-Funktion:

external_define(dll, function, calltype, resulttype, argnumb,arg1type, arg2type, ...)
Definiert eine externe Funktion. dll ist der Name der DLL-Datei, function ist der name der Funktion, calltype ist die Aufrufkonvention (entweder dll_cdecl oder dll_stdcall). resulttype ist der Typ des Rückgabewerts (entweder ty_real oder ty_string). argnumb gibt die Anzahl der Argumente an. Ausserdem müssen die Typen der Argumente angegeben werden. (wieder ty_real oder ty_string). Sind es mehr als 4 Argumente, müssen alle ty_real sein. Die Funktionen gibt die ID der externen Funktion und muss für das Aufrufen verwendete werden.
Also muss beim Start des Spieles folgender Code ausgeführt werden:

```
{
global.mymin = external_define('MYOWN.DLL','MyMin',dll_cdecl,ty_real,2,ty_real,ty_real);
global.doublestring =
external_define('MYOWN.DLL','DoubleString',dll_cdecl,ty_string,1,ty_string);
}
```

Aufgerufen werden die Funktionen dann mit der Funktion:

external_call(id, arg1, arg2,...) Ruft die Funktion mit der ID und den gegebenen Argumenten auf. Du musst die richtige Anzahl von Argumenten und den richtigen Typen (real oder string) angeben. Die Funktion gibt den Rückgabewert der externen Funktion zurück.
Zum Beispiel:

```
{
aaa = external_call(global.mymin,x,y);
sss =
external_call(global.doublestring,'Hello');
}
```

Wenn du die DLL nicht mehr brauchst, gib sie besser wieder frei.

external_free(dll) Gibt die DLL mit dem angegeben Namen wieder frei. Das ist wirklich notwendig wenn das Spiel die DLL entfernen muss (z. B. weil es eine Daten-Datei ist). Solange die DLL nicht freigegeben wird, kann sie nicht entfernt werden. Am besten mache das im "end of game Event".

Du wirst dich vielleicht wundern, wie du eine DLL-Funktion schreiben kannst, die etwas im Spiel macht. Der einfachste Weg ist, dass die Funktion GML-Code als String zurückgibt. Mit der folgenden Funktion kann man diesen dann ausführen:

execute_string(code) Führt den String als GML-Code aus.
Alternativ kann die DLL-Funktion eine Datei mit GML-Code erstellen und den Dateinamen zurückgeben. Dann verwendet man diese Funktion:

execute_file(filename) Führt die Datei als GML-Code aus.

Ein Beispiel:

```
{
ccc =
external_call(global.ddd,x,y);
execute_string(ccc);
}
```

In einigen Fällen benötigt die DLL den Handle für das Spielfenster. Der Handle kann mit folgender Funktion ermittelt werden:

window_handle() Gibt den Handle für das Spielfenster zurück.

DLLs können nicht im Secure-Mode verwendet werden.
Externe DLLs zu verwenden, ist eine mächtige Funktion welche auch missbraucht werden kann. Bitte verwende sie aber nur, wenn du weisst, was du tust.

Das Team

Koordination, Layout, Anpassungen an Version 5.1 & 5.2 & 5.3, finales korrigieren: Windapple & Augenzeuge
Gesammelt, coloriert und bebildert: Windapple
Übersetzungen: Freeman, Augenzeuge, Cygnus, cipher3000, Agnahim, Windapple

Schlussanmerkungen in eigener Sache

Übersetzt mit der Hilfe des Online-Wörterbuches von der TU-Chemnitz:
http://dict.tu-chemnitz.de/

Übersetzt von den Mitgliedern von www.gmaker.de, das Game Maker Headquarter:

Danke an alle, die diese Dokumentation möglich gemacht haben!

Versionshistorie:
5.3
1.0 Erstes Release (Juni 2004)

5.2
1.0.1 Seitennummerierung hinzugefügt, falsch gesetzte Tabulatoren entfernt, mit neuem GhostScript geparst (Februar 2004)
1.0 Erstes Release (Februar 2004)

5.0
0.0 Erste Übersetzte Abschnitte (~September 2003)

In dieser Übersetzung stecken zig dutzende Arbeitsstunden, bitte behandelt sie pfleglich™.

Diese Dokumentation enhält: 62857 Wörter.