ORWB Selbständige Arbeit

https://github.com/SuicidePandax3/orwb1

Dokumentation

IMS, KFTG, FS19

Von Marlon Zimmermann

Inhalt

[1 Management Summary 3](#_Toc2332627)

[2 Rahmenbedingungen 3](#_Toc2332628)

[2.1 Projektvertrag 3](#_Toc2332629)

[2.2 Organisation 3](#_Toc2332630)

[2.3 Must-Have & Nice-to-Have 4](#_Toc2332631)

[2.4 Wieso Object Orientated JavaScript und p5.js? 5](#_Toc2332632)

[3 Realisation 6](#_Toc2332633)

[3.1 Ausarbeitung Konzept 6](#_Toc2332634)

[3.2 Einarbeitung p5.js 6](#_Toc2332635)

[3.2.1 Setup und Draw 6](#_Toc2332636)

[3.3 Einarbeitung Object Orientated JS 7](#_Toc2332637)

[3.4 Website Framework 7](#_Toc2332638)

[3.5 Spielecharakter 8](#_Toc2332639)

[3.5.1 Design 8](#_Toc2332640)

[3.5.2 Eigenschaften 8](#_Toc2332641)

[3.5.3 Implementation Charaktermodell 8](#_Toc2332642)

[3.5.4 Steuerung 8](#_Toc2332643)

[3.6 Game 8](#_Toc2332644)

[3.6.1 Zustand 0 8](#_Toc2332645)

[3.6.2 Zustand 1 8](#_Toc2332646)

[3.6.3 Zustand 2 9](#_Toc2332647)

[3.6.4 Zustand 3, 9](#_Toc2332648)

[3.6.5 Zustand 4 9](#_Toc2332649)

[3.6.6 Start Bildschirm 9](#_Toc2332650)

[3.6.7 Pause Bildschirm 9](#_Toc2332651)

[3.6.8 Game-Over Bildschirm 10](#_Toc2332652)

[3.6.9 Sieg Bildschirm 10](#_Toc2332653)

[3.7 Gegner 10](#_Toc2332654)

[3.7.1 Design 10](#_Toc2332655)

[3.7.2 Implementation 11](#_Toc2332656)

[3.8 Levels 12](#_Toc2332657)

[3.8.1 Design 12](#_Toc2332658)

[3.8.2 Level 1 12](#_Toc2332659)

[3.8.3 Level 2 12](#_Toc2332660)

[3.8.4 Level 3 12](#_Toc2332661)

[4 Kontrolle 12](#_Toc2332662)

[5 Bugfixing 12](#_Toc2332663)

[6 Deployement 12](#_Toc2332664)

[7 Auswertung 12](#_Toc2332665)

[8 Abgabe 12](#_Toc2332666)

[9 Glossar 12](#_Toc2332667)

[10 Quellenverzeichnis 12](#_Toc2332668)

[11 Literaturverzeichnis 12](#_Toc2332669)

[12 Anhang 12](#_Toc2332670)

[13 Sourcecode 12](#_Toc2332671)

# Management Summary

Orwb ist ein 2D-Platfom Spiel (wie z.B. Donkey Kong) für den Webbrowser. Ziel des Spiels ist es, das Portal in das nächste Level zu erreichen, ohne alle Lebenspunkte zu verlieren. Der Spielercharakter ist neben dem Portal, die einzige Lichtquelle im Spiel. Die Lebenspunkte bestimmen die Leuchtkraft des Spielecharakters und somit die Distanz des Lichtkegels, welcher Orwb umgibt. Wenn Orwb Schaden durch Gegner oder Fallen erleidet, sinken seine Lebenspunkte und somit die Distanz des Lichtkegels. So wird das Spiel progressive erschwert, da weniger von der Spieleumgebung sichtbar ist. Um das Spiel weiter zu erschweren, hat der Spiele nur begrenzt Zeit, das Portal zu erreichen.

Das Spiel ist fertig, wenn:

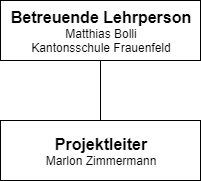
* der Spieler das Portal des dritten Levels erreicht hat
* Orwb alle Lebenspunkte verloren hat
* Orwb durch einen Spalt ins Leere fällt
* die Zeit vorüber ist

Das Spiel ist ein Browsergame. Das heisst es soll über eine Webseite aufgerufen und gespielt werden können. Orwb soll auf allen gängigen Internet-Browsern für PCs funktionieren.

# Rahmenbedingungen

## Projektvertrag

## Organisation



## Must-Have & Nice-to-Have

Must-Haves Charakter bewegen  
Charakter beleuchtet Umgebung  
Leuchtkraft sinkt oder steigt je nach Lebenspunkten  
Gegner greifen an und verursachen Schaden  
Portal ins nächste Level  
Mindesten 3 Spielbare Level  
3 Versuche pro Spiel  
Standalone mittels Webhosting(000Webhost)

Nice-to-Haves Spieler kann Sterne (Punkte) sammeln  
Zeit und Punkte werden angezeigt  
Spieler kann sich in eine Rangliste eintragen  
Power-Ups  
Sound-Effekte

## Wieso Object Orientated JavaScript und p5.js?

P5.js ist eine frei verfügbare Processing Library. Processing ist in Begriff für das künstlerische Programmieren von Graphiken für den Web-Browser. Es ist eine Bewegung, welche in den letzten Jahren viele Mitglieder gewonnen hat. Ich bin über ein Youtube-Video des Programmierers Daniel Shiffman (The Coding Train) auf dieses Library gestossen und habe mich auf der Stelle verliebt. Während der Freizeit habe ich verschiedene Projekte wie die Matrix Animation oder Break-Out selbst mit P5.js programmiert und grossen Gefallen daran gefungen.

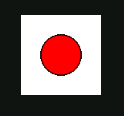
Für mich war von Anfang an klar, dass ich Objekt orientiert programmieren muss. Ein Spiel wie Orwb hat sehr viele Objekte wie zum Beispiel Tiles, auf denen der Spiele Charakter steht. Jedes dieser Plattformen einzeln zu programmieren wäre unnötigen Aufwand und nicht zielführend.

In OOP-Js gibt es sogenannte Classes, welche den Bauplan für das Objekt darstellen. In diesen können auch ganz einfach Methoden erstellt und über das Objekt aufgerufen werden. Variablen werden mit this.variable im Constructor instanziiert und mit this.variable in der Funktion der Class aufgerufen:

Code:

**let** ball;  
//setup function  
**function** setup(){  
 //create canvas  
 **let** canvas = createCanvas(100, 100);  
 //relocates canvas to the p5 html div  
 canvas.parent('p5');  
 //background color  
 background(255);  
 //ball  
 ball = **new** Ball(width/2, height/2);  
}  
//draw function  
**function** draw(){  
 //calls display function of ball  
 ball.display();  
}  
  
**class** Ball{  
 constructor(x, y){  
 **this**.positionX = x;  
 **this**.positionY = y;  
 **this**.radius = 50;  
 **this**.color = color(255, 0, 0)  
 }  
  
 display(){  
 fill(**this**.color);  
 ellipse(**this**.positionX, **this**.positionY, **this**.radius, **this**.radius);  
 }  
}

Ergebnis:



# Realisation

## Ausarbeitung Konzept

Die Idee des Spiels ist mir bereits Anfangs IMS gekommen, als Ich mit Unity ein Game entwickeln wollte. Ich habe aber schnell gemerkt, dass Ich zu wenig Erfahrung und zu wenig Zeit für ein solches Projekt habe. Als dann die Beantragung der Selbstständigen Arbit anstand, habe ich mich dazu entschieden, ORWB als SA zu realisieren.

## Einarbeitung p5.js

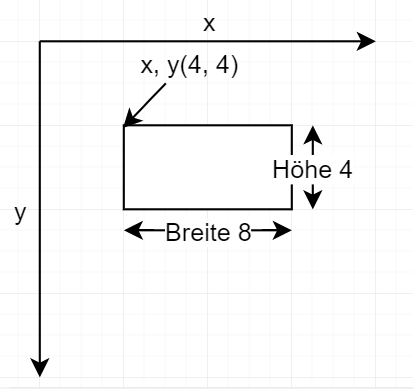
Das einarbeiten in P5 war für mich kein Problem. Es ist eine sehr einfache und intuitive Library, entwickelt für das Programmieren von Grafiken. Die Youtube-Tutorials von Jeff Shiffman haben mir hierbei sehr geholfen.

### Setup und Draw

Es gibt eine setup und eine draw Funktion. Setup() wird zu Beginn des Scripts aufgerufen. Draw() als Loop 60 Mal pro Sekunde (per Default) aufgerufen. In dieser Funktion passiert die ganze Animation und Interaktion.

**function** setup(){  
 //create canvas  
 **let** canvas = createCanvas(100, 100);  
 //relocates canvas to the p5 html div  
 canvas.parent('p5');  
 //background color  
 background(255);  
 //ball  
}  
//draw function 60 fps  
**function** draw(){  
 //calls display function of ball  
 fill(0);  
 rect(10, 10, 10, 10);  
}

Zuerst muss ein Canvas, also der Bereich in welcher das Programm auf der Seite abgebildet wird, erstellt werden. Dies geschieht mit createCanvas(width, height);. Um etwas darzustellen, muss man wissen, der Punkt [0, 0] oben links ist. Die Höhen und Breiten der x und y Koordinaten werden in Pixel angegeben.



In diesem Beispiel habe ich den Canvas der DIV „p5“ auf meiner Seite zugewiesen, um sie am gewünschten Ort darzustellen.

Mit background(255); wird der Hintergrund, also der Canvas, weiss gefärbt.

In der Draw() Funktion wird mit fill(0); die nachfolgende Form eingefärbt. Mit rect(x-Position, y-Position, Höhe, Breite); wird ein Rechteck mit den übergebenen Parametern erstellt.

## Einarbeitung Object Orientated JS

Die Einarbeitung in Objekt orientiertes Programmieren von JS war nicht sehr schwierig. Ich habe hierfür ein paar Youtube-Tutorials von The Coding Train (<https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw>) geschaut und mein Projekt nach und nach Programmiert.

## Website Framework

Das Website Framework habe ich als erstes erledigt. Ich musste hierfür die nötigen Bootstrap, Fontawesome und P5.js Dateien einbilden und ich wollte einen ausfahrbaren Navigationsbereich.

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Orwb</title>  
 <link rel="shortcut icon" type="image/png" href="img/orwb/orwb\_static0.png">  
  
 <!--bootstrap-->  
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"  
 integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">  
 <!--p5.js-->  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/p5.min.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/addons/p5.dom.min.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.1/addons/p5.sound.min.js"></script>  
 <script src="js/collide2D.js"></script>  
  
 <!--Font Awesome JS-->  
 <script defer src="https://use.fontawesome.com/releases/v5.0.13/js/solid.js"  
 integrity="sha384-tzzSw1/Vo+0N5UhStP3bvwWPq+uvzCMfrN1fEFe+xBmv1C/AtVX5K0uZtmcHitFZ"  
 crossorigin="anonymous"></script>  
 <script defer src="https://use.fontawesome.com/releases/v5.0.13/js/fontawesome.js"  
 integrity="sha384-6OIrr52G08NpOFSZdxxz1xdNSndlD4vdcf/q2myIUVO0VsqaGHJsB0RaBE01VTOY"  
 crossorigin="anonymous"></script>  
 <!--CSS-->  
 <link rel="stylesheet" type="text/css" href="css/style.css">  
</head>  
<body>  
<!--Sidebar-->  
<div id="mySidebar" class="sidebar">  
 <a href="about.html">About</a>  
 <a href="contact.html">Contact</a>  
</div>  
  
<div id="main">  
 <button class="openbtn" onclick="nav()"><i class="fas fa-bars"></i></button>  
  
 <div id="container" class="container">  
 <br>  
 <!--Canvas Area-->  
 <div id="p5">  
 <!--Game Script-->  
 <script src="js/game.js"></script>  
 </div>  
  
 </div>  
</div>  
  
  
<!--CDN-->  
<script src="https://code.jquery.com/jquery-3.3.1.js"  
 integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="  
 crossorigin="anonymous"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"  
 integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"  
 crossorigin="anonymous"></script>  
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"  
 integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"  
 crossorigin="anonymous"></script>  
  
<script src="js/welcome.js"></script>  
  
</body>  
</html>

## Spielecharakter

### Design



### Eigenschaften

### Implementation Charaktermodell

### Steuerung

#### Collision Detection

#### Jump

## Game

Das Game ist in verschiedene Klassen unterteilt. Die Loop wird durch die Draw Function von p5.js übernommen. In dieser wird die Funktion game.checkState() aufgerufen, welche überprüft, in welchem Zustand sich das Spiel gerade befindet. Es gibt 5 verschiedene Zustände:

### Zustand 0

Das Spiel wurde gerade gestartet. In diesem Fall soll der Welcome Screen angezeigt werden.

### Zustand 1

Das Spielt läuft. In der Loop wird nun der Aktuelle Level (Spielecharakter, Blöcke, Punkte und Portal) aufgerufen und abgebildet.

### Zustand 2

Das Spiel wird durch die Escape Taste pausiert. Der Pause-Screen wird angezeigt.

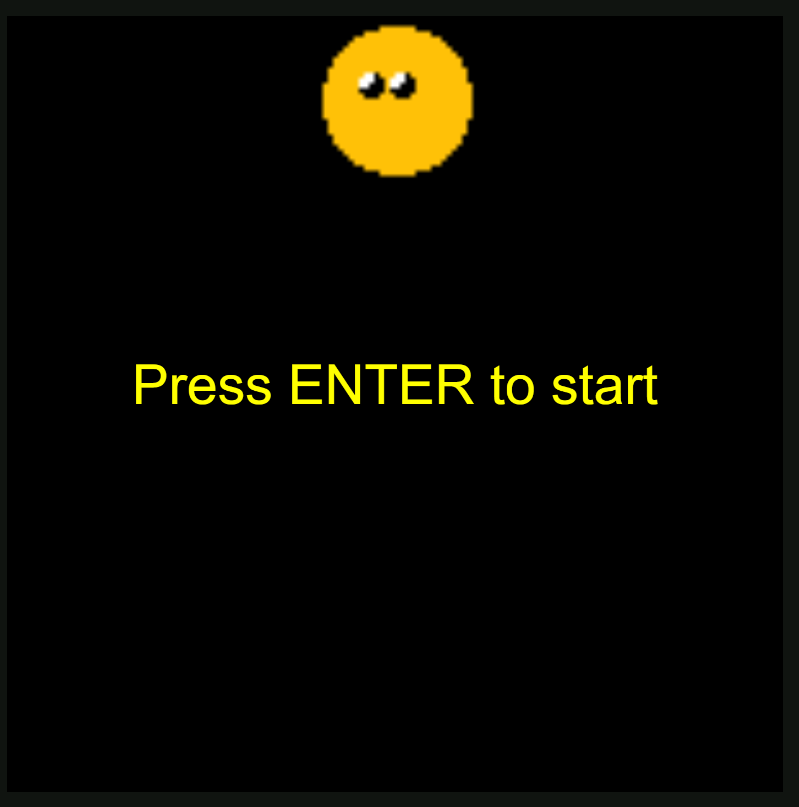
### Zustand 3,

Das Spielt wird fortgefahren. Der Zustand wechselt wieder auf 1.

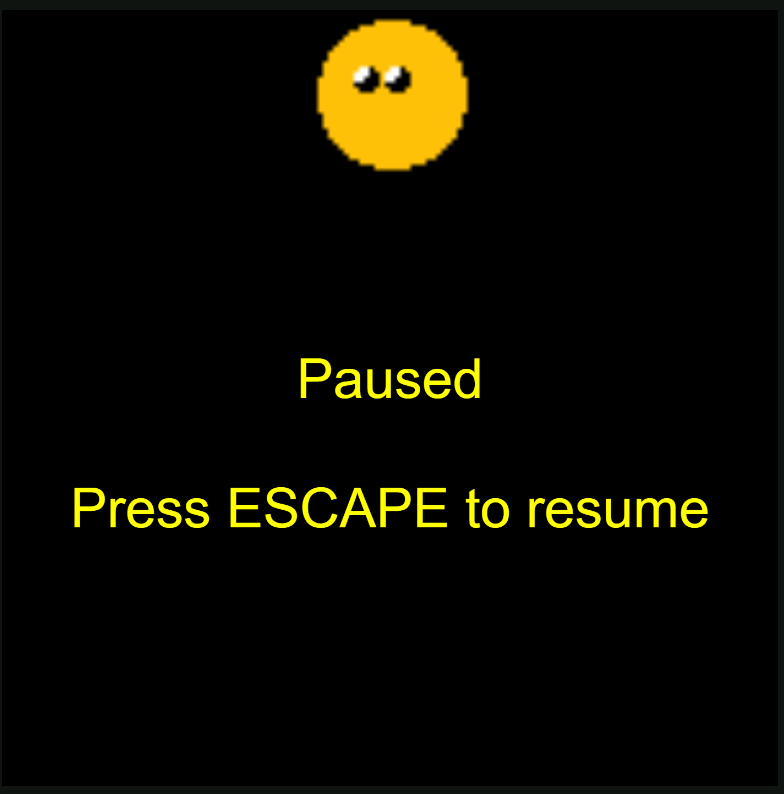
### Zustand 4

Das Spiel ist zu ende. Entweder durch Tod des Spielecharakters, oder weil dieser den letzten Level bezwungen hat.

### Start Bildschirm



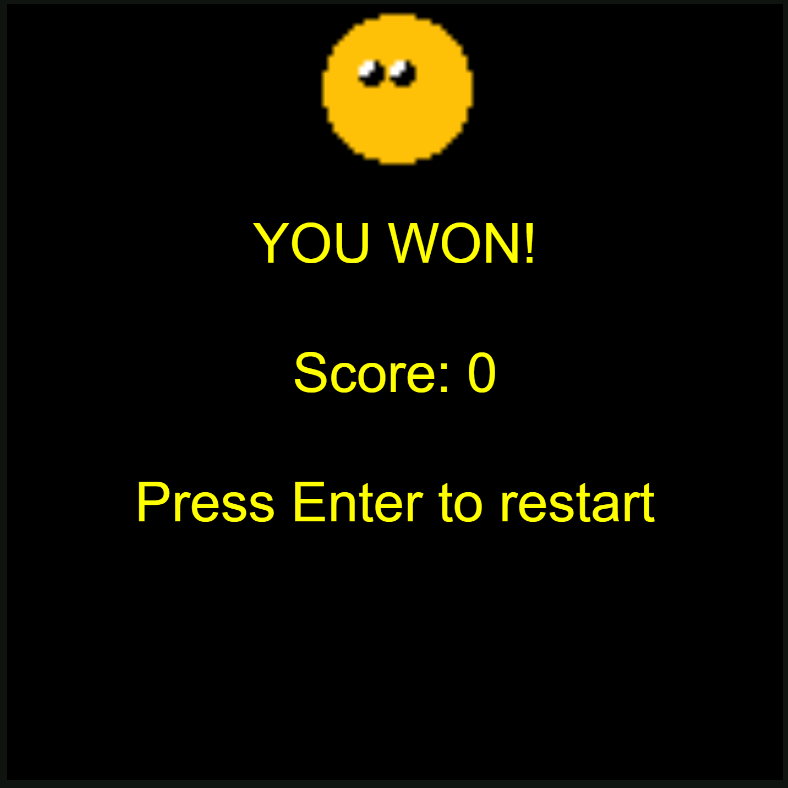
### Pause Bildschirm



### Game-Over Bildschirm



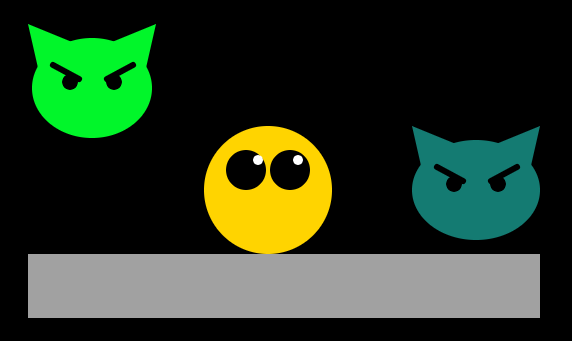
### Sieg Bildschirm



## Gegner

### Design

Es gibt 2 Arten von Gegner: Die statische und die fallende Katze. Sie haben beide das gleiche Design. Die Farbe wird jedes Mal zufällig zugewiesen.



### Implementation

Für Die statische Katze musste ich nur das aussehen und eine Kollision programmieren. Falls Orwb mit der Katze kollidiert, soll er einen Schadenspunkt erhalten. Bei der fallenden Katze habe ich einfach eine Abwärtsbewegung eingebaut. Wenn die Katze den unteren Rand des Spielfeldes erreich, soll sie einfach wieder erscheinen.

## Levels

### Design

### Level 1

### Level 2

### Level 3

# Kontrolle

# Bugfixing

# Deployement

# Auswertung

# Abgabe

# Glossar

# Quellenverzeichnis

# Literaturverzeichnis

# Anhang

# Sourcecode