Inhalt

[1. Einleitung 2](#_Toc527571987)

[2. Spielablauf und -mechanik 3](#_Toc527571988)

[2.1 Spielidee 3](#_Toc527571989)

[2.2 Spielmechanik 3](#_Toc527571990)

[2.3 Spielbalancing 3](#_Toc527571991)

# 1. Einleitung

Bei dem Softwareprojekt „Time Raider“ handelt es sich um ein 2D-Plattformer Spiel, welches von vier Studenten der Hochschule Bochum entwickelt wurde. Wir haben uns für ein 2D-Spiel entschieden, da wir so die Bereiche Programmierung und Design verknüpfen und die Entwicklung abwechslungsreich gestalten können.

Das Spiel wird vorerst für Windows-PCs entwickelt. Mit Unity sind wir allerdings in der Lage „Time Raider“ kompatibel für weitere Betriebssysteme und Plattformen, wie beispielsweise einer mobilen Applikation, zu machen.

# 2. Spielablauf und -mechanik

# 2.1 Spielidee

Die Hauptfigur ist aufgrund einer Zeitreise ungewollt in die Vergangenheit gereist und muss wieder in die Gegenwart zurückreisen, allerdings wurde die Zeitmaschine zerstört. Ab hier übernimmt der Spieler die Kontrolle der Hauptfigur und bewegt sich durch verschiedene Orte aus unterschiedlichen Zeiten. Jeder der vier Akte spielt sich jeweils in einer Zeit ab. In jedem Akt beziehungsweise Zeit, gibt es jeweils eine verschiedene Anzahl an Level (Orte). Damit die Hauptfigur wieder in die Gegenwart gelangt, muss der Spieler verschiedene Aufgaben bewältigen. Gegner und Hindernisse aus den jeweiligen Zeiten versuchen den Spieler davon abzuhalten, die Aufgaben zu erledigen. Der Spieler muss Gegner und Hindernisse bewältigen, um Spielfortschritte zu erlangen.

# 2.2 Spielmechanik

Die Hauptfigur bewegt sich durch verschiedene 2D-Welten. Dieser kann nach links und rechts laufen beziehungsweise kriechen. Der Spieler startet mit 100 Lebenspunkten. Wird der der Spieler von Hindernissen oder einem Gegner getroffen, verliert er Lebenspunkte. Sobald der Spieler keine Lebenspunkte mehr besitzt, hat er das Level verloren und muss von Beginn des Levels anfangen. In jedem Level befinden sich Herzen, die der Spieler aufheben kann, um einen Lebenspunkt zu erhalten. Um Hindernisse bewältigen zu können, kann die Hauptfigur springen. Einige Hindernisse sind zum Beispiel Schluchten. Fällt man in diese hinein, kann man nicht mehr hinauskommen. Der Spieler verliert in diesem Fall Lebenspunkte und wird zum letzten sicheren Checkpoint befördert, den er zuvor erreicht hatte. Gegner können mit Fern- und Nahkampfwaffen angreifen und vom Spieler angegriffen werden. Mit dem Kaufsystem kann der Spieler sich mit Gegenständen ausrüsten, um sich stärker werdenden Gegnern zu stellen (siehe 2.3.2 Kaufsystem).

# 2.3 Spielbalancing

Im Laufe des Spiels werden die Hindernisse immer schwieriger zu absolvieren. Der Spieler kann sich zur Bewältigung der Hindernisse keine Vorteile erschaffen. Hier zeigt sich das Können des Spielers. Die Gegner hingegen werden stärker, indem ihre Lebenspunkte erhöht werden, sie mehr Schaden verursachen oder neue Angriffsmuster benutzen. Damit der Spieler die Möglichkeit hat, die Gegner zu besiegen, sollte er sich für den jeweiligen Akt ausrüsten. Dabei muss er sich das Kaufsystem zu Nutzen machen und passende Gegenstände kaufen, die ihm einen Vorteil verschaffen. Eine wichtige Rolle spielen zeitgemäße Fern- und Nahkampfwaffen, damit die Hauptfigur bei stärkeren Gegnern, mehr Schaden anrichten kann.

# 2.3.1 Gold

„Gold“ ist die Ingame Währung von Time Raider. Mit dieser Währung lassen sich Waffen in verschiedenen Shops in den Städten kaufen. Gold wird von den Gegnern fallengelassen oder liegt zufällig verteilt im Level herum. Dieses sammelt man automatisch auf, indem man über das Goldstück läuft. Die Menge an Gold wird in jeder Szene oben rechts angezeigt.

# 2.3.2 Kaufsystem

Das Kaufsystem ist ein wichtiger Bestandteil des Spiels. Wenn man diesen nicht nutzt, ist es nur schwer möglich, die nächsten Level zu meistern. Der Spieler sollte also in jedem Akt mindestens einmal den Shop aufsuchen und sich für die nächsten Level rüsten. Jeder Akt bietet einen Shop an, der zeitgemäße Nah- und Fernkampfwaffen anbietet. Diese können durch Gold gekauft werden. Der Shop befindet sich meistens in einer sicheren Zone, also in Städte und Dörfer und kann mithilfe des „Shop“-Button geöffnet werden.

# 2.3.3 Waffen

Wie bereits erwähnt gibt es zahlreiche Waffen. Diese unterteilen sich in Nah- und Fernkampfwaffen, welche der Spieler als auch die Gegner nutzen. Den Großteil der Waffen muss sich der Spieler im Shop kaufen. Abhängig der Situation hat jede Waffe ihre Vor- und Nachteile, die der Spieler zu seinem Vorteil nutzen sollte.

# 3. Installationsanleitung

# 3.1 Voraussetzungen

Für Time Raider wird ein Computer mit Windows Betriebssystem vorausgesetzt. Das Spiel lässt sich auch auf anderen Betriebssystemen und Geräten spielen, allerdings muss dies zusätzlich angepasst und exportiert werden. Außerdem sind die Anforderungen für den Computer nicht hoch, sodass das Spiel auf jedem halbwegs modernen Computer lauffähig sein sollte.

# 3.2 Installation

Die Installation erfolgt nicht durch einen Wizard, wie es normalerweise üblich ist, sondern durch das Aufspielen der Spieldateien von einem USB-Stick, einer CD oder einem Datenträger auf den jeweiligen Computer. Sobald sich alle Spieldateien auf dem Computer befinden, kann Time Raider gestartet und sofort gespielt werden.

# 3. Software

# 3.1 Unity

Bei der Wahl der Laufzeit- und Entwicklungsumgebung von „Time Raider“ haben wir Unity vor Unreal Engine bevorzugt, da 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.

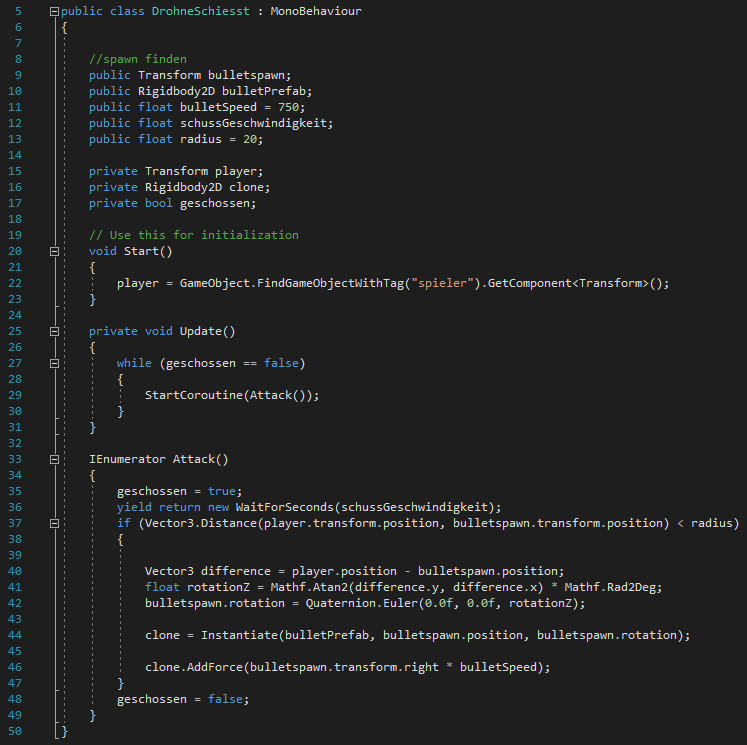
# 3.2 Inkscape

Inkscape ist eine Software zur Erstellung und Bearbeitung von zweidimensionalen Grafiken, welche vektorbasiert arbeitet. Dies hat den Vorteil, dass wir Grafiken ohne Detailverlust vergrößern oder verkleinern können. Dies erspart uns viel Arbeit, da wir uns nicht auf feste Maße einigen müssen. Mit vektorbasierten Grafiken sind wir in der Lage, zukünftig Grafiken verändern zu können, ohne diese auf eine bestimmte Größe anzupassen und zu exportieren, wie es zum Beispiel bei GIMP der Fall wäre.

# 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.

# X.3 Drohnen schießen



Bei dem folgenden Skript handelt es sich um die Attacke 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.

Zunächst werden alle benötigten Variablen definiert und die Position des Spielers wird beim Starten des Skriptes in der Methode Start() initialisiert.

In der Update()-Methode wird die Methode Attack() mit einer Couroutine aufgerufen (Zeile 29). Somit ist die Drohne in der Lage, die Methode Attack() mehrfach gleichzeitig auszuführen. Damit aber nicht unendlich viele Schüsse gleichzeitig abgefeuert werden und das Spiel abstürzt, wird der Schuss nur dann aktiviert, wenn die while-Bedingung wahr ist.

Bei der Attack()-Methode handelt es sich um einen IEnumerator. Dadurch kann man Verzögerungen einbauen, wie in diesem Fall das Abschießen eines Lasers nach der Anzahl der Sekunden, die in „schussGeschwindigkeit“ übergeben wurde. Damit aber nur ein Schuss abgegeben wird, wird die „geschossen“-Variable (Zeile 35) auf „true“ gesetzt. Nach dem Warten der Sekunden wird geprüft, ob der Spieler sich in der Nähe von der Drohne befindet (Zeile 37). Wenn diese Bedingung wahr ist, wird Differenz der Spieler- und Drohnenposition im 3D-Raum ermittelt. Da es sich allerdings um ein 2D-Spiel handelt, müssen wir diese von 3D in 2D mathematisch umwandeln (Zeile 41). Dann erhalten wir die Rotation des Lasers, die von der Drohne in Richtung des Spielers zeigt und übergeben diese an die Drohne (Zeile 42).

Bei dem Laser, der geschossen wird handelt es sich um ein „clone“. Dies ist notwendig, damit die Drohne nicht nur mit einem Laser schießen kann, sondern beispielsweise zwei oder drei Laserschüsse gleichzeitig abfeuern kann (je nachdem, wie „schussGeschwindigkeit“ eingestellt wurde). Geklont wird mit der „Instantiate“-Methode, welche als Parameter das Originalobjekt, die Position der Drohne und Rotation zum Spieler benötigt.

Damit sich nun der Laser in Richtung Spieler bewegt, wird dem „clone“ Kraft zugewiesen. Die Stärke der Kraft 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).