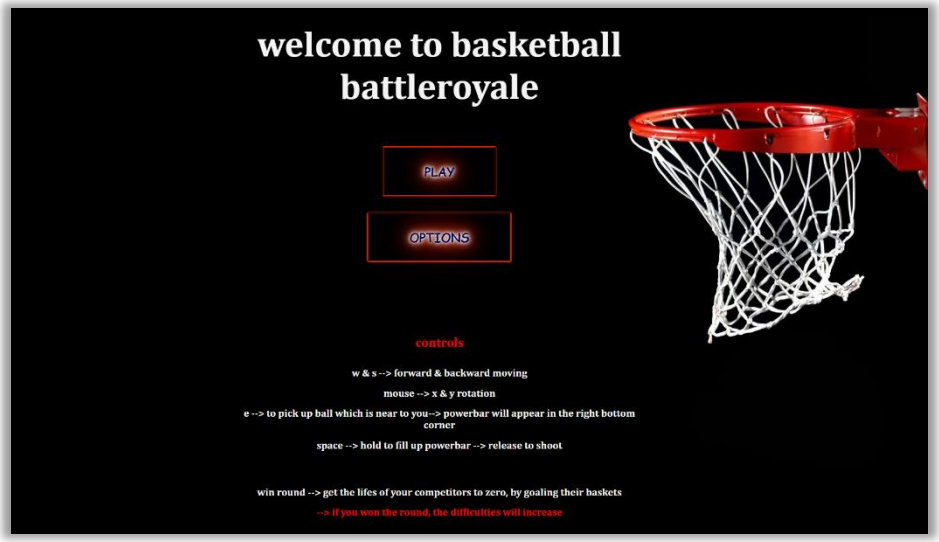
Designdokument

Prototyping interactive media and games

Basketball Battle Royale

Prototype



Schmidberger Valentin

Mühlwies 24

88267 Vogt

vali.vogt@live.de

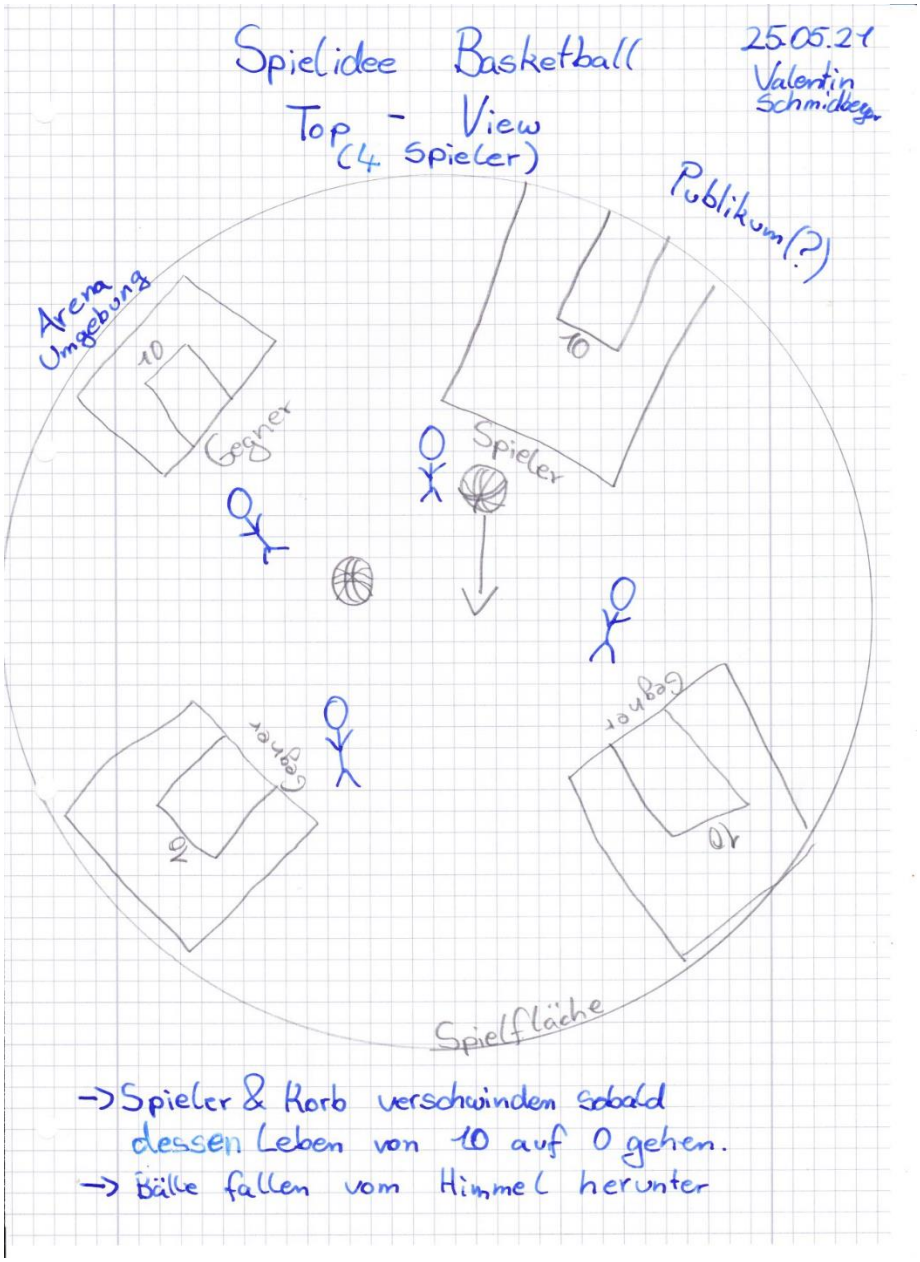
Mat.-Nr. 263249

Betreuer: Jirka Dell`Oro

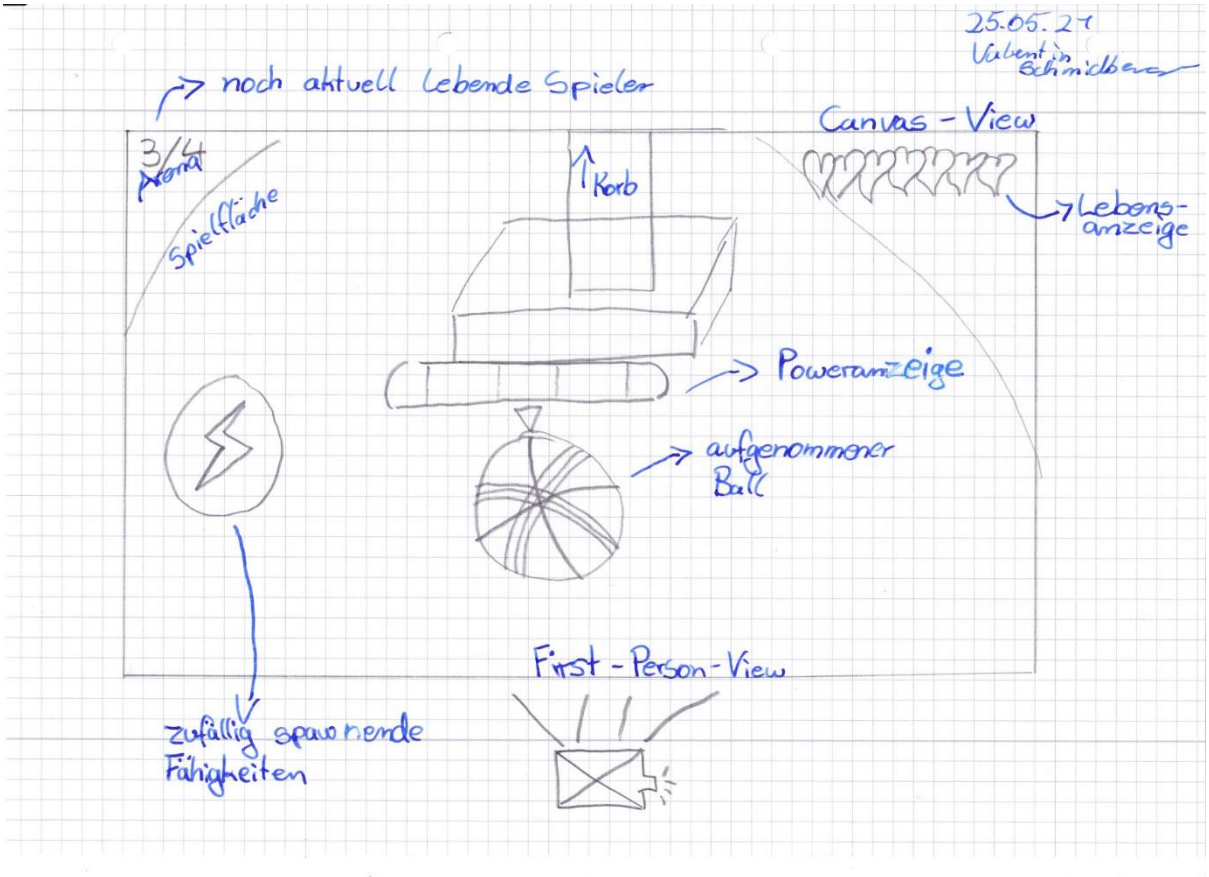
Eingereicht am: 23. Juli 2021

Idee

Nachdem uns Marko die Physik Engine ausführlich nähergebracht hatte, erschien es mir direkt sehr attraktiv ein Prototyp auf Grundlage dieser Engine aufzubauen. Da ich im Praxissemester zuvor schon ein Basket Ball Spiel in der Unity Engine programmieren durfte, tendierte ich schnell auch in Fudge zu so einem ähnlichen Spiel. Es sollte ein 3D FPS Game werden auf der Grundlage der Oimo Physics. Allerdings nicht ein herkömmliches Basket Ball Spiel wie man es kennt, sondern in der Battle Royale Variante: Es gibt insgesamt 4 Spieler auf einer kreisrunden Fläche, die sich jeweils gegenüberstehen. Jeder Spieler besitzt einen eigenen Korb mit einer gewissen Anzahl von Leben. Die Basketbälle spawnen an einer zufälligen Position im Spielfeld, in der Anzahl der noch vorhandenen Spieler. Zur Erklärung: Bei 4 Spielern sind immer 4 Bälle auf dem Feld vorhanden, bei 2 Übriggebliebenen dementsprechend 2 Bälle.



Der Spieler selbst findet sich in einem Avatar, in der first person view, wieder. Kann sich über die „W“, “S“ Tasten vor-/ zurückbewegen und sich mit der Maus um sich selbst drehen. Wenn er zu einem Basketball läuft, kann er diesen mit der Taste „E“ aufnehmen und sieht den Ball vor sich. Über das

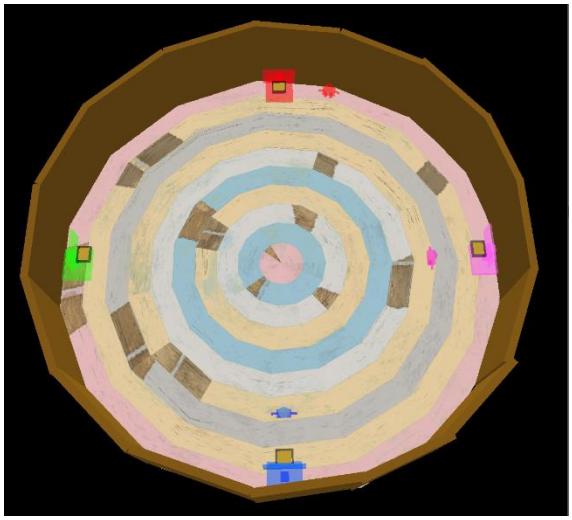


Halten der „Leertaste“ kann er dann eine Powerbar auffüllen, sobald die Taste losgelassen wird, kommt es zum entsprechend starken Schuss des Balls. Für den Ausgang des Spiels gibt es 2 Szenarien: der Spieler eliminiert die NPCs und gewinnt die Runde, oder einer der NPCs setzt den Spieler auf Null und verliert die Runde somit. Falls der Spieler gewinnen sollte, kann er mit einem höheren Schwierigkeitsgrad erneut spielen. Zufällig spawnende Abilities sollten dem Spiel weiteren Pepp geben (konnte ich zeitlich leider nicht mehr umsetzen).

Umsetzung Leveldesign

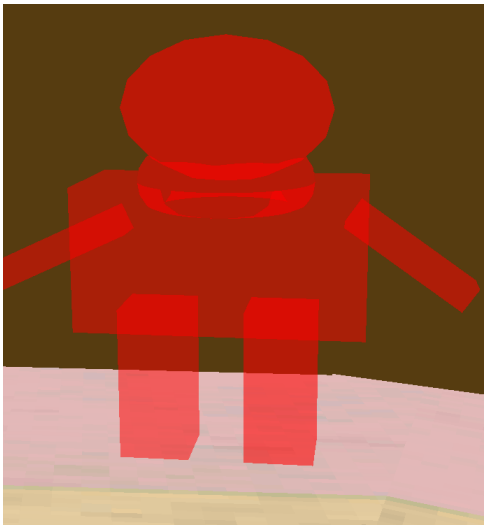
Bevor ich mich ans Werk im Fudge Editor machte, wollte ich mir einen groben Überblick über die sowohl statischen als auch nicht-statischen Objekte verschaffen. Deswegen erstellte ich erst ein Mal eine Objekt Hierarchie:

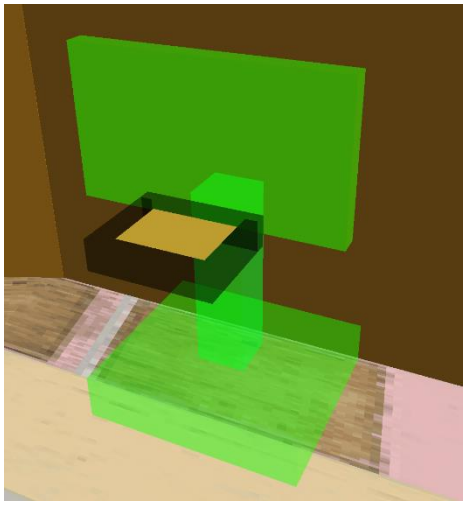
Dann begann ich anhand der oben gezeigten Strukturen, im neuen Fudge Editor die Grundbausteine für das Spiel zu legen. Zuerst begann ich mit dem Herzstück: dem Basketball. Über den Editor inzwischen erfrischend einfach zu bedienen: Sphere im internalen Fenster hinzugefügt, die erstbeste kostenlose Holztextur über das Internet heruntergeladen und auf das Mesh angewendet. Der Basketball wurde später als eigene Graph Instanz abgespeichert, um während dem Spiel eine Referenz dieser Instanz zu bekommen und diese dann immer wieder neu zu instanziieren. Diese Variante entpuppte sich als die Beste, da ich weiterhin geplant hatte Bälle zu zerstören, sobald sie als Treffer gewertet wurden, um dann an einer anderen zufälligen Stelle einen neuen Ball „spawnen“ zu lassen.



Der nächste Schritt war es die Bodenfläche aufzubauen. Auch hier nach einem simplen Prinzip: im Editor den Zylinder auf eine Node gezogen, eine passende Textur gewählt und fertig. Etwas mühsamer waren die jeweiligen Barrikaden an jedem Eck des Zylinders. Diese mussten über ihre Transform Komponente jeweils verschoben und rotiert werden, um einen fließenden Übergang ohne Kanten zu erreichen.

Als auch das geschafft war, baute ich das Mesh für die NPCs. Dabei hielt ich es einfach, um nicht an diesem Schritt zu viel Zeit zu verlieren. Ganz rudimentär bekamen sie Kopf, Rumpf, Arme und Beine. Nur der Rumpf bekam später über das Enemy Skript den Rigidbody, der Rumpf war also das oberste Objekt des NPCs. Der Rigidbody wurde auf Dynamic gesetzt, da ich die NPCs von Ball zu Ball bewegen lassen musste.



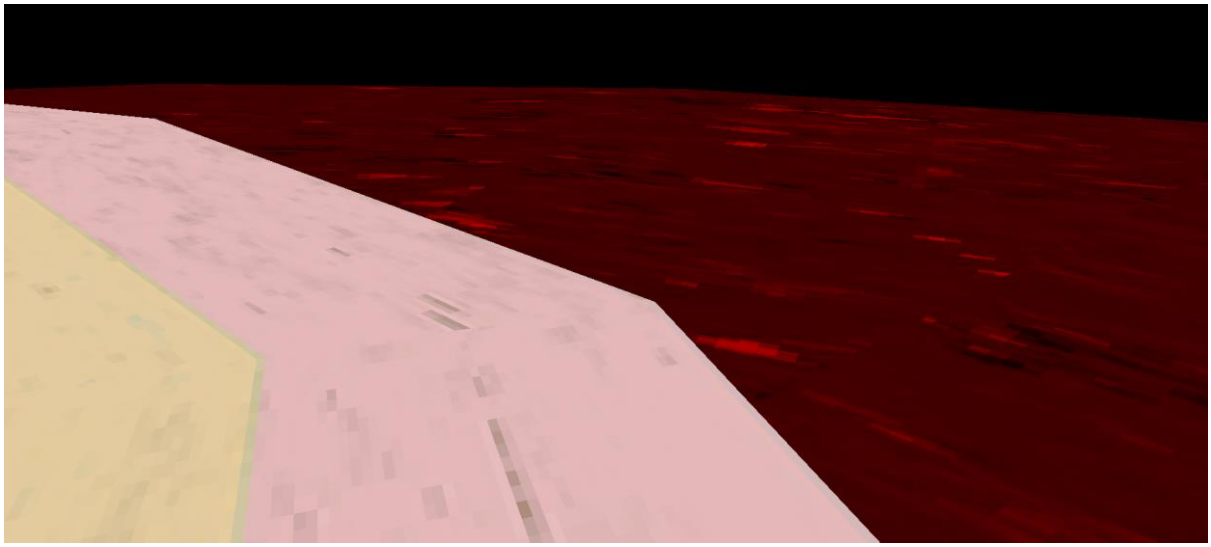


Zuletzt baute ich den Basketballkorb. Auch hier wieder nach demselben Prinzip: erstmal nicht zu viel Zeit verlieren. Was sich im Nachhinein als gute Entscheidung erwiesen hat, da ich trotz einfach gebauten Meshes zum Ende der Prüfungszeit unter Zeitdruck geriet. Der Korb bestand ausschließlich aus verschieden skalierten Würfeln. Innerhalb des Korbs platzierte ich ein statischen Rigidbody der als Trigger fungierte, um später getroffene Bälle registrieren zu können.

Umsetzung Code

Kurzfassung:

Bei der Umsetzung von Code hilft es mir oft, einfach erstmal drauf loszulegen. Denn ich verstehe unter Coding eines größeren Projekts, einen längeren Prozess, der durch Optimierung und neuer Erkenntnisse, weiter angepasst und verbessert wird. Ich begann zuerst alle Meshes, die ein Rigidbody brauchen über den Code damit auszustatten. Das betraf sowohl statische als auch dynamische Objekte. Unter anderem befanden sich dort auch 2 Trigger, einmal jeweils an den Basketballkörben. Und unterhalb der eigentlichen Spielfläche, eine große Fläche mit Trigger, die verloren gegangene Bälle registriert und zerstört. Im Bild entspricht es der roten Fläche.



Danach kam das Skript für den eigentlichen Spieler, welches alle Inputs von Tastatur und Maus, sowie sonstige Spielfunktionalität steuert, wie zum Beispiel die Intensität des Ballschusses. Dann das Skript für die NPCs, welches etwas komplexer ist, aber im Endeffekt so funktioniert, dass es den Gegner zu noch freien Bällen hinbewegt und dann den Ball aufnimmt, und randomisiert ein gegnerischer Korb ins Zielt nimmt und darauf schießt. Um unterschiedlichen Schussintensitäten zu entgehen, lass ich die NPCs einfach immer in die Mitte des Spielfelds laufen, sobald sie im Besitz eines Balles sind, denn dann bleibt der Schuss immer der Gleiche, egal welcher Korb gewählt wurde. Dann baute ich das von mir so benannte BasketBallSpawner Skript, welches alle 10 Frames überprüft, wie viele Spieler noch am Leben sind und wie viele Bälle aktuell im Bälle Array stecken, sollte es weniger Bälle als Spieler geben, „spawnt“ das Skript eine neue Ball Graph Instanz. Somit ist gesichert, dass jeder Spieler immer auf ein Ball Zugriff hat. Danach bekam jeder Trigger im Basketballkorb ein Component Skript das gleichzeitig mehrere Dinge überprüft: den aktuellen Score des Korbes, welcher dekrementiert wird, sobald ein Treffer registriert wurde oder wenn ein 2. Ball auf den Korb trifft, obwohl der 1. Ball noch nicht zu Ende verarbeitet wurde, wirft er den 2. Ball wieder raus. In diesem Skript wird auch gecheckt, ob ein Score auf Null geht, wenn es der des Spielers ist, ist das Spiel verloren und der Nutzer wird wieder ins

Hauptmenü geführt. Falls einer der NPCs gegen Null geht wird dieser mitsamt seinem Korb vom Spielfeld entfernt. Bleibt der Spieler als letzter der Runde bestehen, hat er die Runde gewonnen und wird aufgefordert das Spiel mit einer höheren Schwierigkeit versuchen zu gewinnen. Überprüft wird das Ganze über das Abgleichen des aktuellen Strings im localStorage.

Klassendiagramm