Tutorial – Pong-Klon

1 Einleitung

Ziel dieses Tutorials ist es, eine Einführung in die Programmierung mit der Canvas Game Engine (cge) zu geben. Die Möglichkeiten und Besonderheiten werden dabei anhand der Entwicklung eines kleinen Pong-Klons demonstriert.

2 Vorarbeit

Gestartet wird das Spiel über die Datei pong.html.
Hier wird zum einen die Canvas Game Engine mittels
`<script type="text/javascript" src="lib/cge.js"></script>`
eingebunden wie auch die Spiellogik an sich in der Datei pong.js.
Ebenfalls wird hier das canvas-Element definiert, in das dann gezeichnet wird. Über das div-Tag wird der Punktestand angezeigt. Dieser kann aber auch im Canvas-Element selbst angezeigt werden in dem ein TextObject der Szene hinzufügt.

3 Die Spiellogik

In diesem Kapitel werden die Funktionen der Canvas Game Engine aufgezeigt anhand des Pong-Klons. Wichtig ist hier die Datei pong.js um die Vorgehensweise nachzuvollziehen.

3.1 Initialisierung der Szene

Der erste Schritt ist eine Szene zu erstellen. Das wird wie folgt gemacht:
`var scene = new CGE_Scene('canvas', 640, 480);`
Dem CGE_Scene-Objekt wird als erster Parameter die ID des Canvas-Elements übergeben. Als zweiter und dritter Parameter folgt die Breite und Höhe der Zeichenfläche.
Die Szene müssen wir dann auch noch konfigurieren. Dazu setzen wir mittels
`scene.setSpeed(60);` eine Zeichenwiederholrate von 60 Hz. Das heißt pro Sekunde wird unsere Szene 60-mal aktualisiert.
Mit `scene.setBackground("url(img/pong_bg.jpg) ");` setzen wir ein Hintergrundbild.

Eine Besonderheit der Canvas Game Engine ist es, dass gewisse Funktionen vom Programmierer selbst definiert werden können. Erkennen tut man diese daran, dass solche Funktionen mit „on“ beginnen. Diese Funktionen werden dann intern immer wieder z.B. im Kontext des Renderns aufgerufen oder wenn ein bestimmtes Ereignis eintritt. Z.B. wird in **Zeile 43** die onStop-Methode definiert, um festzulegen was geschehen soll wenn die Szene gestoppt wird. In unserem Fall soll dann auch die Hintergrundmusik beendet werden.
Der Code dafür sieht wie folgt aus:

```
scene.onStop = function() {
  snd_bg.stop();
};
```

Nach diesem Schema werden auch alle weiteren Funktionen für die Szene aber auch für die Funktionen die an Objekte gebunden sind definiert.

Weitere Funktionen die hier definiert werden sind scene.onKeyDown = function(key) { } und scene.onKeyUp = function(key) { }. key ist dabei eine Konstante aus CGE_KEYBOARD und

beinhaltet die gedrückte Tastaturtaste, wie bspw. CGE_KEYBOARD.UP.
In unserem Beispiel merken wir uns welche Taste gedrückt wurde. Das ganze sieht dann so aus:

```
var pressed_up = false;
var pressed_down = false;
var pressed_w = false;
var pressed_s = false;
scene.onKeyDown = function(key) {
if (key == CGE_KEYBOARD.UP)
  pressed_up = true;
else if (key == CGE_KEYBOARD.DOWN)
  pressed_down = true;
else if (key == CGE_KEYBOARD.W)
  pressed_w = true;
else if (key == CGE_KEYBOARD.S)
  pressed_s = true;
};
scene.onKeyUp = function(key) {
if (key == CGE_KEYBOARD.UP)
  pressed_up = false;
else if (key == CGE_KEYBOARD.DOWN)
  pressed_down = false;
else if (key == CGE_KEYBOARD.W)
  pressed_w = false;
else if (key == CGE_KEYBOARD.S)
  pressed_s = false;
};
```

Mit den Tasten „Pfeil oben“ und „Pfeil unten“ soll der eine Spieler seinen Schläger bewegen und mit den Tasten „w“ und „s“ der andere Spieler seinen Schläger.

3.2 Definieren von Objekten

Die Canvas Game Engine stellt eine Reihe von Objekten zur Verfügung. Diese sind zum einen das CGE_Sound-Objekt, wie auch Objekte die gerendert werden. Dazu gehören das CGE_Sprite-Objekt, auf das gleich noch genauer eingegangen wird, das CGE_Circle()-, CGE_Rectangle()- und CGE_Text()-Objekt. Diese sind alle von CGE_SceneObject abgeleitet. Wer sich für die Objekt-Hierarchie näher interessiert, dem sei das Klassendiagramm im Ordner doc nahe gelegt.

Das Sound-Objekt:
Zuerst werde ich das Sound-Objekt näher beschreiben.
Ein Sound-Objekt wird duch CGE_Sound(<Pfad zur Audio-Datei>, Lautstärke) erstellt.
Audio-Dateien können wav-, ogg- Audio-Dateien sein. Die Lautstärke kann einen Wert zwischen 0.0 und 1.0 annehmen. Wenn die Lautstärke nicht angegeben wird, ist sie auf 1.0 gesetzt.
Funktionen die auf ein Sound-Objekt aufgerufen werden können sind play(Lautstärke), um den Sound abzuspielen, stop() um das Abspielen zu stoppen und pause(), um das Abspielen zu pausieren. SetVolume(Lautstärke) um die Lautstärke zu ändern.
Eine Eigenschaft die direkt gesetzt werden kann ist loop. Wenn sie auf true gesetzt ist, wird der Sound endlos wiederholt. Z.B. `my_sound.loop = false;`
Eine Funktion die selbst definiert werden kann ist onEnded() um das Verhalten zu programmieren wenn der Sound durchgelaufen ist.

In unserem Spiel sieht das ganze so aus:

```
var snd_collide = new CGE_Sound("snd/boing.wav");
```

```
var snd_ballout = new CGE_Sound("snd/ballout.wav");
var snd_win = new CGE_Sound("snd/trumpet.wav");
var snd_bg = new CGE_Sound("snd/pong_bg.ogg");
snd_bg.loop = true;
snd_bg.play(0.25);
scene.onStop = function() {
  snd_bg.stop();
};
```
Hier sehen wir auch, dass neben wav-Dateien auch ogg-Dateien unterstützt werden.

Das Sprite-Objekt
Ein Sprite-Objekt wird durch CGE_Sprite(<Pfad zur Bilddatei>, x, y, width, height) erstellt. Bild-Dateien können z.B. png- oder jpg-Dateien sein. Die (x,y)-Koordinate ist der Mittelpunkt des Sprites auf der Zeichenfläche. Width und height geben dabei die Größe des Sprite-Objekts an und stimmen **nicht** unbedingt mit der Breite und Höhe des Bildes überein. Über den sprite_index den ein Sprite-Objekt hat kann dann festgelegt werden, welcher Sprite gezeichnet werden soll. Dies sehen wir aber noch später am Pong-Ball.

Die Schläger
Wenn wir jetzt die Schläger der Spieler näher betrachten, so sehen wir in **Zeile 93** das Attribut solid. Wenn dieses auf true gesetzt ist, bedeutet das, dass das Objekt mit Anderen kollidieren kann. Dies ist wichtig, wenn der Ball den Schläger trifft.
Weiterhin müssen wir das Verhalten des Schlägers definieren, wenn eine Taste in dem Fall „nach unten“ oder „nach oben“ gedrückt wird.
An dieser Stelle möchte ich kurz den Ablauf beschreiben, wenn die Szene gerendert wird.
Als erstes wird für die Szene die Methode onBeforeRender() aufgerufen.
Danach wird für alle Objekte in der Szene zuerst die onBeforeRender()-Methode aufgerufen.
Dann wird das Objekt gerendert und anschließend wird die onAfterRender()-Methode aufgerufen.
Im nächsten Schritt wird für die Szene die onAfterRender()-Methode aufgerufen.
Jetzt folgt der Teil wo die Objekte bewegt werden. Als erstes wird wieder für die Szene die onBeforMove()-Methode aufgerufen.
Für alle Objekte werden dann die onBeforeMove()-Methode aufgerufen und dann das jeweilige Objekt bewegt.
Als nächstes werden Kollisionen behandelt. Dies geschieht in drei Schritten.
Für alle Objekte in der Szene wird im ersten Schritt alle Objekte für die eine onCollide()-Methode definiert ist mit den soliden Objekten auf Kollision getestet und bei denen wo eine stattfindet wird das Objekt mit dem Kollisionsobjekt in einem Array gespeichert.
Im nächsten Schritt werden alle soliden Objekte auf Kollision getestet. Die Objekte, bei denen es zu einer Kollision kommt, werden in einem Array gespeichert. Für diese wird dann die onCollide(Kollisionsobjekt)-Methode aufgerufen, der das Objekt übergeben wird, mit dem eine Kollision stattgefunden hat.
Im zweiten Schritt werden die Objekte die die onIntersectBoundary()-Methode definiert haben auf Kollision mit dem Rand getestet und bei Kollision wird das Objekt ebenfalls in dem Array gespeichert.
Im dritten und letzten Schritt werden die Objekte, welche die onOutsideScene()-Methode definiert haben darauf getestet, ob sie die Szene verlassen haben und falls ja wird direkt die onOutsideScene()-Methode des Objektes aufgerufen.
Anschließend werden für alle Objekte die in dem Array für die Kollisionen gespeichert sind entweder die onCollide() und/oder die onIntersectBoundary()-Methode aufgerufen.
Zum Schluss wird noch für alle Objekte die onAfterMove()-Methode aufgerufen und auch die onAfterMove()-Methode für die Szene.
Nun aber zurück zu unserem Pong-Klon, wo einige Methoden die im Render-Prozess aufgerufen

werden, definiert sind.
In unserem Pong-Klon wollen wir, bevor der Schläger bewegt wird, die neuen Koordinaten setzen, also wohin der Schläger bewegt wird. Da wir in der Szene vorher schon die gedrückte Taste ermittelt und uns gemerkt haben (s. onKeyUp() und onKeyDown()), können wir hier die y-Koordinate erhöhen oder vermindern, indem wir die eigens erstellte Instanzvariable „dir“ hinzu addieren.
Die brauchen wir hier, da der Schläger zum einen beschleunigen soll und auch an der oberen und unteren Wand zurückprallen soll. Das ganze sieht dann so aus:

```
var player_red = new CGE_Sprite("img/pong_red.png", 100, scene.height/2,
10, 60);
player_red.dir = 0;
player_red.onBeforeMove = function() {
if (pressed_w)
  this.dir -= 1;
else if (this.dir < 0) {
  this.dir += 2;
  if (this.dir > 0) this.dir = 0;
}
if (pressed_s)
  this.dir += 1;
else if (this.dir > 0) {
  this.dir -= 2;
  if (this.dir < 0) this.dir = 0;
}
this.y += this.dir;
if (this.y > 455) {this.y = 455;this.dir = -this.dir;}
if (this.y < 25) {this.y = 25;this.dir = -this.dir;}
};
```

Wenn das nicht gewünscht ist hätte auch folgender Code gereicht:

```
player_blue.onBeforeMove = function() {
      if (pressed_up) {
            this.y -= 5;
      } else if (pressed_down) {
            this.y += 5;
      }
      if (this.y < 25) {
            this.y = 25;
      } else if (this.y > 455) {
            this.y = 455;
      }
};
```

Hier erhöhen oder vermindern wir einfach den y-Wert des Schlägers um fünf Pixel und stellen sicher, dass der Schläger nicht den oberen oder unteren Rand verlässt.
Nicht vergessen dürfen wir den Schläger, also das CGE_Sprite-Objekt der Szene hinzuzufügen. Dafür gibt es die Methode addObject(objekt), also in unserem Beispiel:

```
scene.addObject(player_red);
```

Das was wir für den einen Schläger gemacht haben übernehmen wir auch genauso für den anderen Schläger vom zweiten Spieler.

Der Ball
Aber mit den Schlägern alleine haben wir noch keinen Pong-Klon. Dazu fehlt noch der Ball mit dem wir weitere interessante Elemente der Canvas Game Engine kennenlernen werden.

Beginnen werden wir mit Kollisionen. Da Grafiken rechteckig sind, wird CGE_Sprite-Objekten standardmäßig ein rechteckiges Kollisionsobjekt zugewiesen. Dies kann aber wie in unserem Fall nicht erwünscht sein, da der Ball obwohl das Bild rechteckig ist, eigentlich das Kollisionsverhalten eines CGE_Circle-Objekts haben soll.
Dies erreichen wir einfach dadurch, indem wir dem Ball als Kollisionsobjekt ein CGE_Circle-Objekt zuweisen und zwar mit den x- und y-Koordinaten 0,0 und dem Radius, der der halben Breite des Bildes entspricht.
Weiterhin sagen wir mit `ball.solid = true;` das auch für den Ball auf Kollisionen geprüft werden soll und mit `ball.speed = 3;` geben wir dem Ball noch eine Geschwindigkeit von drei.

Bevor der Ball bewegt wird, müssen wir in der onBeforeMove()-Methode erst einmal die neuen Koordinaten festlegen. Dabei müssen wir vier Fälle behandeln und zwar was geschehen soll, falls der Ball den linken, rechten, unteren oder oberen Rand berührt. Wenn wir davon ausgehen, dass die linke Spielfeldseite dem roten Spieler gehört und die rechte Spielfeldseite dem Blauen, dann müssen wir, wenn der Ball den linken Rand berührt, zum einen den Punktestand für Spieler Blau erhöhen und einen Sound abspielen.
```
if (this.x - this.width/2 <= 0) {
      score_blue++;
      snd_ballout.play();
      document.getElementById('score').innerHTML = score_red+' :
'+score_blue;
```

Als nächstes prüfen wir, ob der maximale Punktestand erreicht ist und wenn ja, wird angezeigt, dass Spieler Blau gewonnen hat. Weiterhin wird der Sound für den Sieg abgespielt und die Szene wird gestoppt und sonst wird der Ball wieder in die Mitte gesetzt und mit einer Geschwindigkeit von drei in Richtung roter Spieler bewegt.
```
if (score_blue >= maxScore) {
    document.getElementById('score').innerHTML = "Sieg für Blau!";
    snd_win.play();
    scene.stop();
} else {
    this.setPosition(scene.width/2, scene.height/2);
    this.setMotion(3, 180);
}
```

Wenn der Ball hingegen den rechten Spielfeldrand berührt machen wir das gleiche nur für Spieler Rot. Der Code sieht wie folgt aus:
```
if (this.x + this.width/2 >= scene.width) {
  snd_ballout.play();
  score_red++;
  document.getElementById('score').innerHTML = score_red+' :
'+score_blue;
  if (score_red >= maxScore) {
      document.getElementById('score').innerHTML = "Sieg für Rot!";
      snd_win.play();
      scene.stop();
  } else {
```

```
    this.setPosition(scene.width/2, scene.height/2);
    this.setMotion(3, 0);
  }
}
```

Zum Schluss definieren wir noch, wie der Ball sich verhalten soll, wenn er den oberen oder den unteren Rand berührt. Hier drehen wir einfach den Speed in y-Richtung um und spielen den Sound für die Kollision ab.

```
if (this.y - this.height/2 <= 0 || this.y + this.height/2 >=
scene.height) {
  this.setYSpeed(-this.getYSpeed());
  snd_collide.play();
}
```

In der onCollide()-Methode definieren wir was geschehen soll, wenn der Ball mit irgendeinem anderen Objekt der Szene kollidiert. Dabei wird der Methode das Objekt mitgegeben, mit dem unser Ball kollidiert ist. Interessant ist hier was passieren soll, wenn der Ball mit einem Schläger kollidiert.
Dazu ändern wir zum einen die Geschwindigkeit in x- wie auch in y-Richtung, mit `this.setYSpeed((this.y – object.y)/5);` und t`his.setXSpeed(-this.getXSpeed());`
Dabei wird die Geschwindigkeit in y-Richtung abhängig davon gesetzt wie der Ball zum Schläger in y-Richtung steht. Das dann noch geteilt durch fünf, weil der Ball sich sonst zu steil in y-Richtung bewegt. Die Geschwindigkeit in x-Richtung dreht sich einfach um, damit der Ball in die entgegengesetzte Richtung läuft.
Als nächstes prüfen wir, welcher Schläger getroffen wurde und wechseln das Bild für den Ball. Dies erreichen wir dadurch, dass wir den „sprite_index“ auf zwei setzen. Um das nachzuvollziehen sollten wir uns das Bild, welches bei der Erzeugung des Sprite-Objektes angegeben wird, näher betrachten. Dazu gehen wir in den Ordner „img“ und öffnen die Datei pong_ball.png. Wir sehen, dass dieses Bild drei Sprites der Größe 10x10 Pixel enthält. Diese Größe mussten wir auch bei der Erstellung des Sprite-Objektes angeben. Wenn wir jetzt den „sprite_index“ auf zwei setzen, heißt dass nichts anderes, als das das dritte Sprite gezeichnet werden soll, also der rote Ball. Wenn hingegen das Kollisionsobjekt der blaue Schläger ist, soll der zweite Sprite mit dem „sprite_index“ eins gezeichnet werden. Im Code sieht das dann so aus:

```
if (object == player_red) {
  this.sprite_index = 2;
} else if (object == player_blue) {
  this.sprite_index = 1;
}
```

Den Ball lassen wir dann in x-Richtung entgegengesetzt wieder abprallen.
`this.setXSpeed(-this.getXSpeed());`
Dann prüfen wir noch ob die Geschwindigkeit des Balles kleiner als 20 ist und wenn ja, erhöhen wir ihn, um die Schwierigkeit mit zunehmendem Ballwechsel zu erhöhen.

```
if (this.speed < 20)
  this.speed = current_speed + 1;
```

Als letztes Spielen wir noch den Sound für eine Kollision zwischen Schläger und Ball ab, wenn der Ball einen Schläger getroffen hat. Dies macht folgender Code:

```
if (object == player_red || object == player_blue)
  snd_collide.play();
```

Und zuletzt nicht vergessen den Ball mit `scene.addObject(ball);` der Szene hinzuzufügen.

Der Abschluss

So das war jetzt ein ganzes Stück Arbeit. Und nun bleibt nichts weiter zu tun als mit `scene.start();` zu sagen, dass die Szene gestartet werden soll. Der komplette Code steht in der Datei pong.js im Ordner „js“.

Zum Schluss möchte ich noch anmerken das in diesem Tutorial nicht alle Objekte im Detail besprochen wurden. Wer eine vollständige Übersicht über alle Objekte und Funktionen haben möchte, sollte sich auf jeden Fall die Dokumentation im Ordner „doc“ anschauen.