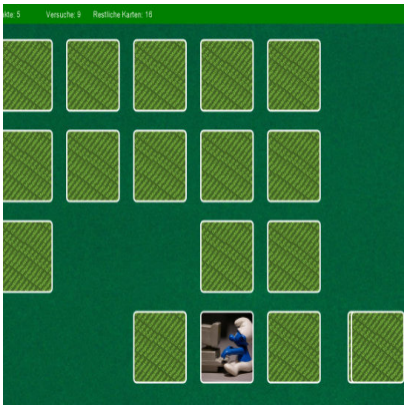
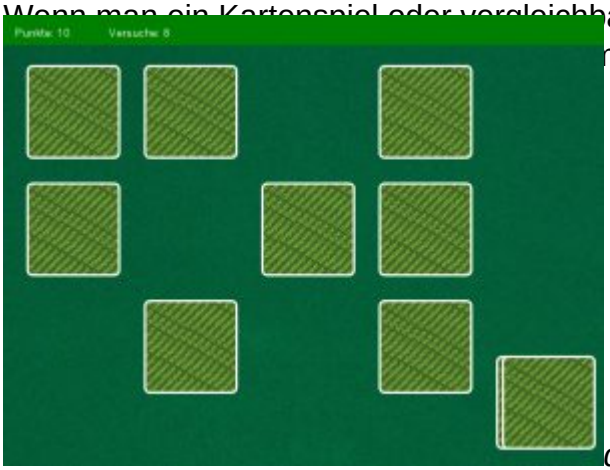


Memory im GameMaker

Description

Wenn man ein Kartenspiel oder vergleichbare Spiele im GameMaker machen möchte, bietet sich



neln, ist Memory das perfekte Spiel.

ds_list ist eine Datenstruktur in der Informationen wie in einem eindimensionalen Array gespeichert werden. Die Funktionen für Arrays sind besonders in GMS 1 stark limitiert, weshalb man für komplexere Dinge auf *ds_list*, *ds_map* oder *ds_grid* zurückgreifen muss. Ein Vorteil von *ds_list* gegenüber einem Array ist, dass sich die Werte in der Datenstruktur mischen lassen, was für ein Kartenspiel nicht unerheblich ist.

Vom Spiel habe ich drei Versionen gemacht. Eine Basic-Version, in der alle nötigen Funktionen enthalten sind. Die Deluxe-Version enthält mehr Karten, zwei Kartensets und am Anfang werden die Karten einzeln auf den Tisch gelegt, was wesentlich hübscher aussieht. Dazu habe ich die Delux-Version noch in GMS2 portiert. Alle drei Downloads befinden sich unten im Artikel. Dazu gibt es noch ein Video, in dem ich die Versionen kurz zeige und ein paar Auszüge erkläre.

Im Tutorial werde ich die Basic-Version ausführlich erklären, die Delux-Version wird nur insofern erklärt, dass ich die Unterschiede beschreibe. Der Code ist jeweils sehr ausführlich kommentiert.

Was wir brauchen

Erst einmal brauchen wir zwei Sprite-Ressourcen. In **spr_karten** kommen alle Karten inklusive der Rückseite hinein. Die Rückseite ist auf Index 0. Für die Basic-Version gibt es 6 verschiedene Karten. Jede Karte wird im Sprite nur einmal benötigt.

spr_kartenstapel brauchen wir für einen Stapel, den wir neben dem Spielfeld platzieren. Bei mir besteht der Stapel auf 5 Frames (0 bis 4) wobei Frame 0 leer ist. Das ist der Ausgangszustand und wenn Karten weggenommen werden, füllt sich der Stapel.

bg_spieltisch ist der Hintergrund für unseren Raum. Das Spiel soll ja auch nach etwas aussehen, weshalb ich eine 1024×768 große Filztisch-Textur verwendet habe.

fnt_punkte ist unsere Schrift für die Punkte und die Anzeige am Ende des Spiels. Unkreativerweise habe ich Arial 14 genommen.

room_memory ist unser Raum. Die Größe beträgt 1024×768, Speed 60.

Für das ganze Spiel brauchen wir drei Objekte. **obj_kartenstapel** ist, wie der Name schon sagt, der Kartenstapel. **obj_memory_karte** stellt die einzelne Karte dar und kontrolliert sie. In **obj_controller** steckt die ganze Spiellogik. Dieses Objekt wird im Raum platziert.

obj_kartenstapel

Dem Objekt weisen wir **spr_kartenstapel** zu. Dann brauchen wir nur noch ein Event. Der Rest wird über **obj_controller** geregelt.

Event Create

```
image_index = 0;
image_speed = 0;
```

Das war es schon. Wenn das Objekt erstellt wird, wird Bild 0 angezeigt (der leere Frame) und die Geschwindigkeit des Sprites steht bei 0.

obj_memory_karte

Hier weisen wir **spr_karten** zu. Dazu brauchen wir drei Events. Ein Teil der Kontrolle wird ebenfalls von **obj_controller** ausgeführt.

Event Create

```
kartenwert = 0;     // Den Kartenwert erhält die Karte vom Controller
rueckseite = true;  // Standardmäßig wird die Rückseite angezeigt
```

Hier definieren wir lediglich zwei Variablen. Der Wert liegt vorerst bei 0 und es wird die Rückseite angezeigt. Das ist im **Draw-Event** relevant.

Event Left Released

```
// Wenn die Rückseite gezeigt wird und der Controller bereit ist...
if (rueckseite) && (obj_controller.aktion)
{
    rueckseite = !rueckseite;   // Die Karte wird umgedreht

    // Wenn die Anzahl Karten 0 ist, wird der Wert in die erste Variable gesch
    // und wenn nicht, in die zweite. Das Step-Event des Kontrollers prüft,
    // ob die Werte gleich sind
    if (obj_controller.aufgedeckteKarten = 0)
    {
        obj_controller.kartenWert[0] = kartenwert;
        obj_controller.aufgedeckteKarten = 1;
    } else {
        obj_controller.kartenWert[1] = kartenwert;
        obj_controller.aufgedeckteKarten = 2;
    }
}
```

Wie man sehen kann, greifen wir hier fleißig auf Variablen von **obj_controller** zu. Wenn wir die Freigabe erhalten, wird die Karte umgedreht. Jede Karte auf dem Tisch bekommt einen Wert und es gibt immer zwei Werte. Die Karte gibt ihren Wert an **obj_controller** weiter. **obj_controller** vergleicht die zwei Werte später und entscheidet, ob die Karten gleich sind und vom Tisch verschwinden, oder wieder umgedreht werden.

Event Draw

```
// Wenn die Karte nicht umgedreht wurde, wird nur die Rückseite gezeigt.
// Wenn sie umgedreht wurde, hängt der Sprite-Index vom Kartenwert ab.

if (rueckseite)
{
    draw_sprite(spr_karten, 0, x, y);
} else {
    draw_sprite(spr_karten, kartenwert, x, y);
}
```

Wenn die Rückseite angezeigt wird, ist der Index 0. Ansonsten wird der Kartenwert als Spriteindex verwendet. Der Kartenwert wird ebenfalls von **obj_controller** vorgegeben.

obj_controller

Jetzt geht es ans Eingemachte. In diesem Objekt brauchen wir mindestens 6 Events. Im Beispiel habe ich 8, wobei ein Event für den Abbruch mit Escape zuständig ist. Der andere startet das Spiel neu, wenn man die Taste R drückt.

Event Create

```
aufgedeckteKarten = 0;  // Wie viele Karten sind aufgedeckt?
kartenWert[0] = 0;      // Wert der ersten aufgedeckten Karte
kartenWert[1] = 0;      // Wert der zweiten aufgedeckten Karte
punkte = 0;             // Punkte
punkteBelohnung = 20;   // Punkte, wenn es richtig ist
punkteBestrafung = 5;   // Punkte die abgezogen werden, wenn es falsch ist
versuche = 0;           // Versuche
aktion = true;          // Kontrolliert den Step. Wenn ein Alarm läuft, darf d
kartenStapel = 0;       // Am Anfang sind noch keine Karten auf dem Stapel
kartenAufDemTisch = 0;  // Wir zählen die Karten auf dem Tisch, damit wir wiss
gameOver = false;       // Variable um zu ermitteln, wann das Spiel aus ist

kartenspiel = ds_list_create(); // Wir legen ein Kartenspiel an

// Wir füllen das Spiel mit 6 verschiedenen Karten. Jede Karte muss doppelt
// im Stapel vorliegen.
// i + 1 bedeutet, dass der Wert größer Null sein muss.
for (i = 0; i < 6; i++)
{
    ds_list_add(kartenspiel, i + 1);
    ds_list_add(kartenspiel, i + 1);
}

randomize();                    // Wir müssen einmal den Zufallsgenerator star
ds_list_shuffle(kartenspiel);   // Wir mischen das Spiel

// Die Karten werden auf dem Tisch verteilt

wert = 0;   // Der Wert wird den Karten weiter gegeben

for (i = 0; i < 4; i++)
{
    xx = 120;       // Startwert für x
    yy = 165;       // Startwert für y
    abstand = 200;  // Abstand zwischen den Karten

    for (a = 0; a < 3; a++)
    {
        wert++;     // Der Kartenwert wird um 1 erhöht, bevor die nächste Kart
        karte[wert] = ds_list_find_value(kartenspiel, wert - 1);    // Der Kar
        spielkarte = instance_create(xx + i * abstand, yy + a * abstand, obj_m
        spielkarte.kartenwert = karte[wert];    // Wir geben der Karte ihren W
        kartenAufDemTisch++;                    // Wir haben eine weitere Kart
    }
}

stapel = instance_create(840, 580, obj_kartenstapel);
```

Das ist jetzt etwas viel Code, aber schauen wir uns das Blockweise an. Zunächst werden viele Variablen definiert. Was sie bedeuten, wird bereits im Kommentar erklärt. Wichtig ist das Prinzip: Wir zählen, wie viele Karten aufgedeckt wurden und welchen Wert die jeweilige Karte hat. Für einen Fehlversuch werden 5 Punkte abgezogen, wenn es zwei gleiche Karten sind, gibt es 20 Punkte. Die Versuche werden ebenfalls gezählt. Wichtig ist die Variable *aktion*, die den Step ausschaltet, wenn ein

Alarm gerade etwas macht.

Das Kartenspiel legen wir mit *ds_list_create();* an. In der ersten Schleife generieren wir die Karten. Wir brauchen von jedem Wert 2 Karten im Stapel. *i* beginnt bei Null. Da 0 im Sprite die Rückseite ist, muss der Wert immer eine Zahl größer sein als 0. Deswegen auch *ds_list_add(kartenspiel, i + 1);*. Mit 6 Durchläufen erzeugen wir also 12 Karten.

Dann starten wir den Zufallsgenerator und mischen mit *ds_list_shuffle(kartenspiel);* den Stapel virtuell durch. *ds_list* ist damit abgeschlossen. Wir haben die Liste erstellt, mit Werten gefüllt und gemischt. Nun müssen die Karten auf den Tisch gelegt werden.

Wie so oft haben wir auch hier Reihen und Spalten. Bei 12 Karten unterteilen wir es in vier Spalten und drei Reihen. In anderen Tutorials sind wir schon oft darauf eingegangen, weshalb ich es an dieser Stelle etwas verkürzt beschreibe. Wir brauchen dazu zwei ineinander liegende Schleifen. Pro Spalte werden drei Reihen generiert. Über die Variablen *i* und *a* der Schleifen sowie *xx*, *yy* und *abstand* errechnet sich die Position jeder einzelnen Karte. Das Ganze findet sich in Zeile 41 wieder, die da Lautet:

```
spielkarte = instance_create(xx + i * abstand, yy + a * abstand, obj_memory_ka
```

Davor wird in Zeile 40 der passende Wert aus dem Stapel gelesen. *karte[wert] = ds_list_find_value(kartenspiel, wert – 1);*. *ds_list_find_value* gibt den Wert der angegebenen Karte wieder. Der Wert wird in Zeile 42 an die auf den Tisch gelegte Karte weiter gegeben.

Am Ende wird nur noch der Kartenstapel, der am Anfang nicht sichtbar ist, auf den Tisch gelegt.

Event Step

```
if (aufgedeckteKarten = 2) && (aktion)
{
    if (obj_controller.kartenWert[0] = obj_controller.kartenWert[1])
    {
        alarm[0] = 1 * room_speed;  // Im Alarm0 wird der Code ausgeführt
    } else {
        alarm[1] = 1 * room_speed;  // Im Alarm1 wird der Code ausgeführt
    }

    aktion = false;
}

if (kartenAufDemTisch = 0)
{
    alarm[2] = 5 * room_speed;  // Das Spiel startet in 5 Sekunden neu
    gameOver = true;            // Das Spiel ist aus
    kartenAufDemTisch--;        // Wir ziehen noch eine Karte ab, damit der Al
}
```

In der ersten Abfrage wird geschaut, ob – wenn zwei Karten aufgedeckt wurden – die Karten identisch sind. Wenn ja, wird **Alarm 0** ausgeführt, wenn nicht, **Alarm 1**. Die zweite Abfrage stellt fest, ob der Tisch leer ist. Wenn ja, ist das Spiel beendet. **Alarm 2** wird gestartet und das Spiel wird in 5 Sekunden

neu ausgeführt.

Event Alarm 0

```
/// Karte wurde richtig aufgedeckt

with (obj_memory_karte)
{
    if (!rueckseite)
    {
        instance_destroy(); // Die richtigen Karten werden gelöscht
    }
    rueckseite = true;      // Wir drehen alle Karten um
}

aufgedeckteKarten = 0;      // Setzen den Wert auf 0, es ist ja keine Karte me
punkte += punkteBelohnung;  // Zur Belohnung gibt es Punkte
versuche++;                 // wir addieren dem Versuch einen Punkt auf.

// Der Stapel wird immer größer, bis das Maximum erreicht wurde
kartenStapel++;

if (kartenStapel < 5)
{
    stapel.image_index = kartenStapel;
}

kartenAufDemTisch -= 2;     // Nun sind zwei Karten weniger auf dem Tisch

aktion = true;              // Nun darf der Step wieder was machen
```

Wenn die Karten identisch ist, wird dieser Alarm vom Step aus aufgerufen. Mit *with (obj_memory_karte)* wird jede Karte gelöscht, die nicht ihre Rückseite zeigt. Das sind somit die beiden aufgedeckten, identischen Karten. Die Variable *aufgedeckteKarten* wird auf 0 gesetzt, die Punkte hoch gezählt und den Versuchen wird eine Zahl aufaddiert. Anschließend wird der Kartenstapel erhöht, zwei Karten auf dem Tisch abgezogen und mit *aktion = true;* dem **Step-Event** erlaubt, weiter zu machen.

Event Alarm 1

```
/// Karte wurde falsch aufgedeckt

with (obj_memory_karte)
{
    rueckseite = true;      // Wir drehen alle Karten um
}

aufgedeckteKarten = 0;      // Setzen den Wert auf 0, es ist ja keine Karte me
punkte -= punkteBestrafung; // Für den Fehlversuch ziehen wir Punkte ab
versuche++;                 // Und addieren dem Versuch einen Punkt auf.

aktion = true;              // Nun darf der Step wieder was machen
```

Auch wenn es etwas kürzer ist, ist das Prinzip fast gleich wie in **Alarm 0**. Statt die Karten zu löschen,

werden sie umgedreht und statt Punkte zu geben, werden die Strafpunkte abgezogen.

Event Alarm 2

```
game_restart();
```

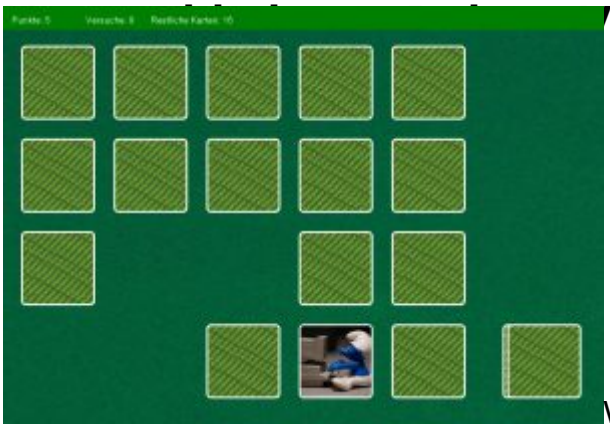
Das Spiel wird neu gestartet.

Event Draw

```
// Hintergrund unserer Punkteanzeige
draw_set_color(c_green);
draw_rectangle(0, 0, room_width, 50, 0);

// Punkte und Versuche
draw_set_color(c_white);
draw_set_font(fnt_punkte);
draw_text(20, 20, 'Punkte: ' + string(punkte));
draw_text(180, 20, 'Versuche: ' + string(versuche));

// Nach Spielende kommt eine Meldung

if (gameOver)
{
    // Der Hintergrund wird gezeichnet
    draw_set_color(c_green);
    draw_rectangle(300, 200, 760, 380, 0);
    // Nun kommt der Text
    draw_set_color(c_white);
    draw_set_font(fnt_punkte);
    draw_text(320, 240, 'Glückwunsch, Du hast alle Karten aufgedeckt!');
    draw_text(320, 300, 'Punkte: ' + string(punkte));
    draw_text(320, 340, 'Versuche: ' + string(versuche));
}
```

Ein paar Sachen müssen wir noch auf dem Bildschirm zeichnen. Letztlich besteht es aus der Punkteanzeige oben und dem Game Over am Ende des Spiels.

ersion



Wie oben beschrieben, ist die Deluxe-Version etwas

umfangreicher und beinhaltet nicht nur mehr Karten, sondern auch zwei Sets. **spr_kartenset01** und **spr_kartenset02** beinhalten jeweils 11 Frames (mit Rückseite) woraus sich ein Spielfeld mit 20 Karten ergibt. Damit auch alle Karten auf den Tisch passen, wurde der Raum auf 1300×900 vergrößert. Auch die Tischtextur wurde entsprechen vergößert (**bg_spieltisch**).

Die Anzahl der Objekte ist gleich geblieben, Unterschiede finden sich lediglich im Objekt **obj_controller**. Das Legen der Karten wurde in Alarm 3 ausgelagert, welches wir uns gleich noch genauer anschauen werden. Ein weiterer Unterschied im **Create-Event** ist, dass hier zufällig eines der beiden Sets ausgewählt wird.

Positionen und Anzahl der Karten wurden natürlich angepasst, das Prinzip ist aber identisch zur Basic-Version.

Event Alarm 3

```
/// Karten werden gelegt

wert++;
karte[wert] = ds_list_find_value(kartenspiel, wert - 1);
spielkarte = instance_create(xx + countI * abstand, yy + countA * abstand, obj
spielkarte.kartenwert = karte[wert];
spielkarte.set = set;
spielkarte.nummer = wert;
kartenAufDemTisch++;

// Reihen
if (countA < 3)
{
    countA++;
} else {
    countA = 0;
    countI++;
}

// Spalten
if (countI < 5)
{
    alarm[3]        = 0.15 * room_speed;                              // Der Ala
} else {
    aktion          = true;                                           // Alle Ka
    kartenausgabe   = false;                                          // Die Kar
    stapel          = instance_create(1080, 685, obj_kartenstapel); // Positio
}
```

Vom Prinzip her ist es ähnlich wie das Legen der Karten im **Create-Event** der Basic-Version. Hier ist aber der Alarm selbst die Schleife, weshalb wir auf die for-Schleifen verzichten können. Alle 0,15 Sekunden wird eine Karte gelegt. *countA* sind die Reihen, *countI* die Spalten. Wenn die letzte Karte auf dem Tisch liegt, wird der Stapel erzeugt und das Spiel freigegeben.

Wenn Du noch etwas Probleme mit dem Verständnis von Schleifen und Alarmen hast, solltest Du Dir das Schleifen-Tutorial einmal anschauen.

GameMaker Studio 2 Version

Die Version für GMS2 ist nahezu identisch mit der GMS 1.4 Deluxe-Version. Letztlich wurden nur Layer erstellt und die beiden Instanzen (Karte und Kartenstapel) werden in diesen Layern erzeugt. Der Befehl **instance_create** wurde durch **instance_create_layer** ersetzt.

Video

Downloads

Memory Basic-Version

1 Datei(en) 1.62 MB
Download

Memory Deluxe-Version

1 Datei(en) 4.29 MB
Download

Memory Deluxe-Version für GMS2

1 Datei(en) 6.65 MB
Download

Date Created
7. März 2017
Author
sven