

# **Entwickeln von PC-Spielen**

**basierend auf der Software  
Game Maker Version 7.0  
von Prof. Overmars**

Wissenschaftlicher Bericht  
Philipp Nägelsbach  
Matrikelnummer: 16423  
Studiengang Audiovisuelle Medien Bachelor  
Oktober 2007

## Inhaltsverzeichnis

1.1 Einleitung	3
1.2 Game Maker: Geschichte	3
1.3 Game Maker: Überblick	4
1.3.1 Objects	4
1.3.2 Events	4
1.3.3 Actions	5
1.3.4 Game Maker Language	5
2. Ein Lernprojekt	6
2.1 Sprites	6
2.2 Backgrounds	8
2.3 Objects und Scripts	8
2.4 Rooms	14
2.5 Abschluss des Projektes	14
3. (Bei-)Spiele	16
3.1 Ninja ♥ Pirate	16
3.2 Coaster Rider	16
3.3 Circular Level	17
4. Fazit	17
5. Quellenangaben	17

## 1.1 Einleitung

*„...there is a world of difference between having a great idea for a game and being a great game designer.“ Phil Wilson<sup>1</sup>*

Wer Game Design lernen will, steht vor einer schwierigen Aufgabe. Kreativität und Phantasie sind zwar Grundvoraussetzung, die Umsetzung der Idee erfordert jedoch viel technisches Wissen. Die heutigen Computerspiele werden in großen Teams entwickelt, bestehend aus Produzenten, Programmierern, Grafikern, Musikern, Autoren, Betatestern sowie aus Sound-, Level- und Gamedesignern. Wer alleine ein eigenes Computerspiel entwickeln will, muss also alle diese Bereiche entweder selbst abdecken oder Teammitglieder finden. Besonders die Programmierung eines Spiels stellt eine große Herausforderung dar, die nicht jeder Anfänger meistern kann. Um den Einstieg in die Spieleentwicklung zu erleichtern, gibt es jedoch einige Programme, die es dem Nutzer erlauben, Spiele ohne Programmierkenntnisse zu erstellen. Eines dieser Programme, ist der Game Maker, der hier vorgestellt werden soll. Der Game Maker zeichnet sich dadurch aus, dass er sowohl einsteigerfreundlich, als auch äußerst mächtig ist und damit ideal dafür geeignet ist, sowohl die Grundlagen der Spieleentwicklung zu erlernen, als auch professionelle Spiele zu entwickeln. Ich werde im Folgenden die Funktionsweise des Game Makers anhand eines Lernprojektes erklären und aufzeigen, welche Vor- und Nachteile die Arbeit mit diesem Programm bringt.

## 1.2 Game Maker: Geschichte

*„People often asked me what computer games I played when I was young. The answer is simple: none.“ Mark Overmars<sup>2</sup>*

Der Game Maker entstand im Jahr 1999 als Hobbyprojekt von Mark Overmars, Professor am *Institute of Information and Computing Sciences* der *Utrecht University* in den Niederlanden<sup>3</sup>. Zu Anfang hieß der Game Maker noch Animo, da die ursprüngliche Idee ein Programm zur Animation von 2D-Grafiken war. Doch die Sache wuchs schnell und so verfügte die erste Version des Game Makers bereits über eine simple Scriptsprache und die Möglichkeit, die Spiele direkt im Editorfenster zu spielen (eine selbstausführende **exe** Datei der Spiele konnte zu diesem Zeitpunkt noch nicht erstellt werden). Version 3.0 des in der Programmiersprache *Delphi* geschriebenen Programms führte 2001 erstmals *DirectX* als Schnittstelle für Grafik ein und der Game Maker erlangte immer mehr Bekanntheit im Internet; die Anzahl der Downloads stieg auf 270 000 (mehr als 6x so viele wie in den beiden Jahren zuvor). Professor Overmars setzte die Software selbst hauptsächlich in seinen eigenen Kursen über Game Design ein, inzwischen dient der Game Maker aber auch in vielen anderen Lehrveranstaltungen rund um die Welt als Unterrichtsmittel. Über die Jahre hinweg entwickelte Mark Overmars das Programm kontinuierlich weiter und fügte immer neue Features hinzu: Multiplayer, Partikeleffekte, Ressourcenverwaltung, Datenstrukturen, Wegfindung, Pfade, *Direct3D* Unterstützung und Extensions sind nur einige Dinge, die bis heute ihren Weg in das Programm gefunden haben. Durch die steigende Popularität (4000 Downloads pro Tag, überlastete Online-Foren) wurde ab Version 5.0 auch die Möglichkeit hinzugefügt, das Programm für einen kleinen Betrag (ca. 18€) zu registrieren. Spezielle Funktionen wie Multiplayer- und 3D-Unterstützung sowie Alpha-Transparenz und Rotation der *Sprites* können seither nur noch in der registrierten Version genutzt werden. Für die Abwicklung der Registrierung wurde 2004 die *Game Maker Company* gegründet, seit 2007 übernimmt die Firma *YoYo Games* den Verkauf und den Support. Das Ziel von *YoYo Games* ist es, eine zentrale Anlaufstelle für alle Nutzer des Game Makers zu schaffen. Die dazu erstellte Website **www**.

**yoyogames.com** bietet dazu die Möglichkeit, eigene Spiele hochzuladen, andere Spiele zu spielen und zu bewerten sowie Hilfe, Tutorials und ein Forum, in dem sich die Hobbyentwickler austauschen können. Über das Browser Plug-In *Instant Play* ist es außerdem möglich, die Spiele direkt aus dem Browser heraus zu starten, ohne das Spiel dauerhaft auf der Festplatte zu speichern. Momentan wird bei *YoYo Games* eine komplett neue Version entwickelt, die nicht mehr auf *Delphi* sondern auf C++ aufbaut, was vor allem die Geschwindigkeit verbessern soll. Allerdings sollen dadurch auch Portierungen auf andere Betriebssysteme und eventuell sogar Konsolen möglich werden<sup>4</sup>. Die aktuelle Version 7.0 läuft nur unter Windows (2000, XP und Vista).

## 1.3 Game Maker: Überblick

Der Game Maker ist eine Entwicklungsumgebung für PC-Spiele, was bedeutet, dass viele für Spiele relevante Funktionen bereits vorgefertigt zur Verfügung stehen. Neben Ressourcenverwaltung (*Sprites, Sounds, Backgrounds/Tiles, Fonts, Paths, Scripts, Time lines, Objects* und *Rooms*) bietet er Funktionen für Sound-/Musikwiedergabe, Grafikdarstellung, Partikeleffekte, Bewegung, Kollision, Wegfindung, Benutzereingabe, Speichern/Laden, Multiplayer, ini-Dateien sowie die Möglichkeit zusätzliche Funktionalität über dll Dateien einzubinden. Obwohl der Game Maker auf *Direct3D* aufbaut und auch nützliche 3D-Funktionen bietet, ist er primär für 2-dimensionale Spiele ausgelegt. Außerdem enthält er ein simples Grafikprogramm und einen Leveleditor.

### 1.3.1 Objects

Der Game Maker ist ein objektorientiertes Programm. Es können beliebige *Objects* definiert werden, deren Aussehen über lokale Variablen festgelegt wird. Von jedem *Object* können in den einzelnen *Rooms* (die Level des Spiels) Instanzen platziert werden. Ein Instanz besitzt zu Beginn alle Eigenschaften des *Objects*, die sich jedoch durch Interaktion mit dem Spieler oder anderen Spielelementen lokal für diese Instanz ändern können. Die wichtigsten Einstellungen sind das *Sprite* mit dem das *Object* im Spiel gezeichnet werden soll, die Maske für die Kollision, der *Depth*-Wert, der die Position in der Zeichenreihenfolge angibt (je niedriger die *depth*, desto weiter im Vordergrund befindet sich das *Object*) und das *Parent Object*, über das Vererbung ermöglicht wird.

### 1.3.2 Events

Die komplette Spiellogik des Game Makers basiert auf einem Ursache-Wirkungs-Prinzip, was bedeutet, dass sämtliche Aktionen des Spiels jeweils Reaktionen auf bestimmte Ereignisse in der Spielwelt sind. Für jedes *Object* können deshalb *Events* definiert werden, die als Auslöser für so genannte *Actions* dienen. Jedes *Event* wird zur Laufzeit des Spiels in jedem *Step* (Der Game Maker läuft standardmäßig mit 30 *Steps* pro Sekunde) abgefragt und je nach Ergebnis dieser Überprüfung werden die korrespondierenden *Actions* gestartet oder nicht.

Die Reihenfolge der Abfrage sieht folgendermaßen aus<sup>5</sup>:

- *Begin Step*
- *Alarm 0-11*
- *Keyboard and Mouse*

- *Keyboard and Mouse Press*
- *Keyboard and Mouse Release*
- *Step*
- *End of Path*
- *Outside Room*
- *Intersect Boundary*
- *Collision*
- *End Step*
- *Draw*
- *Animation End*

Die *Events Begin Step*, *Step* und *End Step* treten immer ein. *Actions*, die für diese *Events* definiert wurden, werden folglich jeden *Step* ausgeführt und verschlingen daher viel Prozessorleistung. Die genaue Bedeutung der einzelnen *Events* kann in der umfangreichen Hilfe nachgelesen werden, die vom deutschen Game Maker Forum [www.gm-d.de](http://www.gm-d.de) auch in die deutsche Sprache übersetzt wurde<sup>6</sup>.

### 1.3.3 Actions

Durch einfaches Drag&Drop können *Actions* (jeweils repräsentiert durch ein Icon) aus einem seitlichen Menü ausgewählt und einem *Event* zugewiesen werden. Die vorgegebenen *Actions* sind in verschiedene Bereiche gegliedert und beinhalten unter anderem Funktionen zur Bewegung von *Instanzen*, Änderung der *Sprites*, Abspielen von *Sounds* und Wechseln des *Rooms*. Die beiden mächtigsten *Actions* sind jedoch *Execute a piece of code* und *Execute a script*, die sich beide im Tab *control* befinden, da sie es ermöglichen eigenen Code in der internen Scriptsprache des Game Makers (GML) zu schreiben.

### 1.3.4 Game Maker Language

Die interne Scriptsprache Game Maker Language (GML) erlaubt den Zugriff auf alle Funktionen und Variablen des Game Makers und ist relativ leicht zu erlernen. Die Syntax orientiert sich an höheren Programmiersprachen, wie C, Pascal oder Java, bietet jedoch enorm viele Freiheiten (zum Beispiel sind Semikolons freiwillig und können auch weggelassen werden). GML beherrscht dynamische Typisierung, d.h. je nach Zuweisung ändert sich der Datentyp einer Variable. Der Game Maker arbeitet intern mit Fließkommazahlen und Strings. Logische Operationen werden ebenfalls mit Fließkommazahlen ausgeführt, die Zahl 1 entspricht dabei true, 0 entspricht false.

Beispiel:

```
a = 0;
a = !a; // 0 entspricht false -> a = !false -> a = true
b = true;
if (a && b) { // b == true -> a && b == true
b = "a und b sind true!";
draw_text(a,100,b); // zeichnet den String b an die Stelle 1,100
}
```

Diese Vereinfachung bringt jedoch den Nachteil mit sich, dass viel Speicherplatz verschwendet wird, da jede Variable, egal ob sie nur einen ganzzahligen oder bool'schen Wert enthält, immer den Speicherplatz einer Fließkommazahl benötigt.

## 2. Ein Lernprojekt

Am besten erklären und lernen kann man die Funktionen des Game Makers anhand eines Lernprojektes. Dieses Projekt ist ein kleines Geschicklichkeitsspiel und soll zeigen, mit wie wenig Aufwand ein einfaches Spiel im Game Maker erzeugt werden kann. Das Designdokument, das am Anfang jeder Spieleentwicklung steht, fällt in diesem Fall sehr knapp aus:

*Spielidee:* Im inneren eines menschlichen Körpers werden Zellorganellen von Viren angegriffen. Der Spieler muss die Viren mit einem Präzisionslaser vernichten.

*Spielablauf:* Anfangs befinden sich 10 Zellorganellen in der Mitte des Bildschirms. Außerhalb des Bildschirms erscheinen zufällig Viren, die sich auf das nächstliegende Organell zubewegen, um es anzugreifen. Der Spieler kann die Viren zerstören und erhält für jedes zerstörte Virus 10 Punkte. Je mehr Punkte der Spieler hat, desto stärker und zahlreicher werden die Viren, so dass der Spieler irgendwann keine Chance mehr hat, die Organellen zu beschützen. Haben die Viren alle Organellen zerstört, ist das Spiel vorbei und der Spieler kann sich in die Highscoreliste eintragen.

*Steuerung:* Das Spiel wird mit der Maus gesteuert, die durch ein Fadenkreuz auf dem Bildschirm repräsentiert wird. Durch Klicken der linken Maustaste wird der Laser abgefeuert. Die Taste Escape beendet das Spiel. Durch Drücken von F4 kann zwischen Fenster- und Vollbildmodus gewechselt werden.

*Grafik:* Das Spiel startet in einem Fenster mit einer Auflösung von 800x600 Pixeln. Für das Spiel werden vier Grafiken benötigt. Ein Hintergrundbild (800x600), ein Virus (ca. 64x64), ein Organell (ca. 64x64) und ein Fadenkreuz (ca. 32x32).

*Sound:* Für die Vertonung des Spiels werden Sounds für folgende Aktionen benötigt: Laserschuss, Lasertreffer, Verletzung. Ferner wird im Hintergrund ein Musikstück im MIDI-Format erklingen.

Um dieses Beispiel nachzuvollziehen, benötigen Sie eine unregistrierte Version des Game Makers 7 von <http://www.yoyogames.com/gamemaker/try>.

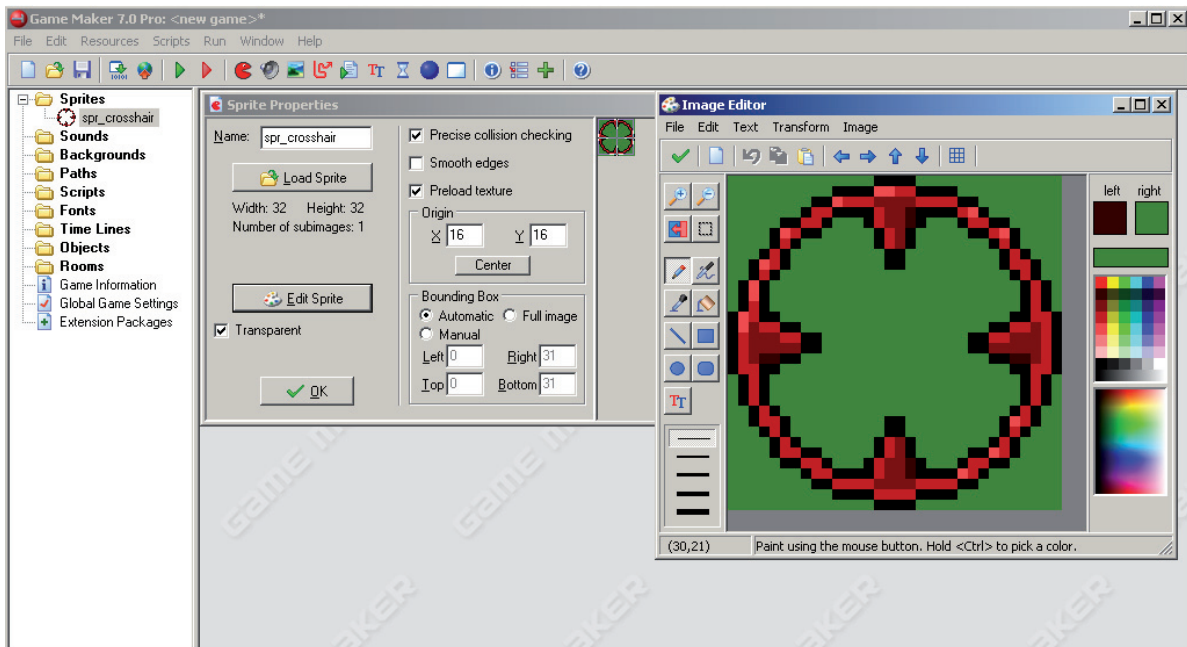
Installieren und starten Sie das Programm. Damit wir Scripts verwenden können, muss der Game Maker im *Advanced Mode* laufen. Falls das nicht der Fall ist, muss im Menü File der Menüpunkt *Advanced Mode* angeklickt werden.

Die Benutzeroberfläche des Game Makers ist sehr einfach gehalten: Oben befindet sich ein Windows-übliches Menü, gefolgt von einer Symbolleiste, die häufig benötigte Befehle, wie Funktionen zum Erstellen neuer Ressourcen oder zum Starten des Spiels enthält. Das Hauptfenster besteht aus dem Arbeitsbereich und dem Ressourcen-Explorer. Im Ressourcen-Explorer werden alle Grafiken, Sounds, Objekte und Räume angelegt, die das Spiel benutzen wird.

### 2.1 Sprites

Sprites sind die Grafiken, die später benutzt werden, um die Spielobjekte auf den Bildschirm zu zeichnen.

Sprites bieten außerdem die Möglichkeit, einen Nullpunkt und eine Bounding Box festzulegen. Der Nullpunkt bestimmt, an welchem Punkt die Grafik bewegt, gedreht und skaliert wird (die gebräuchlichsten Positionen eines Nullpunkts sind Links-Oben oder Mitte). Die Bounding Box wird für die Berechnung der Kollisionen zwischen den Objekten benötigt. Es besteht allerdings auch die Möglichkeit über die Funktion *Precise collision checking* eine pixelgenaue Kollisionsabfrage des Sprites zu verwenden. Ein Sprite kann auch aus mehreren Bildern bestehen, um beispielsweise animierte Grafiken darzustellen.



### spr\_crosshair

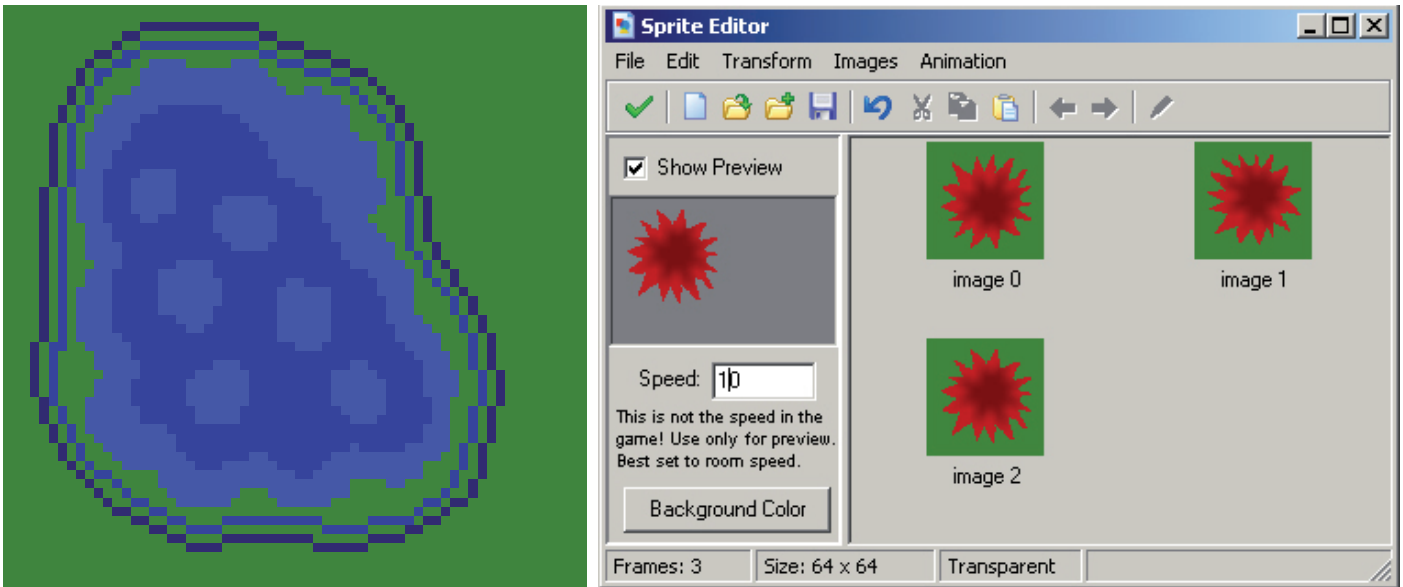
Um das Sprite für das Fadenkreuz anzulegen, klicken Sie entweder auf *Resources->Create Sprite* im Menü, auf das rote „Pacman“-Symbol in der Symbolleiste oder mit der rechten Maustaste auf den Sprites-Ordner im Ressourcen-Explorer und wählen sie *Create Sprite* aus dem Popup-Menü.

Im Arbeitsbereich öffnet sich nun ein *Sprite Properties* Fenster. Geben Sie unter Name `spr_crosshair` ein. Es ist empfehlenswert Präfixe vor die eigentlichen Namen der Ressourcen zu setzen, da der Game Maker Probleme bekommt, wenn Namen öfter als einmal auftreten und dadurch nicht mehr eindeutig sind. Setzen Sie den Nullpunkt des Sprites auf die Mitte des Sprites, indem Sie den Button mit der Aufschrift „Center“ anklicken. Mit *Edit Sprite* kommen wir in den Sprite Editor, der unter anderem das Anlegen von Animationen erlaubt (Über *Load Sprite* können auch externe Bilder geladen werden). Durch einen Doppelklick auf das momentan noch leere grüne Sprite mit der Bezeichnung *image 0* gelangen wir in den *Image Editor*, einem simplen Grafikprogramm. Zeichnen Sie dort ein Fadenkreuz, das genau auf die Mitte des Sprites zielt. Der linke untere Pixel legt die transparente Farbe fest. Falls in den *Sprite Properties* die Option *Transparent* aktiviert ist, werden alle Pixel die diese Farbe verwenden im Spiel nicht gezeichnet. Schließen Sie nun den *Image Editor*, den *Sprite Editor* und die *Sprite Properties* über den grünen „Bestätigungs-Haken“.

### spr\_organell

Legen Sie nun ein zweites Sprite mit dem Namen `spr_organell` an und ändern Sie die Größe durch den Menüpunkt *Transform->Resize Canvas* auf 64x64 Pixel. Zeichnen Sie im *Image Editor* ein rundliches Organell in blauen Farbtönen. Zuletzt sollten Sie auch hier den Nullpunkt zentrieren („Center“ im *Sprite Properties* Fenster).





### spr\_virus

Erstellen Sie noch ein letztes Sprite und nennen Sie es `spr_virus`. Setzen Sie die Maße ebenfalls auf 64x64 und zeichnen Sie ein rotes stacheliges Virus. Damit das Virus noch gefährlicher wirkt, sollte es animiert werden. Wählen Sie im *Sprite Editor* das fertige Bild *image 0* aus und kopieren Sie es über *Edit->Copy* (STRG+C). Mit *Edit->Paste* (STRG+V) können Sie nun mehrere Frames hinzufügen. Verändern Sie das Bild jedesmal ein bisschen und überprüfen Sie die Animation im *Sprite Editor*, indem Sie links *Show Preview* aktivieren. Sie sollten in den *Sprite Properties* die Option *Precise collision checking* deaktivieren, da sonst Probleme mit der Kollisionsabfrage entstehen können. Sie können die *BoundingBox* verkleinern, so dass es nachher aussieht, als würden die Viren ein wenig in die Organellen eindringen.

## 2.2 Backgrounds

Der Game Maker macht eine Unterscheidung zwischen Grafiken, die einem *Object* zugewiesen werden (*Sprites*), und Grafiken, die einem *Room* zugewiesen werden (*Backgrounds*). *Backgrounds* sind nicht animiert, können dafür jedoch ohne großen Rechenaufwand automatisch wiederholt werden, falls die Grafik kleiner als der Bildschirm ist. Außerdem besteht die Möglichkeit *Tiles* anzulegen und die Hintergründe der *Rooms* durch das Platzieren einzelner Kacheln (*Tiles*) zu gestalten.

Erstellen Sie einen *Background* über den Menüpunkt *Resources->Create Background* und geben Sie als Name `bgr_game` an. *Edit Background* öffnet den bereits bekannten *Image Editor*. Ändern Sie die Größe mit *Transform->Resize Canvas* auf 800x600 Pixel und zeichnen Sie ein dunkles Hintergrundbild, damit sich die bereits erstellten *Sprites* später gut abheben.

## 2.3 Objects und Scripts

### obj\_organell

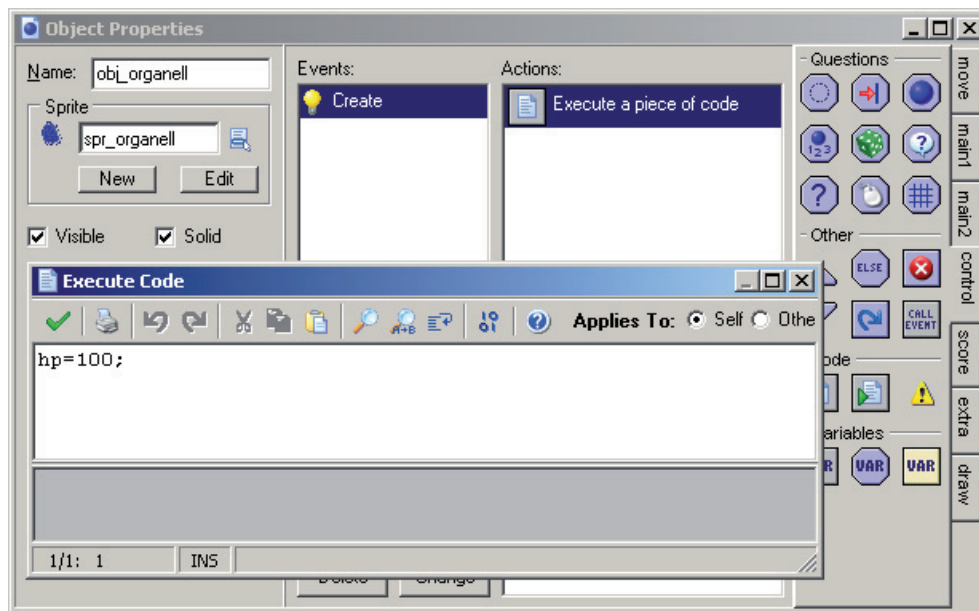
Das einfachste *Object* ist das Organell, da es über kein eigenes Verhalten verfügt. Erstellen Sie analog zu den bisherigen Ressourcen ein neues Object mit dem Namen `obj_organell` und dem Sprite `spr_organell`. Machen Sie einen Haken bei *solid*, damit ein Virus bei einer Kollision nicht in das Organell eindringt. Das Organell soll nicht beim sofortigen Kontakt mit einem Virus sterben, sondern erst nach einigen Attacken. Für dieses Feature benötigen wir eine lokale Variable, die die Lebenspunkte der Instanz speichert. Klicken Sie auf *Add Event* und wählen Sie das *Create Event* aus. Am rechten Rand des Object



Properties Fensters wählen Sie den Reiter *control* und ziehen das Icon *Execute code* in die *Actions* Spalte. Es öffnet sich ein Textfeld, in das Sie folgenden Code schreiben:

```
hp=100;
```

Diese Zeile deklariert die Variable `hp` und weist ihr den Wert 100 zu. Da der Code im *Create Event* steht, wird er nur einmal pro Instanz des Typs `obj_organell` ausgeführt.



### **obj\_virus**

Das Virus wird das wichtigste Object in unserem Spiel. Es muss sich über den Bildschirm bewegen, selbstständig Organellen angreifen und selbst verwundbar gegenüber Mausklicks sein. Wir benötigen also Events für Kollisionen und Mausklicks, sowie eine künstliche Intelligenz, die die Bewegung steuert. Die Bewegung werden wir über ein *Alarm Event* lösen, das nur alle paar Ticks aufgerufen wird und dann ein neues Ziel festlegt. Um den Code übersichtlich zu halten, erstellen wir für das Virus eigene *Scripts*, die wir dann bequem über den Ressourcen-Explorer bearbeiten können. Legen Sie also folgende *Scripts* an (*Resources->Add Script*), die vorerst noch leer bleiben: `scr_virus_create`, `scr_virus_alarm`, `scr_virus_hit` und `scr_virus_attack`.

Erstellen Sie nun ein neues *Object*, nennen Sie es `obj_virus` und weisen ihm das *Sprite* `spr_virus` zu. Jetzt können wir die verschiedenen *Events* erstellen und ihnen jeweils das passende *Script* zuweisen (Das *Execute Script* Icon befindet sich neben *Execute Code* im *control* Tab).

Weisen sie dem *Create Event* das `scr_virus_create` *Script* zu, dem *Alarm Event* „alarm 0“ das `scr_virus_alarm` *Script*, dem *Mouse Event* „Left Pressed“ das `scr_virus_hit` *Script* und dem *Collision Event* „obj\_organell“ das `scr_virus_attack` *Script*.

Erstellen Sie noch das *Event Collision Event* „obj\_virus“ und wählen Sie die *Action Bounce* aus dem *move* Tab. Geben Sie bei *precise* „not precise“ und bei *against* „all objects“ ein. Sollten nun zwei Viren aufeinandertreffen, dann werden diese voneinander abprallen.

Schließen Sie die *Object Properties* und bearbeiten Sie die *Scripts*:

#### **scr\_virus\_create:**

```
hp=5+random(score/50);  
scr_virus_alarm();
```

```
if (place_meeting(x,y,obj_virus)) instance_destroy();
image_speed=0.5;
```

Das Virus wird also ebenfalls Lebenspunkte besitzen, nur diesmal ist der Startwert abhängig von der aktuellen Punktzahl des Spielers. `score` ist eine globale Variable, die der Game Maker vorgibt. Sie kann von jedem *Object* aus abgefragt und verändert werden. Die Funktion `random(x)` liefert einen zufälligen Wert zwischen 0 und x. In der zweiten Zeile wird das Script `scr_virus_alarm` manuell aufgerufen, damit sich das Virus gleich von Beginn an einen Weg sucht. Die dritte Zeile überprüft, ob es an der Stelle, an der sich das Virus gerade befindet schon ein anderes Virus befindet und entfernt sich selbst, falls dies der Fall ist. Damit soll verhindert werden, dass zufällig erscheinende Viren, aufeinander platziert werden, was durch die *Bounce Action* sonst Probleme bereiten würde. Die letzte Zeile verlangsamt die Animation, die sonst viel zu schnell ablaufen würde. Eine `image_speed` von 1 würde bedeuten, dass jeden *Step* das Bild gewechselt wird, der Wert 0.5 erreicht, dass dies nur noch alle zwei *Steps* passiert, so dass sich die Geschwindigkeit der Animation halbiert.

#### **scr\_virus\_alarm:**

```
target=instance_nearest(x,y,obj_organel1);
if (target!=noone){
    direction=point_direction(x,y,target.x,target.y);
    if (random(4)<1) direction=random(360);
    speed=6;
    alarm[0]=10+random(10);
}
```

Das Script `scr_virus_alarm` sorgt dafür, dass das Virus sich ein Ziel sucht und sich darauf zubewegt. In der ersten Zeile wird eine Referenz auf die nächstliegende Instanz vom Typ `obj_organel1` in der lokalen Variable `target` gespeichert. Die Funktion `instance_nearest(x,y,obj)` liefert den Wert `noone`, falls keine Instanz `obj_organel1` existiert. Die *if*-Abfrage in der zweiten Zeile stellt sicher, dass der Code in der Klammer nur ausgeführt wird, wenn die Variable `target` nicht den Wert `noone` erhalten hat. `direction` ist eine lokale Variable und gibt die Richtung der Bewegung der Instanz wieder. Die Funktion `point_direction(x1,y1,x2,y2)` liefert den Winkel zwischen einer Geraden durch die Punkte (x1,y1) und (x2,y2) und der x-Achse. Die vierte Zeile sorgt also dafür, dass das Virus sein Ziel anvisiert. Um die Bewegung etwas weniger vorhersehbar zu machen, setzt die folgende Zeile mit 25% Wahrscheinlichkeit die Richtung wieder auf einen Zufallswinkel. Die lokale Variable `speed` sorgt schließlich dafür dass sich die Instanz auch wirklich bewegt. Der Wert 6 bedeutet, dass sich das Virus fortan jeden *Step* um 6 Pixel in die Richtung `direction` verschiebt. Die letzte Zeile des Blocks setzt den Wert von *Alarm 0* wieder auf einen Wert zwischen 10 und 20 *Steps*. Nach Ablauf dieser Anzahl von *Steps* wird das Script erneut aufgerufen.

#### **scr\_virus\_hit:**

```
effect_create_above(ef_ring,mouse_x,mouse_y,3,c_red);
hp-=10;
if (hp<0){
    score+=10;
    effect_create_above(ef_explosion,x,y,4,c_red);
    instance_destroy();
}
```

Dieses Script wird immer dann ausgeführt, wenn der Spieler das Virus mit der Maus anklickt, es also mit dem Laser unter Beschuss nimmt. Wir nutzen hier die Effektengine des Game Makers, die nicht sonder-

lich umfangreich ist. Wesentlich eindrucksvoller ist es, wenn man eigene Effekte programmiert, aber für unser Lernprojekt reicht dieser Effekt vollkommen aus. Mit der Funktion `effect_create_above(effect, x, y, size, color)` erstellen wir in der ersten Zeile einen roten Ring-Effekt, der einen Treffer signalisieren soll. In der zweiten Zeile wird die lokale Variable `hp` um 10 verringert. Sinkt `hp` unter 0, dann hat der Spieler das Virus zerstört. Die globale Variable `score` wird erhöht, ein Explosionseffekt erzeugt und das Virus mit `instance_destroy()` aus dem Spiel entfernt.

#### **scr\_virus\_attack:**

```
speed=0;
other.hp-=10;
if (other.hp<=0) {
effect_create_above(ef_explosion,other.x,other.y,2,c_blue);
with (other) instance_destroy();
if (instance_number(obj_organe11)==0) {
    highscore_show(score);
    score=0;
    room_restart();
}
scr_virus_alarm();
}
effect_create_above(ef_ring, (x+other.x)/2, (y+other.y)/2,1,c_blue);
alarm[0]=10+random(10);
```

Wenn ein Virus mit einem Organell kollidiert, dann startet dieses Skript. Innerhalb eines Kollisionsevents enthält die lokale Variable `other` eine Referenz auf die Instanz, mit der die Kollision stattfand. Deshalb wird in der zweiten Zeile, die Variable `hp` des Organells verringert. Gleich danach wird abgefragt, ob das Organell durch diesen Angriff zerstört wurde. Falls ja, wird ein Explosionseffekt erstellt (diesmal in blau) und das Organell entfernt. Die `with`-Konstruktion bewirkt, dass sich die nachfolgende Anweisung auf die Instanz in der Klammer bezieht, in diesem Fall auf das Organell (`other`), das die Kollision ausgelöst hat.

Nun wird zusätzlich noch überprüft, ob es sich bei der zerstörten Instanz um die letzte ihrer Art gehandelt hat. Wenn nämlich alle Organellen zerstört sind, dann endet das Spiel. Durch die Funktion `highscore_show(score)` wird dann der aktuelle Punktestand mit den Einträgen einer Highscoreliste verglichen, in die sich der Spieler dann eventuell verewigen kann. Danach werden die Punkte gelöscht und der Room und damit das Spiel, neugestartet.

Sollte es sich jedoch nicht um die letzte Instanz vom Typ `obj_organe11` gehandelt haben, dann wird durch das `scr_virus_alarm` Skript ein neues Ziel gesucht und ganz gleichgültig, ob das Ziel zerstört wurde oder nicht, wird in den letzten beiden Zeilen ein blauer Ring-Effekt erzeugt und `Alarm 0`, der -wie inzwischen bekannt- ebenfalls das `scr_virus_alarm` Skript aufruft, wieder auf 10-20 *Steps* gesetzt. Da am Anfang die Geschwindigkeit auf 0 gesetzt wurde und da das *Object*, mit dem die Kollision stattfand, die Eigenschaft `solid` besitzt, findet bis zum Ablauf des *Alarms* kein neuer Aufruf des *Collision Events* mehr statt.

Das `obj_virus` ist damit fertig und einsatzbereit. Zu Testzwecken können Sie einen Room erstellen (*Resources->Create Room*) und mehrere Instanzen der beiden *Objects* verteilen. Wenn Sie dann das Spiel starten (F5), werden die Viren die Organellen angreifen und Sie können sie mit der Maus „totklicken“.

## obj\_controller

Das Spiel ist allerdings noch nicht sonderlich spannend, da wir spätestens beim zweiten Versuch bereits wissen, wo sich die Viren befinden und diese rechtzeitig ausschalten können. Was dem Spiel noch fehlt, ist ein Mechanismus, der die Viren und Organellen zufällig erzeugt und ständig neuen Nachschub an Viren erzeugt, die mit der Zeit immer stärker und zahlreicher werden, so dass der Spieler irgendwann keine Chance mehr gegen die Übermacht hat. Außerdem fehlen noch einige grafische Effekte wie Laserstrahlen und das Fadenkreuz, sowie eine Punkteanzeige. Hier kommt unser Controller *Object* ins Spiel. Zunächst legen wir wieder einige *Scripts* an, die wir später *Events* zuweisen. Das sind für das Controller *Object* nur drei Stück: `scr_controller_create`, `scr_controller_alarm` und `scr_controller_draw`. Legen Sie ein neues *Object* mit dem Namen `obj_controller` an, erstellen Sie ein *Create Event*, ein *Alarm Event* „alarm 0“ und ein *Draw Event* und weisen Sie die *Scripts* entsprechend zu. Dieses *Object* benötigt kein *Sprite*, da wir das *Draw Event* gerade mit einem eigenen *Script* ersetzt haben, was das Zeichnen des eingestellten *Sprites* deaktiviert. Damit das Controller *Object* als letztes und damit über alles andere gezeichnet wird, geben Sie ihm eine *Depth* von -1. Kommen wir nun zu den *Scripts*:

### scr\_controller\_create:

```
x1=0;
y1=0;
x2=0;
y2=0;
alpha=0;
instance_create(
    random(room_width/2)+room_width/4,
    random(room_height/2)+room_height/4,
    obj_organell);
do{
    x=random(room_width/2)+room_width/4;
    y=random(room_height/2)+room_height/4;
    nearest=instance_nearest(x,y,obj_organell);
    if (point_distance(x,y,nearest.x,nearest.y)>50)
        instance_create(x,y,obj_organell);
} until (instance_number(obj_organell)==10)
scr_controller_alarm();
```

Zunächst werden 5 lokale Variablen `x1`, `y1`, `x2`, `y2` und `alpha` deklariert, die später zum Zeichnen des Laserstrahls verwendet werden sollen. Danach wird eine Instanz von `obj_organell` an einer zufälligen Position, in einem Bereich, der ein Viertel des Rooms ausmacht und sich in dessen Mitte befindet, platziert. In der darauf folgenden nachgetesteten Schleife werden so lange neue Instanzen platziert, bis 10 Organellen existieren. Dabei wird darauf geachtet, dass die einzelnen Instanzen einen Mindestabstand von 50 Pixeln haben, damit sie nicht gelegentlich ineinander kleben. Nach der Schleife wird das Skript `scr_controller_alarm` gestartet, welches dafür sorgt, dass in regelmäßigen Abständen neue Viren erzeugt werden.

### scr\_controller\_alarm:

```
if (random((score/50)+4)>3) {
    switch(floor(random(4))) {
        case 0:
            instance_create(random(room_width),-50,obj_virus);
```

```

        break;
        case 1:
            instance_create(-50,random(room_height),obj_virus);
            break;
        case 2:
            instance_create(random(room_width),room_height+50,obj_virus);
            break;
        case 3:
            instance_create(room_width+50,random(room_height),obj_virus);
            break;
    }
}
alarm[0]=room_speed;

```

Mit einer anfänglichen Chance von 25% wird eine Instanz von `obj_virus` außerhalb des Rooms erstellt. Diese Chance wird mit jedem Punkt größer, nach 10 zerstörten Viren beträgt sie schon 50%. Die Erzeugung der Viren wird wieder durch Zufall in einer *Switch-Case* Verzweigung abgehandelt. Der Zufallswert kann die Werte 0,1,2 oder 3 annehmen, die jeweils für eine Richtung stehen. Ist der Wert 0, dann kommt das Virus von oben, bei 1 von links, bei 2 von unten und bei 3 von rechts. Am Ende wird wieder der Alarm neu gestellt, diesmal allerdings auf den `room_speed`, der die *Steps* pro Sekunde angibt, was bedeutet, dass es genau eine Sekunde dauern wird, bis das *Script* erneut ausgeführt wird.

#### **scr\_controller\_draw:**

```

// draw crosshair
draw_sprite(spr_crosshair,0,mouse_x,mouse_y);
// create red laser beam
if (mouse_check_button_pressed(mb_left)){
    alpha=0.75;
    switch(floor(random(4))){
        case 0: // top border of the room
            x1=random(room_width-20);
            x2=x1+20;
            y1=0;
            y2=0;
            break;
        case 1: // left border of the room
            x1=0;
            x2=0;
            y1=random(room_height-20);
            y2=y1+20;
            break;
        case 2: // bottom border of the room
            x1=random(room_width-20);
            x2=x1+20;
            y1=room_height;
            y2=y1;
            break;
        case 3: // right border of the room
            x1=room_width;
            x2=x1;
            y1=random(room_height-20);
            y2=y1+20;break;
    }
}

```

```

    }
}
alpha-=0.15; // laser fades away
// draw red laser beam
draw_set_alpha(alpha);
draw_set_color(c_red);
draw_triangle(x1,y1,x2,y2,mouse_x,mouse_y,false);
// draw score text
draw_set_alpha(1);
draw_text(25,25,"SCORE: "+string(score));

```

Dieses Script ist etwas lang geraten, weshalb es zur Übersicht beiträgt, wenn man die einzelnen Schritte im Code kommentiert. Ein Kommentar ist durch zwei Schrägstriche gekennzeichnet, die dem Interpreter signalisieren, dass er den Rest der Zeile ignorieren soll.

Die erste Anweisung zeichnet das Fadenkreuz auf die Koordinaten der Maus, damit der Spieler nicht mit dem Standard Windows Mauscursor schießen muss.

Danach wird überprüft, ob die linke Maustaste gerade eben geklickt wurde. Ist dies der Fall, dann wird die lokale Variable `alpha` auf 0.75 gesetzt und anschließend, wie zuvor, in einer Switch-Case Verzweigung per Zufall der Bildrand ausgewählt, von dem der Laserstrahl kommen soll. Dabei werden zwei Punkte (`x1,y1`) und (`x2,y2`) festgelegt, die einen Abstand von 20 Pixeln zueinander haben. Zusammen mit der Mausposition existieren nun 3 Punkte, durch die mit der Funktion `draw_triangle(x1,y1,x2,y2,x3,y3,outline)` ein rotes Dreieck gezeichnet wird. Durch die Zeilen `alpha-=0.15;` und `draw_set_alpha(alpha);` erscheint das Dreieck leicht transparent und wird dann innerhalb von vier Steps komplett ausgefadet. Dies erzeugt einen wesentlich weicheren Effekt und damit einen glaubwürdigeren Laserstrahl.

Die letzten zwei Zeilen deaktivieren die Transparenz und schreiben den aktuellen Punktestand auf die Koordinaten (`x=25,y=25`). Auch hier könnte man mit dem Befehl `draw_set_color(color)` noch einmal die Textfarbe ändern oder eine Schriftart mit `draw_set_font(font)` festlegen.

## 2.4 Rooms

Das Spiel ist fast fertig. Löschen Sie den eventuell vorhandenen Room, falls Sie vorher schon einmal getestet haben, ob das Spiel funktioniert und erstellen Sie einen neuen (*Resources->Create Room*). Zuerst platzieren Sie das `obj_controller Object` irgendwo im Room. Sie müssen im *objects* Tab der *Room Properties* sein, um *Objects* auszusuchen und platzieren zu können. Das Controller *Object* erzeugt alle anderen *Objects*, weswegen es das einzige ist, das wir im Room setzen müssen.

Gehen Sie auf den Tab *backgrounds*, klicken Sie auf das Drop-Down Menü, das momentan `<no background>` anzeigt und wählen Sie `bgr_game` als Hintergrundbild aus.

Unter settings können Sie dem Room noch einen Namen geben und die Größe auf das Hintergrundbild anpassen, indem Sie bei *Width* 800 und bei *Height* 600 eintragen.

## 2.5 Abschluss des Projektes

### Global Game Settings

Das Spiel ist endlich fertig, es müssen nur noch einige Kleinigkeiten angepasst werden. Der vorletzte Punkt im Ressourcen-Explorer heißt *Global Game Settings* und erlaubt grundlegende Einstellungen am



Spiel. Zuerst sollten wir im Tab *Graphics* die Option *Display the cursor* deaktivieren, damit der normale Mauszeiger von Windows nicht noch zusätzlich über dem Fadenkreuz gezeichnet wird. Außerdem sollten wir die Option *Let <F5> save the game and <F6> load a game* unter dem Tab *other* deaktivieren, damit der Spieler nicht durch wiederholtes Speichern und Laden schummeln kann.

## Sounds

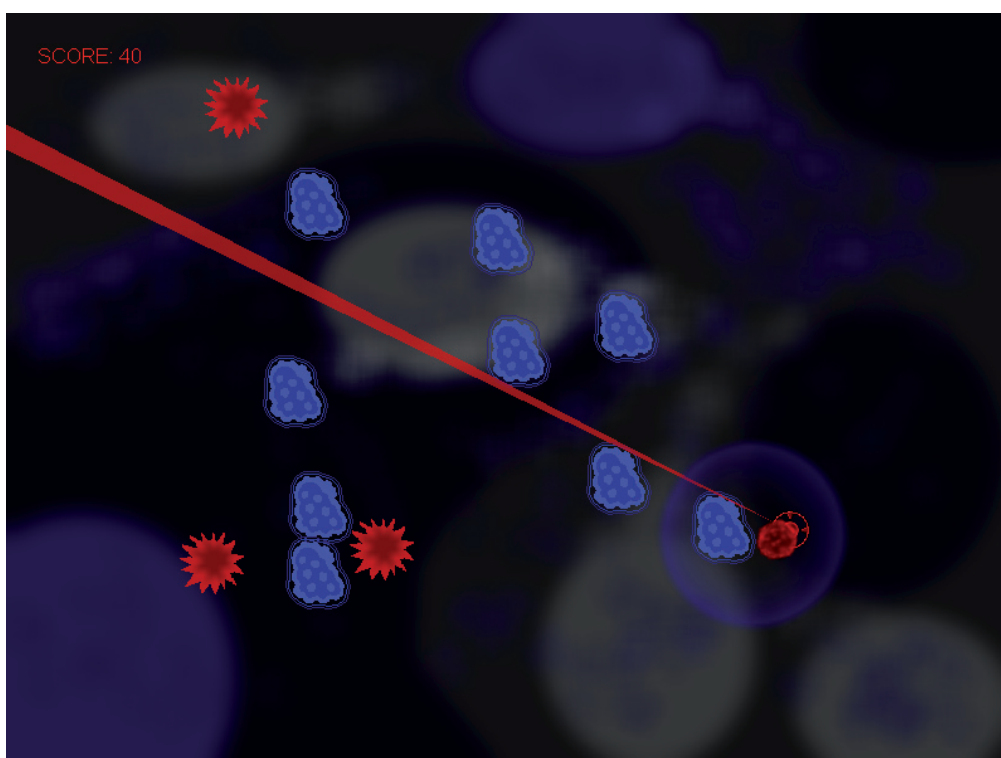
Das Spiel enthält noch keine Sounds oder Musikuntermalung. Das Integrieren von Sounds stellt jedoch keinen großen Aufwand dar. Es müssen lediglich Ressourcen vom Typ *Sound* erstellt und in den Scripts an den entsprechenden Stellen mit der Funktion `sound_play(sound)` gestartet werden. Um den Sound in Schleife zu spielen, was für Musik recht praktisch ist, existiert die Funktion `sound_loop(sound)`.

## Menü

Das Spiel besteht momentan nur aus einem Raum, was nicht sehr professionell wirkt. Besser ist es, wenn eine Partie erst über ein Hauptmenü gestartet wird, das auch Funktionen zum Anzeigen des Highscores und Beenden des Spiels bietet. Erstellen Sie dazu einen schönen *Background*, drei *Sprites* für die anklickbaren Buttons „New Game“, „Show Highscore“ und „Exit“ und drei *Objects*, die diese *Sprites* benutzen. Jedes Object bekommt ein Mouse Event „Left Button“, das im ersten Fall die Funktion `room_goto_next()`, im zweiten Fall `highscore_show(-1)` und im dritten Fall `game_end()` aufruft. Ob Sie *Execute Code*, *Execute Script* oder die passenden Drag&Drop *Actions* benutzen, bleibt Ihnen überlassen. Zuletzt müssen Sie nur noch einen neuen *Room* erstellen, ihn im Ressourcen-Explorer an die erste Stelle der Liste ziehen und die drei neuen *Objects* darin platzieren sowie den *Background* einstellen.

## Create Executable

Damit das Spiel auch von anderen Menschen gespielt werden kann, die nicht den Game Maker besitzen, können Sie mit dem Befehl *Create Executable* aus dem *File* Menü eine allein lauffähige Windows **exe** Datei erzeugen lassen. Dieses Spiel unterliegt keiner Lizenzpflicht seitens des Game Makers und kann kostenlos veröffentlicht oder auch verkauft werden.





### 3. (Bei-) Spiele

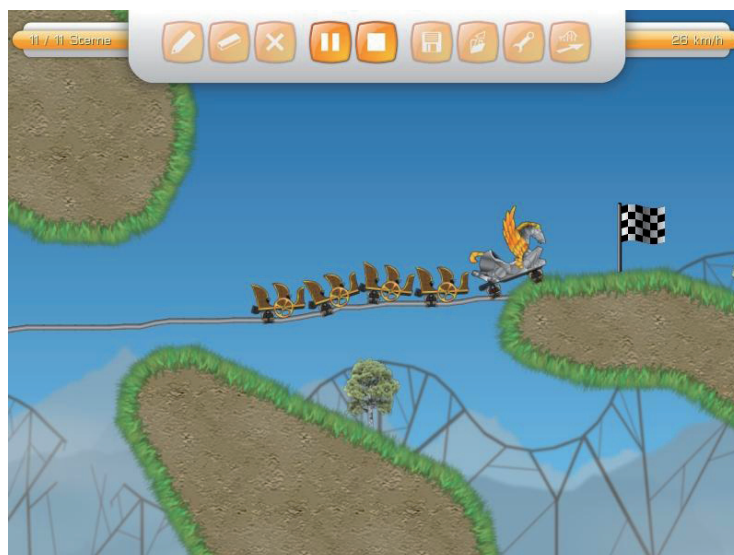
Um die Möglichkeiten des Game Makers noch umfangreicher darzustellen, werde ich in diesem Kapitel noch einige Spiele vorstellen, die mit diesem Programm erstellt wurden.

#### 3.1 Ninja ♥ Pirate



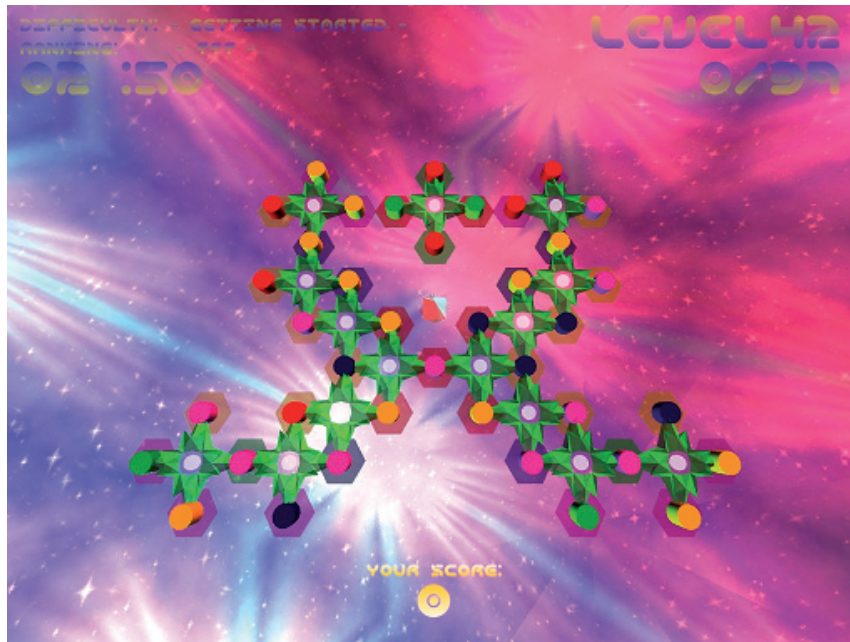
Hierbei handelt es sich um ein Jump'n'Run, das vor allem durch seine detaillierte grafische Ausarbeitung und schönes Leveldesign beeindruckt. Technisch lehnt es sich an die Spiele der *Super Nintendo* Konsole an, weshalb die Auflösung sehr niedrig ausfällt. Das Spiel wurde speziell für den **4 Elements Contest IV** der Seite [www.gamedev.net](http://www.gamedev.net) entwickelt, den es auch gewonnen hat. Eine Demo Version des Spiels kann auf der Seite [www.ninjalovespirate.com](http://www.ninjalovespirate.com) heruntergeladen werden.

#### 3.2 Coaster Rider



In Coaster Rider muss der Spieler mit der Maus eine Bahn zeichnen, über die ein Achterbahnwagen fährt, der bestimmte Ziele erreichen muss. Das Spiel hat die Pegasus Achterbahn des Europa Parks zum Vorbild und wurde für die Europa-Park Fanseite [www.ep-inside.net](http://www.ep-inside.net) entwickelt, von der es auch kostenlos heruntergeladen werden kann.

### 3.3 Circular Level



Circular Level ist ein abstraktes Denkspiel, in dem verschiedenfarbige Spielsteine durch Drehung kreuzförmiger Objekte in die farblich passenden Ziele befördert werden müssen. Das Spiel nutzt schöne 3D-Grafik und kann in einer 5-Level Demo Version von der Homepage des Entwicklers [www.n-organism.com](http://www.n-organism.com) heruntergeladen werden.

### 4. Fazit

Der Game Maker ist sicher keine Anwendung, mit der Spiele in der Größe eines World of Warcraft oder Command & Conquer 3 erstellt werden können. Doch für kleinere Projekte, Werbespiele oder Minigames ist er ideal geeignet. Er verbindet große Komplexität mit einem leichten Einstieg und einer einfach zu erlernenden Scriptsprache. Der große Nachteil des Programms ist die Speicherung des Projektes in einer einzelnen Datei. Grafiken, Sounds und Scripts können zwar ausgelagert und erst zur Laufzeit geladen werden, die Arbeit im Team ist trotzdem stark erschwert. Trotzdem ist der Game Maker gerade für einzelne das ideale Werkzeug zum Erlernen von Game Design und ein erster Schritt in die Welt der Programmierung.

### 5. Quellenangaben

<sup>1</sup> **Jacob Habgood, Mark Overmars „The Game Maker’s Apprentice“** ISBN 1-59059-615-3

<sup>2</sup> **Mark Overmars’ GLOG** <http://mark.glog.yoyogames.com/?p=6>

<sup>3</sup> **YoYo Games Homepage** <http://www.yoyogames.com/wiki/show/Some%20Facts>

<sup>4</sup> **YoYo Games GLOG** <http://glog.yoyogames.com/?p=8>

<sup>5</sup> **GM Knowledge Base** [http://gmkb.madladdesigns.co.uk/index.html?order\\_of\\_events.htm](http://gmkb.madladdesigns.co.uk/index.html?order_of_events.htm)

<sup>6</sup> **Deutsches Game Maker Forum** <http://www.gm-d.de/help/>