Tätigkeitsbericht von Sarah Johannsen

Kurs: Video Game Development
Bearbeitetes Projekt: Auto Chess

Link zum OpenGL-Build: https://connect.unity.com/mg/other/untitled-7209
Link zu einem Windows-Build auf GitHub: https://github.com/SarahJoh/Battle-Sim

In dem GitHub-Repository findet man außerdem noch einmal diesen Tätigkeitsbericht und alle C# Skripte (falls die für die Benotung wichtig sein sollten). In diesem Tätigkeitsbericht orientiere ich mich an der Vorlage, die von Opencampus zur Verfügung gestellt wird.

1. Das Projekt, das ich im Rahmen dieses Kurses bearbeitet habe ist eine Abwandlung von Auto Chess. Beim Auto-Chess treten zwei Spieler gegeneinander an indem sie in der Kaufphase Spielfiguren kaufen und auf einem Spielfeld platzieren. Diese greifen sich dann in der Kampfphase gegenseitig an und derjenige der als letztes noch lebende Spielfiguren auf dem Feld hat gewinnt die Runde. In meinem Spiel tritt der Spieler gegen den Computer an. Ziel ist es fünf Runden zu gewinnen, wobei jede Runde eine Spielfigur mehr auf das Feld gestellt werden kann. In der letzten Runde treten also 5 Spielfiguren vom Spieler gegen die des Computers an. Wird eine Runde verloren muss man wieder bei der Ersten anfangen.

2. Ich habe dieses Projekt alleine bearbeitet, kann also keinen Projektpartner vorstellen

3. Es gab mehrere Herausforderungen in diesem Projekt, die Größte war aber definitiv meine fehlenden Vorkenntnisse in Unity und C#. Doch Dank sehr vieler Tutorials und den C# Microsoft Docs finde ich mich jetzt in Beidem gut zurecht. Eine weitere Sache für die ich sehr lange gebraucht habe waren die Shader um den Artstyle umzusetzen, auf den ich mich festgelegt hatte. Ich wollte einen Cel-Shader in Unity entwickeln, der am Ende auf alle Objekte angewandt werden kann, sodass die Objekte zwar farblich alle nur Volltonfarben haben, aber trotzdem interessant aussehen. Dazu wollte ich durch einen weiteren Shader schwarze Outlines erzeugen, wie man sie beispielsweise aus Borderlands kennt. Diese sollten aber nicht als Textur importiert, sondern automatisch generiert werden. Bei der Implementierung die ich verwendet habe wird die Breite dieser Outlines dadurch bestimmt wie weit ein bestimmter Teil eines Objekts von einem anderen dahinterliegenden Teil im Viewport entfernt ist. Dies kann man beispielsweise gut an den Kanten der Insel erkennen, da hinter ihr ja nur der Hintergrund als Volltonfarbe liegt, wodurch sie eine Outline mit maximaler Strichstärke hat. Um den Shader bestmöglich zu sehen empfehle ich das Spiel nur im Vollbildmodus zu spielen.

4. Zu Beginn des Projektes habe ich mir zuerst Gedanken darüber gemacht wie ich die Grundidee von Auto Chess umsetzen möchte ohne das Spiel zu aufwändig werden zu lassen. Im Endeffekt habe ich mich auf die Idee festgelegt die ich schon in 1. beschrieben habe. Danach habe ich die 3D-Assets für das Spielfeld und die fünf Charaktere in Blender erstellt und in Unity importiert. Es erschien mir sinnvoller das zu machen bevor ich anfange den Code zu schreiben, damit ich später den geschriebenen Code direkt mit meinen eigenen Modellen testen kann. Als nächstes habe ich angefangen das Skript BoardManager zu schreiben, das später die komplette Logik des Spielfeldes enthalten wird. Hierzu habe ich zuerst eine Funktion geschrieben, die die derzeitigen x und y Positionen der Maus zwei Variablen zuordnet um schauen zu können ob sie sich auf dem Spielfeld befindet.

Danach habe ich eine weitere Klasse namens Character erstellt die später auf jeden einzelnen Charakter, der sich im Spiel befindet, angewendet wird und auf die durch eine Instanz vom BoardManager aus zugegriffen werden kann. Ich habe darauf hin ein 8x8 Array erstellt, das Elemente vom Typ Character speichern kann und das komplette Spielfeld darstellt. Durch die Funktion SpawnCharacter ist es möglich einen Charakter anhand seiner ID an einer bestimmten Position des Spielfeldes zu platzieren, indem eine Instanz des gewählten Charakter-Prefabs erstellt wird. Dieser Charakter kann dann (während die Kampfphase nicht aktiv ist) durch Mausklicks frei auf der Spielerseite des Spielfeldes bewegt werden. Jedes Mal wenn ein Charakter bewegt wird, wird durch das Charakter-Array geprüft ob die angeklickte Position zulässig ist und das Array geupdated. Als das alles funktioniert hat habe ich zuerst den Cel-Shader und danach den Outline-Shader implementiert. Da nur ein Shader auf ein Material angewendet werden kann habe ich mit Hilfe der Blit Klasse definiert, dass die Outlines auf alle Objekte angewendet werden, die von meiner Render-Pipeline gerendert werden (also auf alle Objekte in meiner Szene). Als nächstes habe ich einen Timer implementiert, der in der Kaufphase aktiv ist und die Kampfphase startet wenn er zehn Sekunden gelaufen hat. Der nächste Teil des Codes hat besonders viel Zeit beansprucht, da ich eine Funktion entwickeln musste durch die jeder Charakter den Gegner finden kann der am nähesten bei ihm liegt. Im Endeffekt habe ich das Problem so gelöst, dass Charaktere einen „Player“ oder „Enemy“ Tag zugewiesen bekommen, basierend auf dem y Wert an dem sie gespawnt sind. Es wird dann eine Liste erstellt die alle Charaktere mit dem jeweilig anderen Tag enthält und für jedes Objekt der Liste die Distanz berechnet. Die erste betrachtete Distanz wird als kürzester Weg gespeichert und überschrieben falls eine noch kürzere Distanz gefunden wird. Nachdem alle Objekte betrachtet wurden wird die Position des Objektes, das den kürzesten Weg besitzt, ausgegeben. Der Charakter bewegt sich darauf hin zu dieser Position und rotiert gleichzeitig in die Richtung des Objektes. Wird eine Kollision erkannt zieht der Charakter dem Objekt mit dem er kollidiert ist pro Frame einen Teil seines Lebens ab. Wie groß dieser Teil ist, wird von der Funktion TypeCorrelations bestimmt. Nun musste ich noch Vorschaubilder für die einzelnen Charaktere erstellen und diese den vier Buttons zuweisen, mit denen man in der Kaufphase Charaktere kaufen kann. Gleichzeitig habe ich auch eine Übersicht über die Spielregeln erstellt, die sichtbar wird wenn man auf das Fragezeichen unten links klickt. Schlussendlich habe ich noch eine Funktion erstellt die jede Runde einen weiteren Gegner auf dem Spielfeld spawnt, bis die fünfte Runde gewonnen oder eine Runde verloren wurde. Das Spiel habe ich dann mit Hilfe von OpenGL online veröffentlicht.

5. In dem Projekt lief nichts wirklich schlecht doch es war sehr zeitintensiv, da ich mir meine Ziele wohl etwas zu hoch gesetzt hatte. Aus diesem Grund hatte ich zum Ende hin nicht mehr die Zeit Animationen hinzuzufügen und konnte auch einen bestimmten Bug nicht mehr fixen: Im Moment gibt es leider noch das Problem, dass einer von zwei Charakteren wenn sie hintereinander stehen und sich zum selben Gegner bewegen nie mit dem Gegner kollidiert, da der Eine dem Anderen im Weg steht. Dies zu korrigieren wäre noch sehr zeitaufwändig gewesen, deshalb ist der Fehler derzeit noch im Spiel. Hätte ich noch einmal die Chance mir ein Projekt auszusuchen hätte ich definitiv eine simplere Projektidee gewählt. Außerdem hätte ich mir vorher einen festen Zeitplan erstellen sollen. Hätte ich dies gemacht wäre es mir bestimmt auch möglich gewesen die beiden eben genannten Dinge noch vor der Deadline zu implementieren, doch auch so ist es mir gelungen einen funktionsfähigen Prototyp zu entwickeln und meine gesetzten Ziele zu erreichen.

6. Arbeitsaufwand: Präsenztermine (Auftaktveranstaltung ausgeschlossen): 7 Termine = **14 Stunden**
Erstellung von 2D und 3D Assets: **5 Stunden**
Implementierung von Shadern: **6 Stunden**
Schreiben der Skripte und Debugging: **30 Stunden**
Insgesamt: 55 Stunden