# X. Code Erklärungen

# X.2 Parallaxing

Auf der folgenden Seite wird das Skript Parallaxing ausführlich erklärt. Das Skript wird in nahezu allen Szenen eingesetzt und beschäftigt sich mit den Hintergründen der Levels. Diese werden anhand der Bewegung des Spielers verschoben um einen dynamischen Eindruck zu übermitteln.

# X.2 Fortsetzung Parallaxing

Zu Beginn werden fünf Variablen festgelegt, welche im weiteren Verlauf noch genauer erläutert werden. Danach folgen drei Funktionen „Awake“, „Start“ und „LateUpdate“. Als Erstes wird die Funktion „Awake“ erörtert.

Beim Betreten einer Szene werden alle benötigten Skripte geladen, auch unser Parallaxing Skript. Dabei werden zunächst die „Awake“ Funktion aufgerufen. Sie wird bei uns dafür benutzt die Variable „cam“ mit der Hauptkamera zu belegen (Zeile 16). Auf diese Weise kann zukünftig leichter auf diese zugegriffen werden.

Nach der „Awake“ Funktion wird die „Start“ Funktion aufgerufen und initialisiert das Parallaxing. Am Anfang wird die aktuelle Kameraposition in „previousCamPos“ abgespeichert (Zeile 21). Dadurch kann man später berechnen in welche Richtung sich die Kamera bewegt hat.

Danach werden die „parallaxScales“ berechnet, sie werden benutzt, um hinterher ausrechnen zu können um wie viel der einzelne Hintergrund verschoben werden soll. Zunächst wird ein float-Array erstellt, welches die gleiche Länge hat wie „backgrounds“ (Zeile 23). Es kann also genau so viele Werte speichern wie es Hintergründe in der jeweiligen Szene gibt. Zum Berechnen der einzelnen Werte wird das Array nun in einer Schleife durchlaufen. Der Wert wird nun auf die Z-Position des Hintergrundes mal minus Eins gesetzt (Zeile 26). Es entscheidet also wie weit der Hintergrund von der Kamera entfernt ist wie schnell er sich bewegt, die weiter entfernt sind bewegen sich langsamer als die Hintergründe die näher an der Kamera dran sind. Dadurch wirkt der Hintergrund lebendiger, da sich weiter entfernte Objekte auch in der realen Welt scheinbar langsamer bewegen als wir selbst.

Sind alle Werte berechnet ist die „Start“ Funktion abgeschlossen. Als letztes widmen wir uns nun der Funktion „LateUpdate“. Diese wird bei jedem Frame des Spiels aufgerufen und führt das eigentliche Parallaxing aus. Um jeden Hintergrund einzeln verschieben zu können werden diese in einer for-Schleife durchlaufen. Innerhalb dieser wird vorerst die Variable „parallax“ berechnet (Zeile 34). Sie ergibt sich aus der alten und neuen Position der Kamera sowie dem zuvor berechneten „parallaxScales“ Wert. Die Variable sagt aus um wie viel der Hintergrund bewegt werden soll.

Mit dieser Variable kann jetzt die „backgroundTargetPosX“, die neue X-Koordinate des Hintergrundes, ermittelt werden (Zeile 36). Dafür wird die alte X-Koordinate mit der „parallax“ Variable addiert.

Aus der neuen X-Koordinate wird nun die neue Position des Hintergrundes, „backgroundTargetPos“ aufgestellt (Zeile 38). Dabei werden Y- und Z-Koordinate aus der alten Position übernommen und mit der neuen X-Koordinate kombiniert.

# X.2 Fortsetzung Parallaxing

Daraufhin kann das Parallaxing ausgeführt werden. Anstatt direkt dem Hintergrund die neue Position zuzuweisen wird „Lerp“ benutzt, um ein flüssigeres Bild zu erhalten. Die Methode „Lerp“ berechnet in Abhängigkeit von der Zeit einen Punkt zwischen der alten und der neu bestimmten Position. Wird ein früher Zeitpunkt übergeben gibt Sie einen Punkt näher an der alten Position zurück, bei einem späteren Zeitpunkt wird eine Position näher an der neueren Position zurückgegeben. Das Ergebnis von „Lerp“ wird dann dem Hintergrund als neue Position zugewiesen (Zeile 41).

Danach wird der Vorgang für den nächsten Hintergrund gestartet. Nach dem alle Hintergründe verschoben wurden, wird die aktuelle Kameraposition noch in „previousCamPos“ abgespeichert damit sie im nächsten Durchlauf benutzt werden kann (Zeile 45).

# X.3 Drohnen schießen

Bei dem folgenden Skript handelt es sich um den Angriff der Drohnen, die hauptsächlich im vierten Akt vorkommen. Allerdings wird ein ähnliches Skript auch für die anderen Gegner mit Fernkampf eingesetzt, welche es in jedem Akt gibt. Das Skript schießt sobald der Spieler in Reichweite der Drohne ist einen Laser auf ihn.

Zunächst werden alle benötigten Variablen definiert. In der einmalig beim Initialisieren der Drohne aufgerufen Methode „Start“ wird anschließend das Objekt des Spielers gesucht. Daraus wird danach die Position des Spielers entnommen und so abgespeichert, dass sie sich automatisch aktualisiert (Zeile 22).

# X.3 Fortsetzung Drohnen schießen

Wie bei dem Parallaxing wird auch die „Update“ Methode hier ein Mal pro Frame aufgerufen. In ihr wird zuerst getestet ob bereits ein Schuss abgefeuert wurde. Trifft dies nicht zu wird die Funktion „Attack“ als Coroutine aufgerufen (Zeile 29). Eine Coroutine wird verwendet da „Attack“ länger braucht als ein Frame lang ist. Würde sie normal gestartet werden müsste das gesamte Spiel warten bis der Schuss abgefeuert wurde. Die Coroutine pausiert die Methode sobald der Frame fertig ist und startet sie an der Stelle wieder sobald der nächste Frame anfängt. Dies ist besonders wichtig bei Methoden die Wartezeiten eingebaut haben wie unsere „Attack“ Funktion.

In „Attack“ wird zunächst die Variable „geschossen“ auf true gesetzt damit nicht noch ein Schuss abgefeuert werden kann (Zeile 35). Die Variable „geschossen“ war auch die, auf die in der „Update“ Methode getestet wurde. Als Nächstes wird gewartet, dadurch wird die Schussfrequenz der Drohnen reguliert (Zeile 36). Da die Drohne vor jedem Schuss zielen muss haben wir das Warten an den Beginn der Methode gesetzt. Danach wird geprüft ob sich der Spieler in der Reichweite der Drohne befindet. Dazu wird die Distanz zwischen dem Spieler und dem Gewehrlauf berechnet und mit dem Radius, in dem die Drohne Angreifen kann verglichen (Zeile 37).

Ist der Spieler in Reichweite der Drohne brauchen wir den Schusswinkel in dem abgefeuert werden soll. Dafür wird die Differenz zwischen der Spieler- und Drohnenposition im 3D-Raum ermittelt (Zeile 40). Wir benutzen einen 3D-Raum obwohl wir ein 2D-Spiel entwickeln, da beispielsweise der Spieler immerzu im Vordergrund sein soll, ein Hintergrund hingegen soll immer weiter hinten liegen. Aus der Differenz der Positionen können wir nun x und y entnehmen und dann mit der Hilfe des Arkustangens den Schusswinkel berechnen. Dieser wird anschließend noch mit der „Rad2Deg“ Konstanten in Grad umgewandelt (Zeile 41). Der ausgerechnete Winkel wird dann an die Drohne übergeben, dabei verwenden wir „Quaternion.Euler“ um die Winkel in eine Rotation zu verwandeln (Zeile 42).

Nach dem der Schusswinkel fest steht wird als nächstes die Kugel erzeugt. Wir klonen dafür mit der „Instantiate“ Methode die Originalkugel dazu geben wir noch die Position des Laufes sowie den Schusswinkel an (Zeile 44). Wir klonen sie, da alle Drohnen so auf die gleiche Originalkugel zugreifen können. So kann man nicht nur leicht das Aussehen aller Drohnenschüsse auf einmal ändern, sondern eine Drohne kann auch mehrmals hintereinander Feuern, ohne das die Originalkugel jedes Mal abgefeuert wird und deswegen nie ihr Ziel erreicht.

Damit sich nun der Laser in Richtung Spieler bewegt, wird dem „clone“ eine Kraft zugewiesen. Die Stärke der Kraft beziehungsweise die Geschwindigkeit des Lasers ist abhängig von der Variable „bulletSpeed“ (Zeile 46).

Anschließend wird die „geschossen“-Variable wieder auf „false“ gesetzt, damit die Drohne erneut einen Schuss abfeuern kann (Zeile 48).

# 3.1 Unity

Bei der Wahl der Laufzeit- und Entwicklungsumgebung von „Time Raider“ haben wir uns für Unity vor Unreal Engine entschieden. Im Hauptfenster, dem Editor, können Objekte per Drag & Drop erstellt werden und so die verschiedenen Level zusammengebaut werden. Daneben im Fenster „Inspector“ können danach die einzelnen Attribute der Objekte angepasst werden, also wie viel Leben ein Gegner hat, ob ein Objekt Schwerkraft hat und auch selbst erstellte Skripte werden hier hinzugefügt.

Doch das Meiste davon bieten alle Entwicklungsumgebungen, für Unity entschieden wir uns da es sehr einsteigerfreundlich ist, viele Lernmaterialien anbietet und auch Beispiele bereitstellt. Außerdem ist die Community sehr aktiv und stellt Tutorials zu nahezu jedem Thema im Bereich Spielentwicklung zur Verfügung. Leider unterstützt Unity nur C# und JavaScript und wir mussten uns zu Beginn in die Sprachen einarbeiten. Ein Vorteil von Unity hingegen ist, dass es für Projekte die weniger als 100.000€ Umsatz machen kostenlos verfügbar ist.

Leider fehlt Unity ein eingebautes Bildbearbeitungsprogramm und auch die Skripte mussten wir in einem extra Programm erstellen.

# 3.2 Inkscape

Da wir alle Hintergründe, Charaktere, Gegner und Items selbst zeichnen wollten brauchten wir ein Bildbearbeitungsprogramm. Unsere Wahl fiel dabei auf Inkscape.

Inkscape ist eine Software zur Erstellung und Bearbeitung von Grafiken. Wir entschieden uns gegen das am weitesten verbreitete Bildbearbeitungsprogramm Photoshop und für Inkscape aus drei wichtigen Gründen.

Der erste ist das Inkscape mit sogenannten Vektorgrafiken arbeitet. Diese haben keine feste Pixelgröße und können daher beliebig skaliert werden. Dies wird umgesetzt indem jeder Strich, Punkt, Kreis, etc. als einzelne Objekte gespeichert werden. Dadurch lassen sich auch Fehler leicht im Nachhinein beheben, da man das gewünschte Objekt einfach in der Hierarchie auswählen und bearbeiten kann.

Der zweite Punkt ist, dass Inkscape sehr intuitiv und leicht zu handhaben ist. Die meisten Features sind direkt per Maus ansteuerbar und keine komplizierten Tastenkombinationen müssen auswendig gelernt werden.

Der dritte Grund ist das Inkscape kostenlos verfügbar ist. Da nicht alle Teammitglieder Photoshop besaßen und dies momentan 24€ im Monat kostet war Inkscape eine willkommene Alternative.

# 3.3 Visual Studio

Um unsere Skripte zu Schreiben entschieden wir uns für Visual Studio. Es unterstützt eine Vielzahl von Programmiersprachen von Basic, über C# bis hin zu Python. Es markiert den Code farbig und schlägt automatisch Verbesserungen vor. Visual Studio wird außerdem von Unity empfohlen, da sich die beiden Programme miteinander verknüpfen lassen. Danach werden von alleine einige wichtige Packages eingebunden sobald man ein neues Skript beginnt und man kann direkt mit dem Programmieren beginnen. Auch dabei hilft die Verknüpfung, spricht man ein Objekt an werden direkt die Attribute vorgeschlagen oder andere in dem Objekt verwendete Skript angezeigt. Auch dieses Programm ist kostenlos sofern das Projekt weniger als fünf Entwickler hat oder Open-Source ist.

# 3.4 GitHub

Zu Beginn unseres Projekts entschieden wir uns für das von Unity gestellte Feature „Unity Teams“ um unseren Fortschritt untereinander auszutauschen. Leider wurde dieses Feature im Laufe unserer Entwicklung umgestaltet und war nur noch für Projekte mit bis zu drei Entwicklern kostenlos. Daher stiegen wir auf den kostenlosen GitHub Desktop Client um.

GitHub ist eine weit verbreitete Versionsverwaltungssoftware. Sie basiert, wie der Name bereits verrät auf Git und ist solange man seine Projekte öffentlich zugänglich macht kostenlos. Mit Ihr können wir von verschiedenen Computern aus arbeiten und trotzdem alle die gleichen Neuerungen erhalten. Hat beispielsweise Entwickler A ein neues Skript erstellt kann Entwickler B dieses einsetzten, ohne dass es über einen USB Stick oder ähnliches übertragen werden muss. Veränderungen können leicht zurückgenommen werden oder eigene „Branches“, also Kopien des gesamten Projekts zu einem beliebigen Zeitpunkt der Entwicklungsphase erstellt werden. Mit diesen kann man verschiedene Dinge ausprobieren, ohne das gesamte Projekt zu gefährden.

Der Desktop Client bietet dazu noch den Komfort, dass all Dies funktioniert ohne das man Befehle in die Kommandozeile eingeben muss. Mit wenigen Klicks sind wir so in der Lage unsere Neuerungen und Veränderungen untereinander auszutauschen und immer auf dem neusten Stand zu sein.