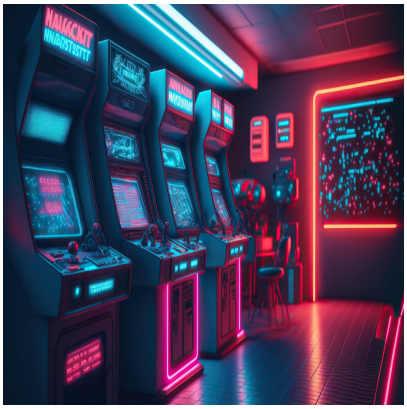

Selbstspielendes Breakout

Description

In meiner kleinen Demo “NO RUST“, die ich in GameMaker Studio entwickelt habe, wird beim Abspann-Scroller ein **Breakout**-Spiel im Hintergrund gezeigt. Dabei spielt die KI alleine. In diesem Tutorial möchte ich zeigen, wie einfach man dies programmieren kann.

Was ist “Breakout”?

Breakout ist ein legendäres Arcade-Spiel, das von Atari entwickelt und erstmals im Mai 1976 veröffentlicht wurde. Es wurde von Nolan Bushnell und Steve Bristow entworfen, wobei die Idee von Bushnells Wunsch inspiriert war, ein Spiel zu schaffen, das einfacher als das damals beliebte Pong war.

Der Spieler steuert einen Paddle am unteren Bildschirmrand, um einen Ball nach oben zu lenken, um damit Steine am oberen Bildschirmrand zu zerstören. Das Ziel ist es, alle Steine mit dem Ball zu treffen, ohne dass der Ball den Boden erreicht. Der Schwierigkeitsgrad steigt mit jedem Level, da die Steine sich anders anordnen und zusätzliche Herausforderungen hinzukommen. Meine bescheidene Version kannst du am Ende dieser Demo sehen:

Breakout hat sich zu einem Kultspiel entwickelt, das auch heute noch zahlreiche Fans auf der ganzen Welt hat. Es hat einen zeitlosen Reiz aufgrund seiner Einfachheit und Suchtfaktor. Die minimalistische Ästhetik, das schnelle Gameplay und die Herausforderung, jeden Stein zu treffen, machen es zu einem zeitlosen Klassiker. Darüber hinaus war Breakout wegweisend für das Genre der Arcade– und Videospiel-Industrie und hat viele weitere Spiele beeinflusst.

Das hat also mein Opa gespielt – was soll ich damit?

Es ist eine kleine, aber interessante Programmieraufgabe. Du kannst es zu einem eigenen Spiel umbauen oder für Demos, Credits oder als Hintergrund deines Menüs nutzen.

Von Breakout gibt es heute noch zahlreiche Varianten, die sogar verkauft werden, wie etwa **Alien Wall**.

Das Konzept

Für das ganze Spiel brauchen wir lediglich ein Objekt und ein kleines Skript. Schläger und Bricks werden per `draw_rectangle()` gezeichnet, wir verzichten also komplett auf Sprites. Die Bricks stecken wir in ein `ds_grid`. Das ist eine GameMaker-Datenstruktur, die einem 2D-Array entspricht.

In der Breite haben wir 24 Bricks, in der Höhe 8 Zeilen. Das kannst du bequem anpassen, ebenso alle Farben und Positionen. Außerdem kannst du noch Punkte und Leben hinzufügen. Im Beispiel beginnt der Ball einfach wieder bei der Startposition, sobald er unten rausfällt. Da die KI perfekt spielt, sollte das aber nicht passieren, außer du änderst die Ballgeschwindigkeit und die des Schlägers so, dass der Schläger nicht hinterher kommt. Bei einer sehr hohen Geschwindigkeit kann es sogar passieren, dass der Ball im Schläger stecken bleibt.

Unser Objekt

Wie du das Objekt nennst, spielt eigentlich keine Rolle. Bei mir heißt es schlicht `o_breakout`. Hier brauchen wir lediglich die Events

- Create
- Step
- Draw

Außerdem noch ein Skript mit dem Namen `get_brick_color()`.

Event Create

```
paddle_x = room_width / 2;
paddle_width = 70;
ball_x = room_width / 2;
ball_y = room_height - 200;
ball_speed_x = 8;
ball_speed_y = -ball_speed_x;
paddle_speed = ball_speed_x * 1.5;
brick_width = room_width / 24; // Breite basierend auf der Anzahl der Spalten
brick_height = 20;
brick_rows = 8;
brick_cols = 24; // Anzahl der Spalten
brick_offset = 100; // Oberer Abstand der Bricks
bricks = ds_grid_create(brick_cols, brick_rows);

// Bricks erstellen
var i, j;
for (i = 0; i < brick_cols; i++) {
    for (j = 0; j < brick_rows; j++) {
        bricks[# i, j] = true;
    }
```

```
}
```

Wie du sehen kannst, definieren wir am Anfang sehr viele Variablen. Hier kannst du bereits viele Veränderungen vornehmen. Die Höhe der Bricks beträgt 20 Pixel, die Breite ist abhängig von der Raumbreite. Am Ende erstellen wir noch unser `ds_grid` mit dem Namen `bricks` und füllen dies mit einer for-Schleife.

Event Step

```
// Begrenze den Zielwert innerhalb des Bildschirms
target_x = max(paddle_width / 2, min(room_width - paddle_width / 2, ball_x));

// Bewege den Schläger in Richtung des Zielwertes
if (paddle_x < target_x) {
    paddle_x += paddle_speed;
} else if (paddle_x > target_x) {
    paddle_x -= paddle_speed;
}

// Bewege den Ball
ball_x += ball_speed_x;
ball_y += ball_speed_y;

// Ball-Kollision mit den Wänden
if (ball_x <= 0 || ball_x >= room_width) {
    ball_speed_x *= -1;
}
if (ball_y <= brick_offset) {
    ball_speed_y *= -1;
}

// Wenn der Ball raus geht
if (ball_y >= room_height) {
    ball_x = room_width / 2;
 ball_y = room_height - 200;
}

// Ball-Kollision mit dem Schläger
if (ball_y >= room_height - 130 && abs(ball_x - paddle_x) < paddle_width / 2)
    ball_speed_y *= -1;
 // Hier könnte man noch einen Soundeffekt einfügen
}

// Ball-Kollision mit den Bricks
var brick_hit = false;
for (i = 0; i < brick_cols; i++) {
    for (j = 0; j < brick_rows; j++) {
        if (bricks[# i, j]) {
            var brick_x = i * brick_width + brick_width / 2;
            var brick_y = j * brick_height + brick_height / 2 + brick_offset;
            if (abs(ball_x - brick_x) < brick_width / 2 && abs(ball_y - brick_
                ball_speed_y *= -1;
                bricks[# i, j] = false;
```

```
                brick_hit = true;
                break;
            }
        }
    }
    if (brick_hit) {
  // Hier könnte man ebenfalls noch einen Soundeffekt einfügen
    }
}
```

Da ich alles in einem Objekt mache, musste ich auf die GameMaker-Funktionen für Kollision verzichten. Aber das ist eigentlich auch kein Problem. Im Step-Event steuern wir Ball, Schläger und regeln die Logik. Außerdem habe ich die Stellen gekennzeichnet, wo du Soundeffekte abspielen könntest.

Die Bricks werden im `ds_grid` nicht gelöscht, sondern lediglich auf `false` gestellt. Das ist diese Zeile: `bricks[# i, j] = false;`

Dadurch findet keine Kollision mehr statt und diese Bricks werden auch nicht mehr angezeigt.

Event Draw

```
draw_set_alpha(1);
draw_set_color(#021BD2); // Farbe für den Schläger

// Zeichne den Schläger
draw_rectangle(paddle_x - paddle_width / 2, room_height - 120, paddle_x + padd

draw_set_color(#FFFFFF); // Farbe für den Ball

// Zeichne den Ball
draw_circle(ball_x, ball_y, 10, false);

// Zeichne die Bricks
var i, j;
for (i = 0; i < brick_cols; i++) {
    for (j = 0; j < brick_rows; j++) {
        if (bricks[# i, j]) {
            var brick_x = i * brick_width + brick_width / 2;
            var brick_y = j * brick_height + brick_height / 2 + brick_offset;
            draw_set_color(get_brick_color(j)); // Farbe basierend auf der Zei
            draw_rectangle(brick_x - brick_width / 2, brick_y - brick_height /
            draw_set_color(#000000);
   draw_rectangle(brick_x - brick_width / 2, brick_y - brick_height / 2, brick
        }
    }
}
```

Jede der acht Zeilen hat eine eigene Farbe. Die Bricks werden doppelt gezeichnet, weil wir einmal das Brick und dann noch einen schwarzen Rahmen zeichnen. Das kannst du natürlich beliebig anpassen. Die Farben der Zeilen werden im Skript `get_brick_color(row)` geregelt.

Skript get_brick_color()

```
function get_brick_color(row) {
 switch (row){
  case 0:
   return #B40000;
  case 1:
   return #FD0909;
  case 2:
   return #B84F02;
  case 3:
   return #FF7815;
  case 4:
   return #00960F;
  case 5:
   return #0BFC23;
  case 6:
   return #D1D900;
  case 7:
   return #F8FF31;
  default:
   return #FFFFFF;
 }
}
```

Das ist sehr simpel. Je nach Zeilennummer gibt das Skript eine entsprechende Hex-Farbe zurück.

Das war es auch schon. Nun kannst du das Spiel nach deinen Vorlieben anpassen. Viel Spaß damit!

Weiterführende Links

Loading-Sequenz in GameMaker
Timing in GameMaker-Projekten
Projekt Snake
Projekt Tic-Tac-Toe – Teil 1
Shader-Programmierung 1: Grundlagen und Sprites

Date Created
29. März 2024
Author
sven