



Mario vs. Goomba – Designdokument

Alida Kohler, MKB 5, 263819

Sommersemester 2022

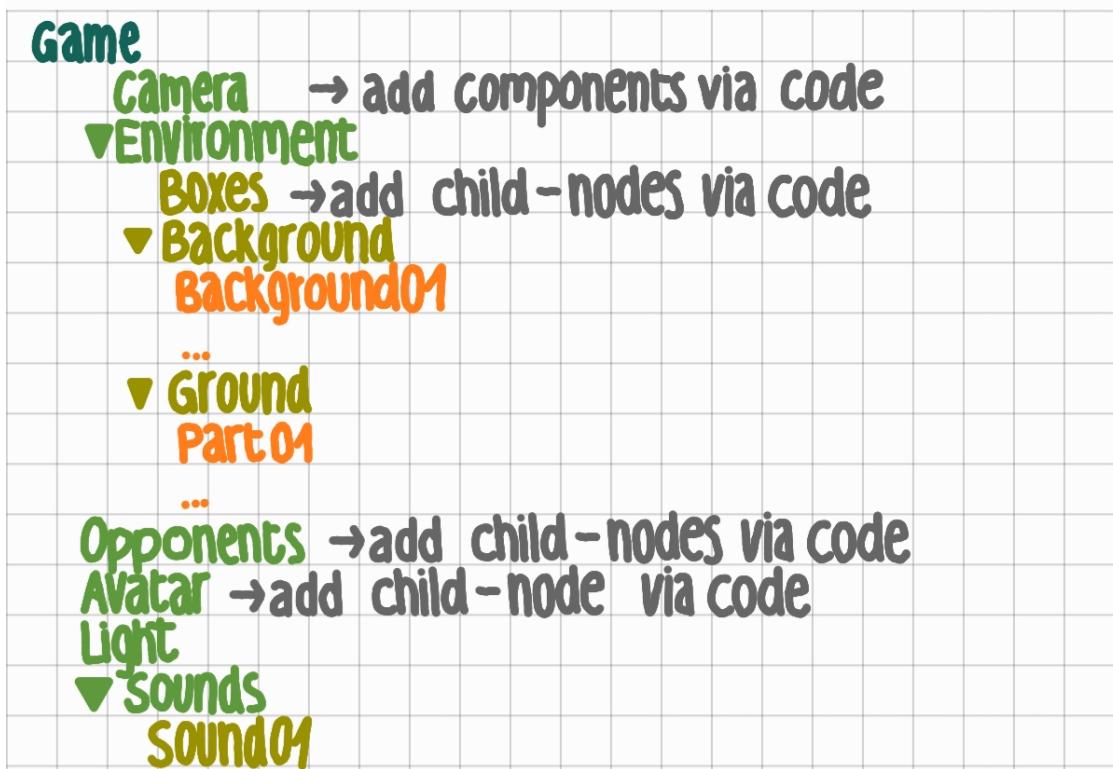
PRIMA

Links zu den Punkten in den jeweiligen Dateien sind grün unterstrichen.

1. Wo ist 0 und was ist 1?

0 ist in der unteren linken Ecke des Spielfelds. Das ermöglicht ein einfaches Positionieren von den verschiedenen Komponenten, da die y-Position auf 0 gesetzt werden, bzw. davon ausgehend hochgezählt werden kann. Auch die Überprüfung, ob Mario oder die Goombas heruntergefallen sind ist damit einfacher, weil einfach nach einem negativen y-Wert gesucht werden kann. Da das Spiel von links nach rechts gespielt wird, macht es Sinn, links zu starten. 1 bzw. eine Einheit in meinem Spiel ist eine Boden-Textur oder eine Item-Box, die jeweils 1m breit und hoch ist.

2. Graph-Setup und Editor



farbig = Editor, grau = Code

Die grundlegende Struktur des Spiels wird im Editor erstellt. Fixe Parts, die sich nicht verändern werden, sind ebenfalls im Editor festgelegt, dazu zählen der Hintergrund, die Boden-Parts, das Licht und die Sounds. Im Code bekommt die Kamera ihre Components, Mario und die Goombas werden gespawnt und die Boxen erstellt. Die Anzahl der Goombas und die der Boxen können, über die Config-Datei festgelegt werden, deshalb macht es wenig Sinn diese im Editor zu erstellen.

Die fixen Komponenten wie der Boden oder Hintergrund sind im Editor einfacher erstellt, hier kann man dann auch direkt sehen, was man macht und es gibt eine Basis, um den Code zu testen.

3. ScriptComponents

Die Positionsfindung für die Boxen und die Goombas wird über den ScriptComponent [SetPosition](#) geregelt, da hier das gleiche Prinzip verwendet werden kann. Durch den Script-Component habe ich es mir zwar gespart, zwei Methoden in die jeweiligen Klassen zu schreiben, dafür musste ich die gesamte Datei mit dem ScriptComponent aufsetzen. In meinem Fall war der Component also weder besonders nützlich, noch besonders hinderlich.

4. Extend

Die Klassen [Coin](#), [Item](#), [Goomba](#) und [Mario](#) sind Subklassen von f.Node.

[SetPosition](#) ist eine Subklasse von f.ComponentScript.

Das extenden der Klassen war in meinem Fall sehr nützlich, dadurch konnte ich die Methoden und Eigenschaften von Node verwenden und Ergänzen was ich zusätzlich noch benötigt habe. Auch das extenden von ComponenScript hat mir das Unterbringen der Script-Komponente deutlich vereinfacht.

5. Sound

Mario macht einen Sound, wenn er [springt](#) und die Münzen machen ein Geräusch, wenn sie [gespawnt](#) werden. Je nach dem, ob der Spieler gewinnt oder verliert, kommt entweder der klassische [Mario-Death-Sound](#) oder eine [Gewinn-Melodie](#).

Alle Sounds werden direkt vom Graph aus abgespielt, da es in Supermario keine Geräusche gibt, die aus einer bestimmten Richtung kommen.

6. VUI

Über das [VUI](#) werden die verbleibende Zeit und die aktuelle Punktzahl dargestellt. Die Zeit wird als Countdown dargestellt und das Spiel ist beendet, wenn 0 erreicht wird. Jedes Mal, wenn ein Goomba von den Plattformen fällt oder eine Münze eingesammelt wird, wird die Punktzahl geupdated. Damit der Spieler weiß, was die Zahlen bedeuten, stehen neben den jeweiligen Werten noch „Time“ bzw. „Points“

7. Event-System

Der Graph hört auf das „[gameEnd](#)“-Event, das das Spiel beendet und eine entsprechende Nachricht in einem Div darstellt. Mit der Detail-Eigenschaft des CustomEvents werden noch Informationen mitgeliefert, ob das Spiel gewonnen oder verloren wurde.

Getriggert wird das Event von der Mario-Klasse, wenn Marios Position einen negativen y-Wert bekommt, vom Timer, wenn dieser 0 erreicht und von der Enemy-StateMachine, wenn der letzte Goomba stirbt.

In meinem Fall fand ich das Custom-Event ziemlich hilfreich, da ich das Event für jeden Game-Over Fall verwenden und mit der Detail-Eigenschaft des CustomEvents noch zusätzliche Informationen mitschicken konnte.

8. External Data

In der Datei [config.json](#) können die Anzahl der Item-Boxen und die der Gegner sowie die zur Verfügung stehende Zeit festgelegt werden. Mit diesen Parametern kann man die Schwierigkeit des Spiels einstellen und auch zum Testen waren die Variablen sehr angenehm.

A. Light

In Supermario gibt es kein Licht und Schatten, alles ist gleichmäßig beleuchtet. Daher habe ich mich bei der Beleuchtung für ein einfaches [Ambient-Light](#) entschieden.

B. Physics

[Mario](#), die [Goombas](#), die [Item-Boxen](#) und der Boden verwenden RigidBodys. Die Goombas und die Item-Boxen registrieren Kollisionen mit Mario und triggern entsprechendes Verhalten ([Coins spawnen](#), [Änderung des State-Machine States](#)).

Mario springt durch die [applyLinearImpulse](#) Methode des RigidBody.

C. Net

/

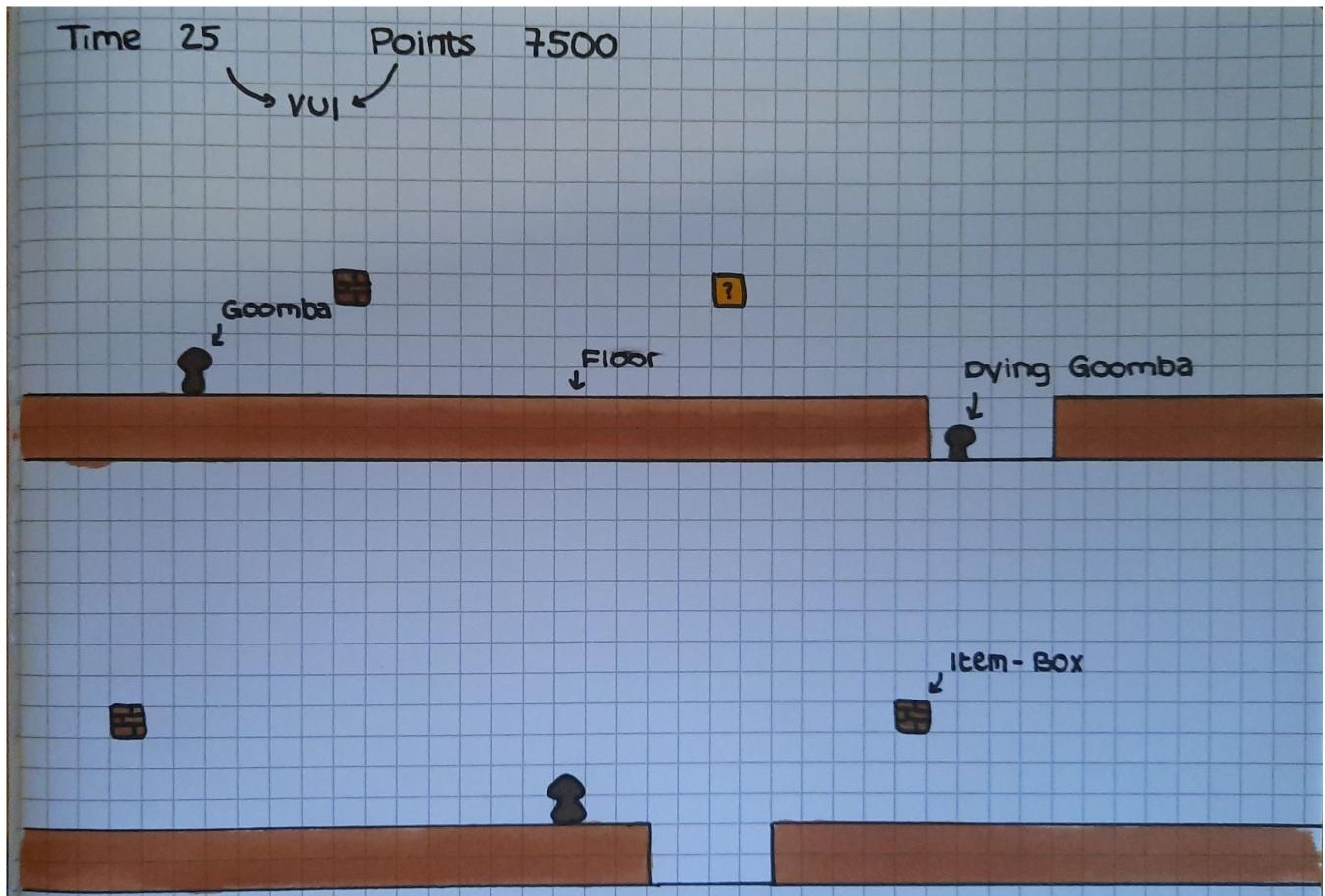
D. StateMachines

Das Verhalten der Goombas wird durch eine [ComponentStateMachine](#) gesteuert, mit den States Walk, Fight und Die.

E. Animation

Mario und die Goombas haben eine [Sprite-Animation](#). Die Münzen werden durch eine normale [Animation](#) bewegt.

Skizze



Sprites und ImageTextures

Alle Sprites und Texturen wurden von mir in Aseprite mit Vorlage von den originalen Supermario-Spielen von Hand gezeichnet.



Bestandteile der Dateien

Super Mario ~ Dateien

Main

- start / setup
- update
- get External Data
- Timer
- end Game
- ground Parts (pos.)
- create Components
 - ↳ Mario, Items, Goombas
- create random numbers

Camera

- setup
- move camera

Mario

- setup
- walk
- jump
- update
- check position

Goomba

- setup
- update
- flip Sprite
- direction
- StateMachine

Item

- setup
- manage collision
- spawn coin
- coin

Coin

- spawn
- animate Coin
- lifespan