Designdokument – Kreuzungs-Chaos

PRIMA WiSe 20/21 – Jirka Dell'Oro-Friedl | Vorgelegt von Alexander Reiprich (263006)

Grundkonzept

In dem Spiel Kreuzungs-Chaos übernimmt man die Kontrolle über die meistbefahrene Straße in Prima-City. Während Autos von allen Richtungen kommen und in alle Richtungen wollen, ist man damit beschäftigt, die Ampeln so zu steuern, dass keine Unfälle passieren. Wie man es aus dem echten Leben kennt, sind rote Ampeln für alle Autofahrer jedoch nichts Erfreuliches. Stehen sie zu lange auf einer Stelle, werden sie wütend und überqueren die Kreuzung ohne Rücksicht auf Verluste. Das Spiel endet, wenn eine Kollision stattfindet.

Funktionsweise – Code

Das Spiel wird anfangs durch eine Load-Funktion geladen und danach mit einer Loop-Funktion geleitet.

In der Load-Funktion werden die benötigten Daten geladen, Variablen und Texturen gesetzt, sowie Audio und den Viewport zu definieren.

Die Hauptfunktion der Loop-Funktion ist das Verwalten der Autos, das Prüfen auf Kollisionen, sowie das HUD zu updaten.

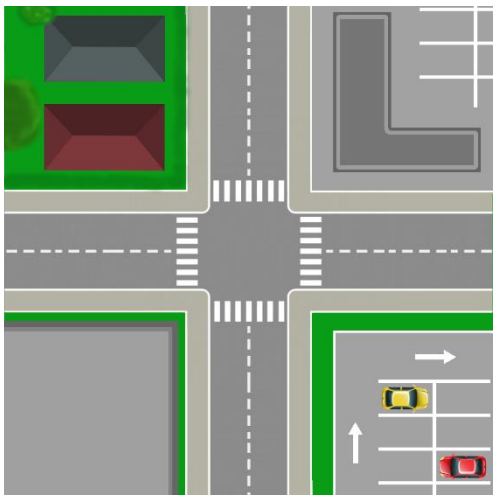
Der Ablauf des Spiels ist hauptsächlich durch Timer und Zufallsgenerierung bestimmt. Die Autos werden periodisch mit leichten Variationen generiert, bei dem Umschalten der Lichter gibt es einen „Cooldown“, die Autos haben einen internen Timer für die Wartezeit an einer Ampel. Ebenfalls wird die Start- und Endlocation der Autos, die Farbe, und zum Teil die Zeit zwischen der Generierung der Autos durch zufällige Zahlengeneration bestimmt.

Ideenfindung und Entwicklung des Projekts

Ursprünglich stand die Idee im Raum, ein Puzzlespiel zu programmieren, welches an das Brettspiel „Rush Hour“ angelehnt ist.

Ich habe mich jedoch dagegen entschieden, da es mir nicht originell genug war, der Aufwand für die Erstellung einzelner Level mir zu hoch erschien, und das Programmieren mir generell für ein Rätselspiel eher langweilig vorkam.

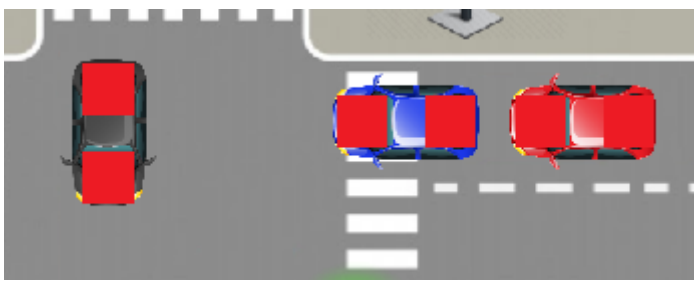
Ich wollte jedoch bei dem Konzept eines Verkehrsmanagement-Spiels bleiben, und habe mich daher für eine Art „Endless Runner“/Arcade-Genre entschieden.



Sobald das Konzept feststand, habe ich mich um das Design gekümmert. Die Kreuzung ist Eigenkreation, die Sprites der Autos sind farbliche Variationen eines Cartoon-Autos aus dem Internet. Weitere Quellen der benutzten Images sind in der Readme verlinkt.

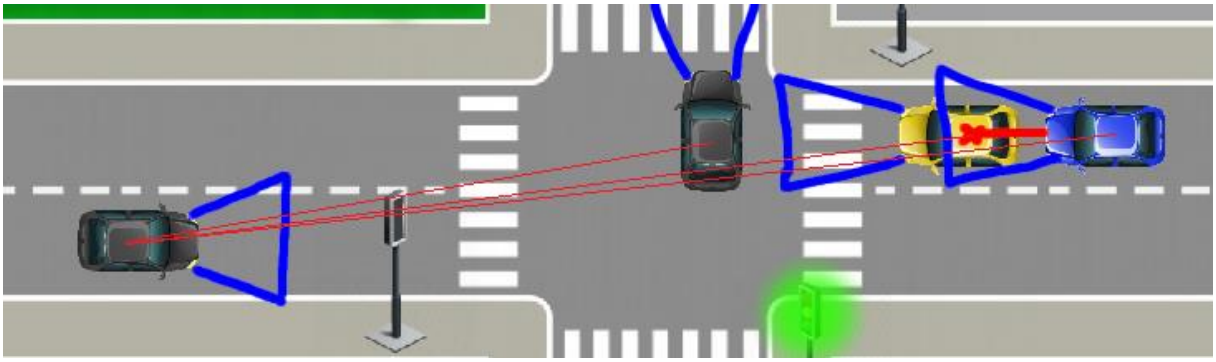
Als es schließlich an das Programmieren ging, stand der Hauptfokus zunächst auf dem Erstellen und dem Verhalten der Autos. Ohne Ampeln und sonstige Hindernisse funktionierte dort alles ohne Probleme. Diese kamen jedoch mit dem Hinzufügen des Abbiegens dazu. Linksabbieger richtig einzubinden stellte sich als schwer heraus, da der Spieler nicht genug Kontrolle hat, Linksabbieger separat zu anderen Autos auf einer Spur zu behandeln. Dadurch waren Unfälle mit dem Gegenverkehr unvermeidbar, auf welche der Spieler keinen Einfluss haben konnte. Dies führte dazu, dass im endgültigen Spiel die Autos nur geradeaus oder nach rechts fahren.

Nachdem die Autos ihrem Pfad nun folgen habe ich mich der Kollision gewidmet. Diese kam mir anfangs recht komplex vor, ist aber simpler gewesen als gedacht. Jedes Auto besitzt ein vorderes und ein hinteres Rectangle, welche an das Auto angeheftet sind. Diese bewegen sich bei jeder Bewegung mit dem Auto mit, da diese hierarchisch dem Fahrzeug untergeordnet sind. Die Hitboxen können in den Spieleinstellungen aktiviert werden, sodass diese sichtbar sind. Ebenfalls kann man die Kollision dort ausschalten, was das Spiel endlos weiterlaufen lässt, da ein Unfall die einzige Bedingung ist, unter der der Spieler das Spiel verliert.



Weiter ging es mit den Ampeln und das Verhalten der Autos mit diesen. Die Tastenbelegung zum Umschalten der Phasen änderte sich mehrmals über den Zeitraum der Entwicklung, da es immer vorkam, dass man die Taste gedrückt halten konnte um die Ampeln in jedem Frame umzuschalten. Um dieses Problem zu umgehen wurde das Wechseln auf die linke Maustaste gelegt, was, wenn auch nicht durch FudgeCore gelöst, immerhin richtig funktioniert.

Die Autos halten bei einer roten Ampel an der vorgesehenen unsichtbaren Haltelinie der Straße. Alle darauf an der Kreuzung ankommenden Autos sollen hintereinander in einem festen Abstand halten. Dies stellte sich als recht kompliziert heraus da jedes Auto „vorausschauend“ fahren muss. Insgesamt gab es unzählige Ideen und Ansätze für dieses Problem, welche allesamt entweder in bestimmten Szenarios oder programmiertechnisch nicht funktioniert haben. Am Ende habe ich mich dazu entschieden, dass jedes Auto ein Vektor zu jedem anderen Auto zieht und dann jeder dieser Vektoren durchgegangen wird, um zu prüfen ob dieser innerhalb seines Sichtkegels liegt.



Dies kam ebenfalls mit Problemen – Autos würden immer anhalten und das Spiel würde nie enden, da Kollisionen durch diese Funktion verhindert werden.

Schlussendlich gab es noch einige Designentscheidungen welche es nicht in die endgültige Fassung geschafft haben.

Das Konzept von speziellen Ereignissen, sogenannten Events war geplant und auch quasi fertig implementiert, jedoch reichte die Zeit nicht um alles reibungslos laufen zu lassen.

Es gab einen zweiten Knopf mit welchem der Spieler für 5 Sekunden lang alle Ampeln auf rot stellen konnte, welcher allerdings durch die fehlenden Events unnötig geworden ist. Der ursprüngliche Plan war, dass der Spieler diesen betätigen konnte, sollte z.B. ein Krankenwagen über die Kreuzung wollen, damit der Querverkehr keine Gefahr darstellt. Dieser Not-Aus-Knopf hätte auch als eine kurze Pause des Spielers agieren können, was allerdings nicht komplett durchdacht war, da sich in dieser Zeit weitere Autos an der Ampel sammeln. Deshalb, aber auch da der Knopf Probleme bei der Programmierung verursachte, wurde er schlussendlich entfernt.

Zum Ende des Projektes wurde noch ein Start- und Endscreen hinzugefügt.

Anforderungen

1	Nutzerinteraktion	Kontrolle der Ampelanlage
2	Objektinteraktion	Interaktion der Autos miteinander -> Unfälle | Interaktion der Autos mit der Kreuzung -> Ungeduld bei langem Warten, Zu langes Warten -> Über Rot fahren
3	Okjektanzahl variabel	Autos werden zur Laufzeit generiert
4	Szenenhierarchie	Siehe Diagramm
5	Sound	Ambientsounds, Hupen, "Crashsound"
6	GUI	Anzeigen des Scores
7	Externe Daten	Änderbare Einzelheiten zu Autos, Spielschwierigkeit etc. in der JSON
8	Verhaltensklassen	Kollision der Fahrzeuge, Vorrausschauendes Fahren, Abfahren des Weges und wenn nötig Anhalten
9	Subklassen	Node -> GameObject -> Vehicle, Trafficlight | Vehicle -> Car | GameObject -> Background
10	Maße & Positionen	Nullpunkt unten links, zwei Achsen (x,y) | Maße der Kreuzung: ~35x35 units | Maße eines Autos: 3x2 units
11	Event-System	Input - EventListener -> Steuerung | Load & Loop

Szenenhierarchie