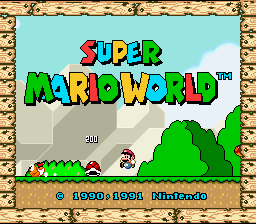
**SuperMarioVT – Projekt**

**Spielentwickler: Jeganivas Ulaganathan & Vassilios Tsiotras**

**Klasse: BK-TA14**



**Inhaltsverzeichnis**

Abbildungsverzeichnis 3

(Abkürzungsverzeichnis) 4

1 Einleitung 5

2 Planungsphase 6

2.1 Terminplanung 6

2.4 Ablaufplanung 6

3 Durchführungsphase 7

3.1 Aufgetretene Fehler 7

4 Projektabschluss 8

4.1 Projektübergabe 8

4.2 Abnahmeprotokoll 8

4.3 Projektbewertung 8

Quellenverzeichnis 9

# Abbildungsverzeichnis

Abbildung 1: Hintergrund-Büsche 7

Abbildung 2:Hintergrund Untergrundlevel 7

Abbildung 3: Hintergrund Levels 7

Abbildung 4: Sprite für Enemy-Pilz 8

Abbildung 5: Sprite für Coins 8

Abbildung 6: Sprite für Fragezeichen 8

Abbildung 7: Sprite für Pipes 8

Abbildung 8: Sprite für Mario 8

Abbildung 9: Sprite für Boden 9

Abbildung 10: Sprite für Ziel 9

Abbildung 11: Sprite für Rocket 9

Abbildung 12: Level-GUI Sprite Editor 9

Abbildung 13: Components Player 10

Abbildung 14: Animation-Controller Player 11

Abbildung 15: Components Enenemy Pilz 12

Abbildung 16: Components Coins 13

Abbildung 17: Layout Startmenü 14

Abbildung 18: Levels bilden/einbeziehen in Unity 14

# Einleitung

Aufgabe des Projekts ist es ein Super Mario Spiel für einen Spieler bereitzustellen, der folgende Features besitzt:

• Spieler bekommt eine Startseite zum Spielstarten

• Spieler bekommt ein Super Mario Figur als sein Avatar

• Spieler bewegt sich mit den Pfeiltasten:

1. Leertaste=Zum springen

2. Linkstaste=Schritt nach links

3. Rechtstaste=Schritt nach rechts

• Spieler muss Coins sammeln, dies ist auch gleichzeitig sein Highscore ist

• Spieler wird mit „Gegenständen“ konfrontiert, die er ausweichen muss.

Ein Zusammenstoß führt zum Game Over.

• Es gibt ein Rohr indem der Spieler reingehen kann um Coins zu sammeln.

• Erreicht der Spieler das Ziel gewinnt der Spieler.

Um dieses Spiel zu programmieren verwenden wir die Umgebung Unity, weil im Unity die Spielumgebung herzustellen wesentlich einfach ist. Die Programmiersprache ist C#.

Da wir ein großes Interesse am Programmieren besitzen, entschieden wir uns direkt für ein Programmier-Projekt in der Richtung Game-Development. Da fiel uns das Klassikerspiel „SuperMario“ ein und dachten dies wäre ein passendes Projekt, das sehr umfangreich ist.

# Planungsphase

## Terminplanung

Abgabe Themenvorschläge: 17.10.2018

Themen besprechen: 24.04.2018

Abgabe Projektanträge: 07.11.2018

Rückgabe und Besprechung Projektanträge: 14.11.2018

Abgabetermin Endfassung Projektanträge: 21.11.2018

Projektgenehmigung: 28.112018

Ende Planungsphase: 05.12.2018

Ende der Planungsphase: 05.12.2018

Beginn der Projekt Durchführungsphase: 12.12.2018

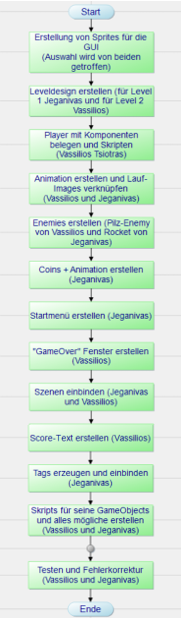
Ende Projekt Durchführungsphase: 27.03.2018

Abgabetermin Projektdokumentation: 10.04.2018

Bekanntgabe Dokumentationsergebnisses 17.05.2018

Projektpräsentation 27.05.2018

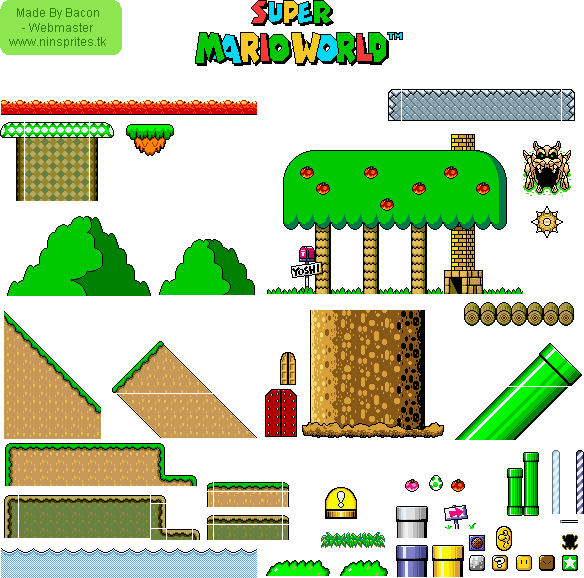
## Ablaufplanung



# Durchführungsphase

Schritt 1: Sprites

Für das Projekt SuperMario, das in einer Game-Engine, Unity, erstellt wird braucht man natürlich zuerst Objekte, Hintergründe etc. Die aus Images bestehen. Auf der Seite <https://www.spriters-resource.com/snes> haben wir die einzelnen Images gefunden:



Hintergrund-Büsche

Abbildung

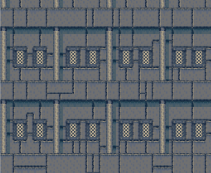


Abbildung 3

Abbildung

Hintergrund für Untergrundlevel Hintergrund für die Levels



Abbildung 4

Coins um die Animation zu erstellen

Abbildung 5

Enemy-Pilze um die

Animation zu erstellen



Abbildung 7



Abbildung 6

Pipes

Fragezeichen um die Animation zu erstellen

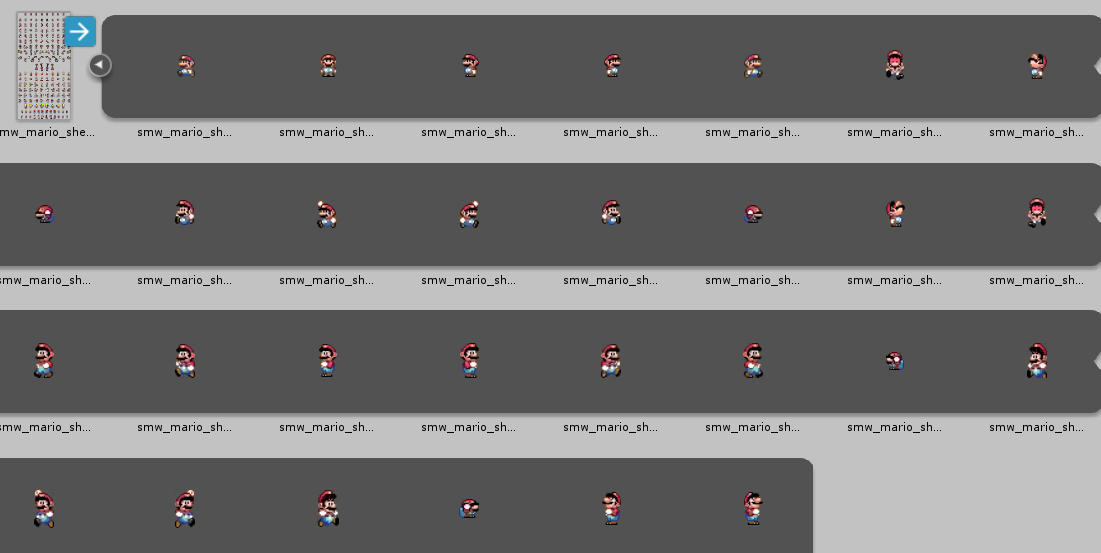


Abbildung 8

SuperMario um die Animationen zu erstellen.



Abbildung 10

Abbildung 9

Ziel

Boden



Abbildung 11

Rackete+Erstellung von Animation

Schritt 2: Level-GUI

Als nächstes haben wir unsere Grafische Oberfläche für unsere Levels erstellt. Müssten dafür die Images als Sprites „erzeugen“. Dazu erstellt man dank Unity ein leeres GameObject und fügt das Bild hinein und öffnet den Sprite Editor und schneidet sein gewünschtes Bild zurecht und fügt es in der Spielumgebung ein. Und fügt natürlich bei Boden, Pipes etc. Colliders hinzu damit der Spieler nicht runterfällt.

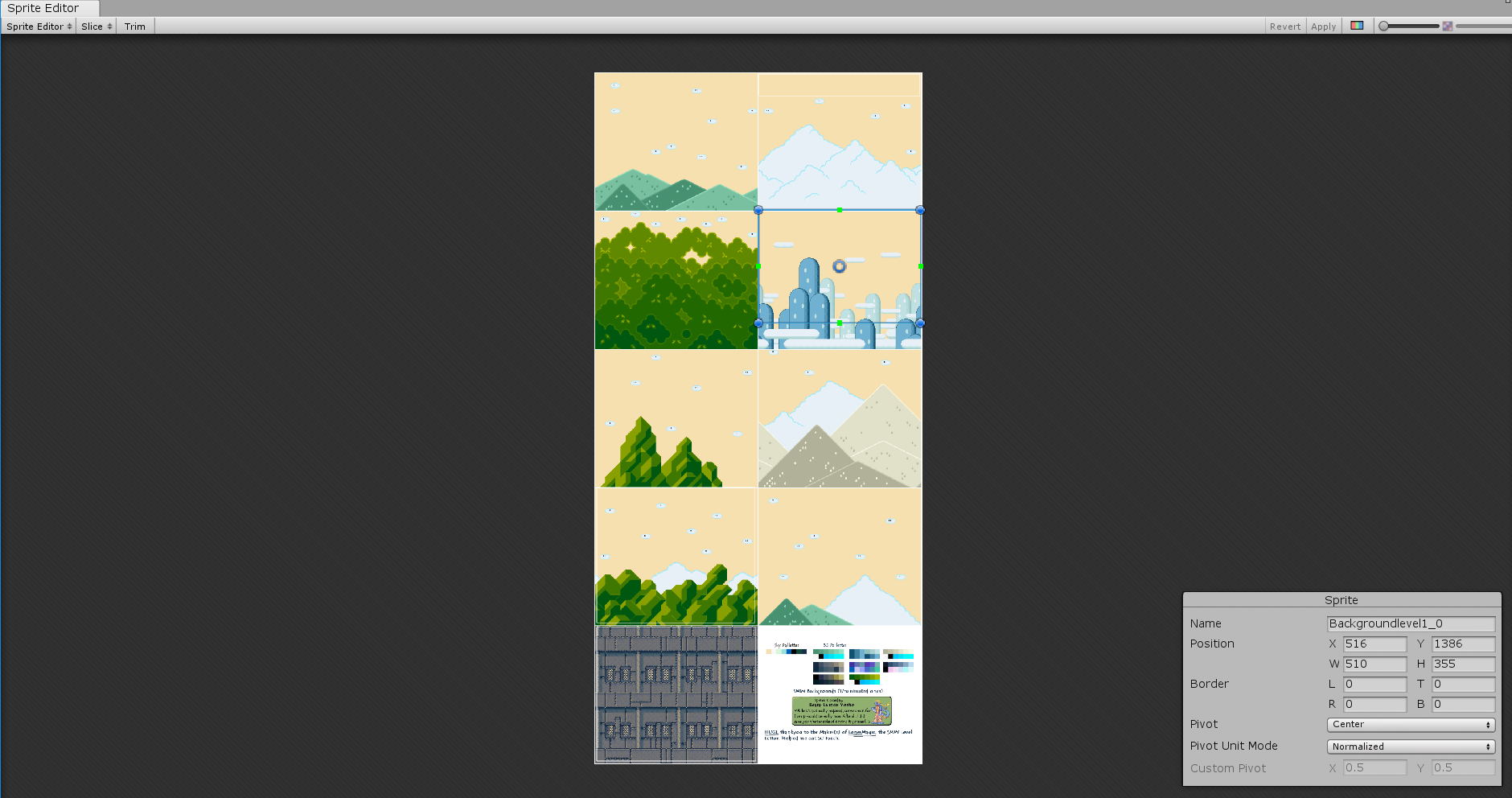
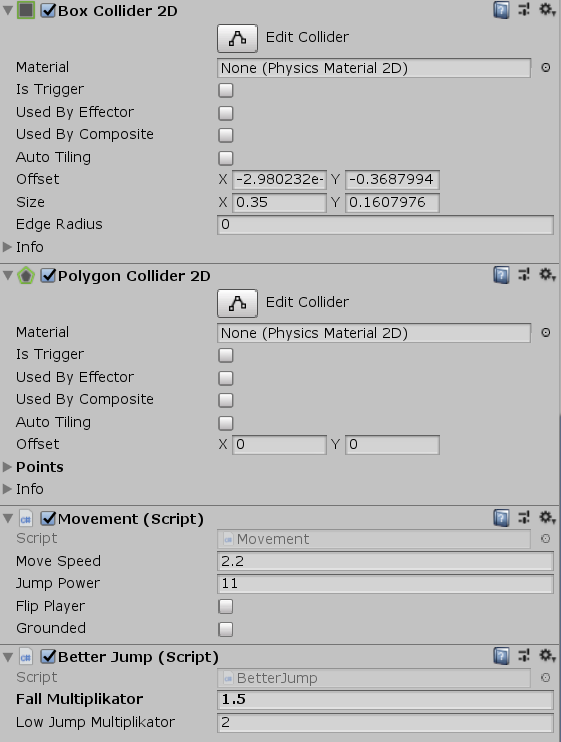


Abbildung 12

Schritt 3: Player (Mario)

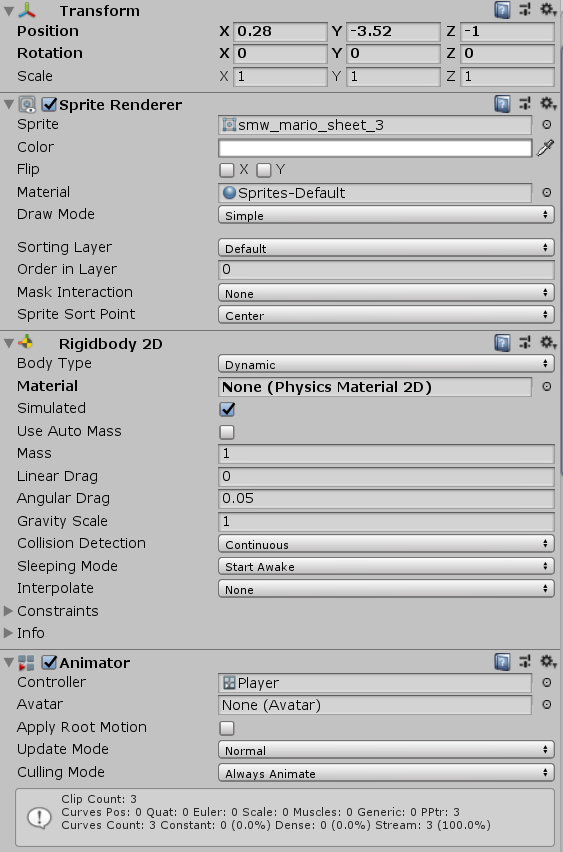
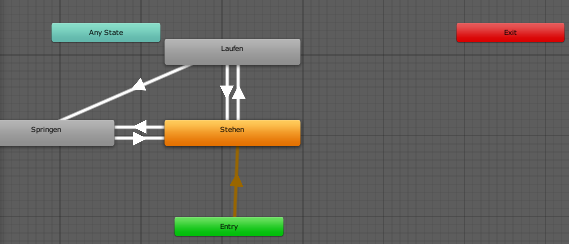
Schritt 4: Animation-Controller (für Player)

Abbildung 13

**Box Collider2D:** Player kriegt einen Box Collider damit er nicht durch die Gegenstände durchlaufen kann (Wichtig!: Das Gegenstand muss auch einen Collider besitzen.)

**Polygon Collider2D:** Hat die selbe Funktion wie Box Collider, allerding kann man diesen genauer anpassen an das Objekt.

**Movemement (Script):** Script für Bewegung und zerstören von Enemies.

**Better Jump (Script):** Proportionale Sprung Funktion.

**Sprite Renderer**: Rendert ein Sprite für 2D-Grafiken.

**Rigidbody 2D:** Unser Spieler bekommt physikalische Eigenschaften (wie zb. Schwerkraft).

**Animator:** Animator dient für das Animation-Controller. Der Conroller dient zur Bewegung, bzw. Image wechsel, für den Player.

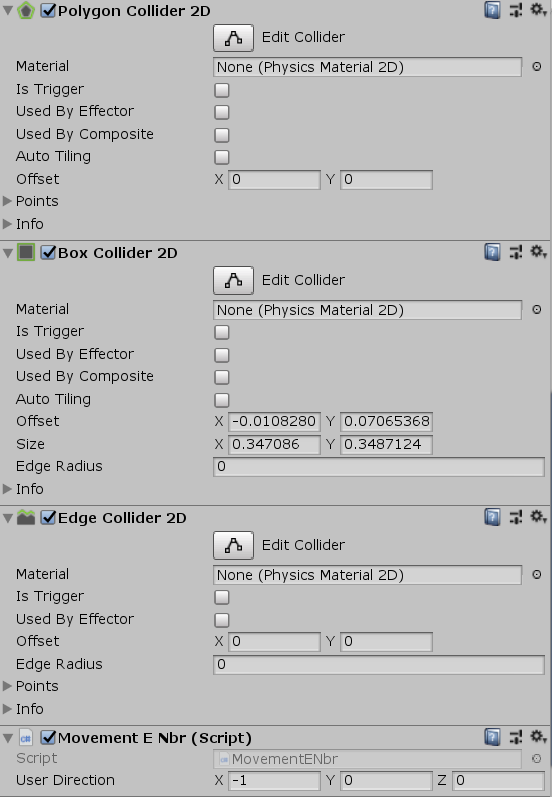
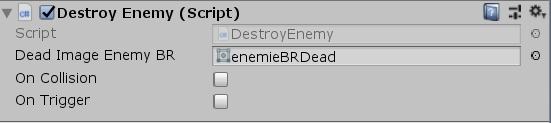
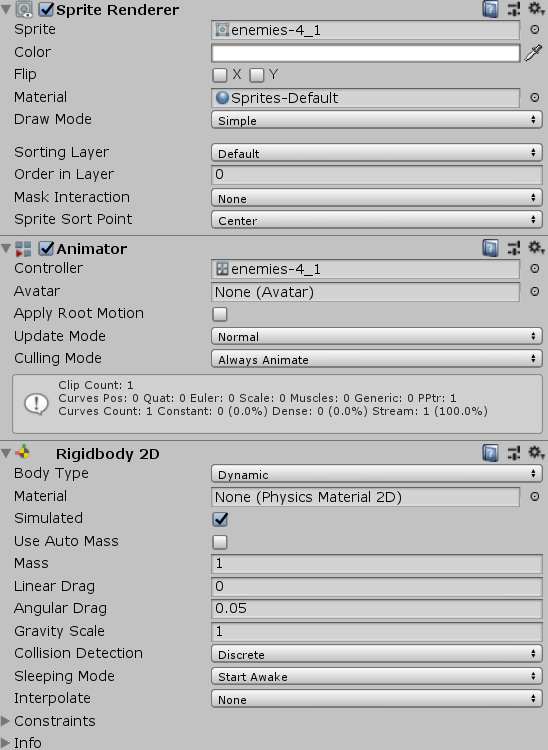
Abbildung 14

Animationen, die zur Verfügung stehen: Stehen, Laufen (Rechts, Links), Springen. Während er steht wird die Steh-Animation durchgeführt, wenn er während stehen springt wird die Spring-Animation durchgeführt. Wenn die Rechts oder Links-Taste betätigt wird die Animation von Laufen durchgeführt. Während laufen kann der Spieler auch Springen. Wenn der Spieler keine Tasten betätigt wird natürlich die Steh-Animation durchgeführt.

Laufen besitzt die Variable „Speed“, dies dient dazu um zu entscheiden ob der Spieler nach rechts oder links läuft. Und Springen besitzt die Variable „Springen“. Weiteres wird im Skript-Bereich beschrieben.

Schritt 5: Enemy-Pilz

Als nächstes werden Enemies erstellt. Indem fall unser Pilz-Enemy.



**Sprite Renderer:** Rendert ein Sprite für 2D-Grafiken.

**Animator:** Animator dient für das Animation-Controller. Der Conroller dient zur Bewegung, bzw. Image wechsel, für den Enemy.

**Rigidbody 2D:** Unser Pilz bekommt physikalische Eigenschaften (wie zb. Schwerkraft).

**Polygon Collider2D:** Hat dieselbe Funktion wie Box Collider, allerding kann man diesen genauer anpassen an das Objekt.

**Box Collider2D:** Player kriegt einen Box Collider damit er nicht durch die Gegenstände durchlaufen kann (Wichtig! Das Gegenstand muss auch einen Collider besitzen.)

**Edge Collider:** Eine unsichtbare Form, mit der physische Kollisionen für ein Objekt verarbeitet werden.

**Movement E Nbr (Script):** Skript für das Bewegen von Pilz. Die X-Achse ist auf – gesetzt, weil unser Pilz sich nur links bewegt.

**Destroy Enemy:** Image Wechsel bei zerstören von Pilz durch Mario.

Abbildung 15

Schritt 6: Enemy-Rocket

Bei Enemy-Rocket ist der Aufbau genau gleich wie Pilz, nur es besitzt kein Destroy Enemy (Skript) und kein Edge Collider. Dafür hat unser Rocket einen kinematischen Körper, damit er in der Luft sich bewegen kann.

Schritt 7: Coins

**Sprite Renderer:** Hat immer noch die gleiche Funktion, wie zuvor auch.

**Animator:** Da ist die Animation für das Drehen von Coins.

**Capsule Collder2D:** Ist ebenfalls ein Collider aber in einer Kreisform.

**Coins(Skript):**Funktion wann Coin verschwindet und somit der Score hochzählt.

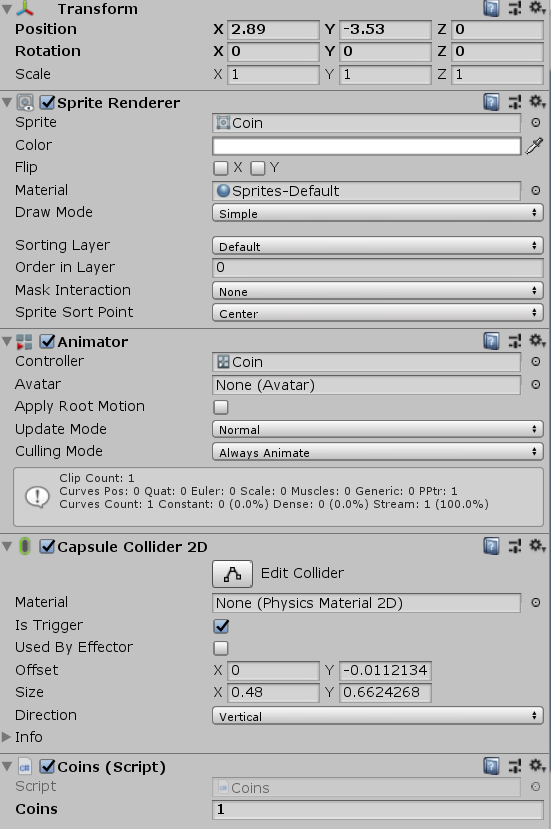


Abbildung 16

Schritt 8: Startmenü/GameOver



Abbildung 17

Unser Startmenü besteht aus einem Hintergrundbild, Textfeld (in Unity: TextMeshPro)

und aus 2 Buttons:

* START: Da wird die erste Szene abgespielt indem fall Level1.
* QUIT: Passiert nichts.

Beim GameOver hat der Spieler eine Wahl, das Spiel zu neustarten oder zu beenden.

Schritt 9: Szenen einbinden

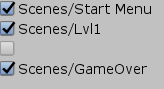
Damit man Szenen switchen kann, muss man die zuerst in Unity Bilden bzw. einbinden:

Abbildung 18

Levelswitches sind vorhanden in: Startmenü, Enemies (GameOver), Pipes (für Untergundlevel), Ziel (1. Von Level 1 zu Level 2; 2. Endziel „EndGame“).

Alles andere wird natürlich geskriptet, was und wann die Szene geswitcht werden soll, beim Drücken eines Buttons oder durch eine Kollision mit GameObject. Dazu erfahren Sie mehr im Skript-Teil.

Schritt 10: Score

Unser Score besteht (Oberflächlich) aus einem Textfeld. Score ist am Anfang gleich 0 gesetzt. Das Textfeld ändert sich bei jedem einsammeln von Coins.

Schritt 11: Tags

Bestimmte GameObjects besitzen Tags, die wir selbst benennen können. Dies vereinfacht das Skripten von unseren Player. Damit man nicht jedes Mal alles neu schreiben muss, wie zb. Destroy bei Coins und Enemies, kann man ganz de Spieler taggen und somit sagen was passiert, wenn ein Objekt mit diesem Objekt (mit der tag) zusammenprallt.

Player-Tag: Player (die Funktion wird im Skript beschrieben)

Boden und Pipes: GroundColl (die Funktion wird im Skript beschrieben)

## Aufgetretene Fehler

Hinweis: Fehler sind beim Testen aufgetreten! Ansonsten wurde bei jeder Änderung/Hinzufügung von Skripts, das Spiel getestet und lief fehlerfrei!

1. Background

Beim Testen unseres Spieles haben wir festgestellt, dass sich die Kamera mit dem Spieler sich bewegt, weil wir die Kamera auf unseren Spieler gesetzt haben, aber das Hintergrund stehen bleibt. Um dieses Problem zu beheben haben wir gemeinsam versucht die Kamera auf das Hintergrund zu fixieren. Da trat aber das Problem auf, dass sich die Kamera unproportional bewegte. Uswwwwwwwww

1. Score

Unser Score-Zähler zählt aus unergründlicher Weise falsch hoch. Wir haben alles Mögliche versucht, indem wir unseren Score-Text anders gestalten haben und oder das Skript geändert haben. Leider konnten wir das nicht fixieren.

1. Enemies zerstören

Da die Pilz-Enemies erst durch das draufspringen von Mario zerstört werden und die Raketen durch einen „Kopfnuss“ oder von drauflanden, hat dies große Probleme zubereitet. Zuerst codierten wir so, dass unser Spieler mit seine Enemies kollidiert, egal von welcher Richtung, und die sich dann auflösen. Dies lösten wir indem man mehrere, unterschiedliche Colliders hinzugefügt haben. Collider für den Kopfbereich von Mario und für den Fußbereich. Da wir ein Collider nur einmal im Skript verwenden dürfen hatten wir keine andere Wahl als unterschiedliche Colliders anzuwenden wie CapsuleCollider, EdgeCollider usw. Dies hatte aber auch den Vorteil das wir unsere Objekte auf Millimeter genau mit einem Collider verweisen könnten. Und konnten somit pro Collider eine Funktion erstellen und die Probleme lösen.

# Projektabschluss

## Projektübergabe

Das Projekt wird als ein funktionsfähiges Spiel weitergeben mit ein, zwei kleine Fehler. Die aber das Spielablauf nicht stark beeinflusst.

## Abnahmeprotokoll

Festhalten was funktioniert und was nicht.

## Projektbewertung

Soll-Ist-Vergleich:

Soll: was wollte man erreichen

Ist: was hat man erreicht

Abweichungen zur Planung

Abweichungen vom ursprünglichen Ziel.

# Quellenverzeichnis

<https://www.spriters-resource.com/snes>

<https://docs.unity3d.com/Manual/index.html>

(Persönliche Erklärung)