# 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.

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).

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

# 3.1 Unity

Bei der Wahl der Laufzeit- und Entwicklungsumgebung von „Time Raider“ haben wir uns für Unity vor Unreal Engine entschieden. Unity sehr einsteigerfreundlich ist und viele Lernmaterialien, sowie Beispielspiele bereitstellt. Außerdem ist die Community sehr aktiv und bietet unzählige Tutorials an. Unity unterstützt C# und JavaScript. Mit nur wenig Arbeit ist es möglich, Spiele für verschiedene Plattformen zu generieren.

# 3.2 Inkscape

Neben Unity brauchten wir eine weitere wichtige Software. 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 GitHub

GitHub wird zur Versionsverwaltung genutzt. Mit wenigen Klicks sind wir in der Lage unsere Neuerungen und Veränderungen untereinander auszutauschen. Zuvor haben wir „Unity Teams“ genutzt, welches in der Unity Engine eingebunden ist. Diese wurde allerdings im Laufe unseres Projekts kostenpflichtig wodurch nur noch drei Leute zusammenarbeiten konnten.