20. JANUAR 2017

PROJEKTDOKUMENTATION

ESCAPE

ZUMSTEIN SABINE, SCHÄR STEPHAN, UTZ MICHEL

Herbstsemester 2016

Inhalt

1 Einleitung

Für das Projekt 1 konnten wir uns ein Thema aus einer Auswahl von etwa zwanzig Themen aussuchen. Glücklicherweise erhielten wir die erste Wahl und so konnten wir uns auf das Unity-Projekt stürzen, welches mit dem Thema Adventure Game lockte. Was uns erwarten würde, wussten wir nicht genau. Wir hatten schon einiges von diesem Unity gehört, uns jedoch noch nie eine Erfahrung am eigenen Leibe gemacht. Also haben wir uns in das Abenteuer gestürzt und was dabei herausgekommen ist, wurde in den folgenden Seiten dokumentiert.

2 Vision

Am Anfang des Projekts wurde in Form eines Projektbeschriebs dargelegt, wie unsere Vorstellung von dem Endprodukt aussieht. Dabei wurde zuerst die Abgrenzung eines Adventure Games gegenüber anderen Spielgenres versucht, damit die Projektvorgabe „Adventure Game“ eingehalten wird. Das Adventure Game zeichnet sich dadurch aus, dass es von der Story und Puzzles lebt. Der Schwerpunkt liegt nicht auf Aufregung und Geschicklichkeit wie beispielsweise bei einem Action Game, sondern auf Spannung und Erlebnis. Es kann zwar auch andere Elemente zur Abwechslung enthalten, aber die Geschichte oder Puzzles machen den wichtigsten Teil aus. Oft könnte ein Adventure Game mit einem Film verglichen werden, bei dem der Spieler die verschiedenen Schauplätze aufsucht und dort meist nach dem Lösen von Rätseln die Fortsetzung erfährt. Das Spiel muss dabei nicht linear sein, sondern kann sich auch leistungsbedingt entwickeln; schneidet der Spieler bei Rätseln nicht gut ab, oder wählt er bei Dialogoptionen die unfreundlichen Antworten, so kann die Geschichte manchmal „weniger gut“ enden.

Auf dieser Grundlage wurde eine Spielidee entwickelt, die diese Elemente enthält. „Escape“ sollte das Spiel heissen und in etwa den real existierenden Spielen „Adventure Room“ oder auch „Mystery Room“ entsprechen. Diese Spiele setzen ein Konzept um, bei dem mehrere Personen zusammen in einem Raum eingesperrt werden und dann versuchen müssen, innerhalb einer vorgegebenen Zeitspanne den Weg hinaus zu finden. Dabei ist meist nicht das Finden des Wegs das Problem,

sondern die Hindernisse in Form von Schlössern, die durch das Lösen von Rätseln geöffnet werden können. Diese Spielidee versprach auch für ein Game einiges an Spannung und einen Aufbau, der modular ist und dem Projektteam bei der Entwicklung einige Freiheiten lassen sollte. Zusätzlich wurde eine Geschichte formuliert, damit auch das Element der Story nicht zu kurz kommt. Diese ist im nächsten Abschnitt kurz zusammengefasst.

In einer Fabrik werden künstliche Intelligenzen programmiert. Sind sie bereit, werden sie in einem Test-Haus überprüft und auf ihre Qualität hin getestet. Dafür wird eine K.I. jeweils auf die Festplatte unterschiedlicher Körper geladen, deren Kontrolle sie damit übernimmt. So kann sie einmal als Kugel und einmal als Roboter die Rätsel lösen. Labyrinthe, Denkaufgaben, typische Frage-Antwort-Spiele und viele andere Aufgaben soll die K.I. im Rahmen des Tests lösen. Je nachdem, wie gut ihr, gesteuert durch den Spieler, das gelingt, wird sie bewertet. Die Bewertung basiert je nach Rätsel auf die Schnelligkeit, Vollständigkeit oder Effizienz mit der es gelöst wird. Jedes Level wird also bewertet und aus allen Bewertungen zusammen ergibt sich ein ständig angepasster Index-Wert, der das Ende der Geschichte beeinflusst. Spielern, die nur einen tiefen Index-Wert erreicht haben, wird eröffnet, dass sie höchstens als schlechtes Beispiel in der Theorie auftauchen werden. Mittelgute Ergebnisse führen dazu, dass die K.I. zwar als akzeptabel eingestuft wird, aber nie produktiv eingesetzt, sondern höchstens als Grundlage für verbesserte Weiterentwicklungen verwendet wird. Schneidet der Spieler als K.I. besonders gut ab, so wird ihr eine glorreiche Zukunft als produktiv eingesetztes Programm prophezeit.

3 Endprodukt

Im letzten Kapitel wurde beschrieben, wie wir uns das Spiel zu Beginn vorgestellt haben. Im Folgekapitel wiederum werden die Änderungen an der Vision aufgezeigt. Um die Änderungen zu verstehen, muss man zuerst wissen, was umgesetzt wurde. Deswegen hier die Inhalte des Projektes:

- Hauptmenü
    - Ein 2D-Level, das als Dreh- und Angelpunkt dient. Von hier aus kann man die anderen Levels besuchen und die Punktzahl ist ersichtlich.
- Raum 0 – Intro
    - Der Benutzer wird in die Geschichte eingeführt. Die Kamera dreht sich um den Roboter Kyle (unsere Spielfigur) und die Informationen werden einem offengelegt.
- Raum 1 – Wächter
    - Ein begehbarer Level, in dem man sich in einem Raum mit drei Türen befindet und herausfinden muss, welche weiterführt. Hinter zwei von drei Türen lauert ein Wächter. Als Hilfestellung hat man Zugriff auf die Sicherheitskameras in den Räumen und kann sich so Hinweise erarbeiten, die bei der Lösung des Problems helfen.
- Raum 2 – Labyrinth
    - In diesem Level befindet sich der Spieler in einem Steinlabyrinth. Einziges sichtbares Lebewesen ist ein Pferd, das vorausläuft. Man steht bei der ersten Abzweigung vor der Entscheidung, ob man dem Pferd folgen soll oder nicht. Verteilt im Labyrinth befinden sich Luftaufnahmen der Umgebung, auf denen die aktuelle Position im Labyrinth rot markiert ist. Dies wurde als Hilfe eingebaut, weil der Level doch nicht ganz einfach ist. Gewonnen hat man, wenn man den Ausgang findet.
- Raum 3 – Adventure Room
    - Dieser Raum ist relativ klein, beinhaltet jedoch viele Objekte, die mit Klick darauf auch Informationen preisgeben. Ziel des Raumes ist es, den Code für die Türe herauszufinden. Wer nicht wissen will, wie man die Lösung findet, soll den Rest dieses Punktes überspringen. Die Lösung findet man wie folgt: Man beachte die kleinen runden Teppiche, welche in verschiedenen Farben die Zahlen 1, 2 und 7 bilden. Nun muss man diese noch richtig anordnen. Ein Hinweis dazu findet man im Bücherregal (einen Hinweis auf RGB – rot/grün/blau), womit man das

Lösungswort bestimmen kann. Zuerst die rote Eins, dann die grüne Sieben und am Schluss die blaue Zwei. Gibt man beim Elektronikkasten neben der Tür also 172 ein, hat man es geschafft!

- Raum 4 – Ende
    - Hier wird man, je nach Leistung in den vorhergehenden Levels, in eine gute oder in eine schlechte Situation geworfen. Das gute Ende beinhaltet ein Rampenlicht und man befindet sich mittendrin. Man wird gelobt und freut sich. Beim schlechten Ende befindet man sich auf einem Laufband, ist nicht fähig, sich zu bewegen und wird in Richtung eines Hammers bewegt und dann zerstört.

Das alles wurde mit vielen Zeilen Code, einigen selbst erstellten Modellen und eigenen Animationen realisiert.

4 Änderung an der Vision

In diesem Kapitel werden die drei wichtigsten Punkte beschrieben, die nicht wie geplant umgesetzt wurden.

4.1 Story

In dem Projektbeschrieb wurde als Eigenheit eines Adventure Games die Story hervorgehoben, da diese ein Unterscheidungsmerkmal von diesem Genre zu anderen ist. Es geht darum, dass der Spieler sich in einer Geschichte wiederfindet und durch das Spielen herausfinden kann, wie diese ausgeht. Bei manchen Spielen kann sie sich je nach Spielweise oder bestimmten Umständen unterschiedlich entwickeln und gibt dem Spieler damit die Motivation, das Spiel mehrmals zu spielen und dabei unterschiedliche Entwicklungen zu provozieren.

Die im Projektbeschrieb zusammengefasste Storyidee wurde in einem ersten Schritt noch nicht detailliert ausgearbeitet. Es war geplant, dass die Story nach einer anfänglichen Einarbeitungsphase, in welcher das Team ein gutes Verständnis für die Unity-Entwicklungsumgebung erreicht, ausformuliert wird. Jedoch hat sich gezeigt, dass zu einer Spiele-Entwicklung viel mehr gehört als angenommen; die Einarbeitungsphase hätte sich daher noch einige Monate verlängern lassen. Somit

blieb es bei der anfänglich grob umschriebenen Story und sie wurde nicht weiter ausgebaut. Da es sich um ein Modul handelte, das nebst der Projekterfahrung viele Freiheiten liess, wurde also bei dem aus Entwicklersicht weniger wichtigen Geschichte-Design Zeit gespart und der Schwerpunkt auf die Vervollständigung des Spieles gelegt.

4.2 Verschiedene Spielfiguren

Als weiterer Unterhaltungsfaktor war geplant, dass der Spieler seine Abenteuer als K.I. in verschiedenen Körpern absolvieren kann. Die K.I. sollte pro Raum auf die Festplatte von verschiedenen Robotern oder elektrischen Gegenständen gespeichert werden und so einmal rollend und einmal laufend ein Level bestreiten. Dies wurde aus zwei Gründen nicht umgesetzt: erstens gibt es im Unity Asset Store allgemein nicht viele kostenlose Spielfiguren und wenn, sind es keine elektrischen und sie sind meist auch noch nicht animiert. Zweitens ist es sehr aufwändig, ein eigenes Modell zu erstellen. Für eine graphisch und designerisch begabte Person wäre es eine Herausforderung gewesen, aber die Entwicklungszeit eines akzeptablen 3D-Charakters hätte bedeutet, dass wir uns in ein weiteres Gebiet begeben, von dem wir noch nicht viel wissen. Ein Teammitglied hat trotzdem einige Zeit investiert, dies ausprobiert (mit Blender, einem bekannten Programm zur Erstellung von 3D-Objekten) und beschlossen, dass in diesem Projekt darauf verzichtet werden muss. Es müsste zu viel Zeit investiert werden, damit etwas Gutes dabei herauskommt. Jedoch konnten wir eigene Animationen umsetzen und so einen Charakter beleben, der gratis vom Unity Asset Store heruntergeladen wurde.

4.3 State Machine

Ein weiterer Punkt, der während der Entwicklung einige Änderungen erfahren hat, war die globale Einrichtung einer State Machine. Sie wurde in jedem Komplexheitsgrade diskutiert, aber dann wurde beschlossen, diese einfach zu halten und erst einmal ein paar Räume zu gestalten, bevor man diverse Abhängigkeiten berücksichtigt. Angedacht war nämlich ein flexibles Spiel, bei dem der Spieler sich frei bewegen und mehrere Räumen parallel lösen könnte. Beim Erreichen eines Check Points, zum Beispiel beim Umlegen eines Schalters, hätte der aktuelle Raum seinen

State gewechselt und damit in einem anderen Raum einen Fortschritt bewirkt. Statt dies zu planen, bevor überhaupt ein einziger Raum beendet ist, hat man die Priorität auf das Fertigstellen der Level gelegt. Der grosse Vorteil bestand so in der unabhängigen Entwicklung der Räume; jedes Teammitglied konnte für sich arbeiten und musste sich noch keine Gedanken machen, ob Designentscheidungen für den eigenen Raum eine Konsequenz für einen anderen Raum mit sich ziehen könnten. Nachdem die Räume erste Formen angenommen hatten, wurde dann ein einfaches State Pattern umgesetzt. Dieses wird im nächsten Kapitel beschrieben, wie auch die Animationen, welche durch eine State Machine kontrolliert werden.

5 Globale Designs

5.1 Animation mit State Machines

Animationen in Unity funktionieren mit Hilfe von State Machines. Man kann Animationen erstellen, welche für sich alleine stehen können. Möchte man jedoch, dass ein Objekt je nach seinem Zustand anders animiert wird, muss man vom „Animator-Tab“ in der Unity-Entwicklungsumgebung Gebrauch machen. Dort kann man Animationen hinzufügen und mit Status-Übergängen verbinden. Angestossen wird ein Übergang jeweils durch eine sogenannte „Condition“ (Voraussetzung).

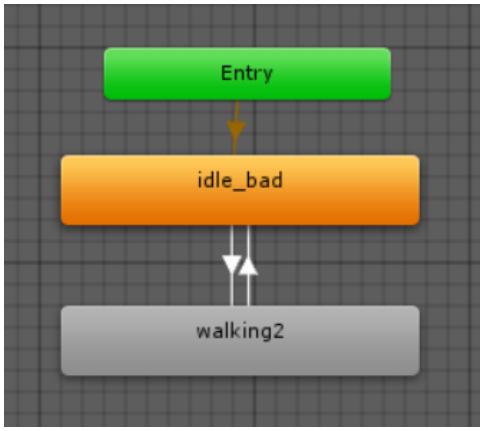

Abbildung 1 Beispiel einer State Machine in Unity für die Animation des Protagonisten

Conditions verwenden Parameter, welche wiederum fix einem State-Diagramm zugeordnet sind. Diese Parameter sind dann von überall her änderbar. So kann mit der Änderung eines Parameters ein Zustandsübergang angestossen werden.

Ein konkretes Beispiel ist die Geh- und Stillstandanimation einer Spielfigur. Bewegt sich der Spieler, soll die Figur auch gehend aussehen. Steht sie still, soll sie nicht laufen, sondern stehen. Dafür werden also eine IDLE- und eine Gehend-Animation gemacht und einem State-Diagramm hinzugefügt. Anschliessend wird ein Parameter "Walking" definiert, welcher kontrolliert, in welchem Status sich das Diagramm befindet. Standardmässig hat Walking den Wert „false". Wir definieren nun den Zustand „idle_bad" als Standardzustand und erstellen zwei Übergänge: Einer von idle_bad nach walking2 und einer, der den umgekehrten Weg geht. Beide Übergänge haben eine entsprechende Condition, welche sicherstellt, dass, wenn der Spieler anhält oder losläuft, der Übergang eingeleitet wird.

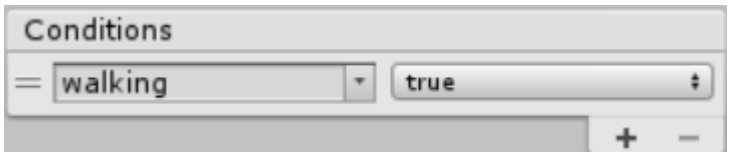

Abbildung 2 Beispiel einer Übergangsbedingung

Im Verlauf des Projekts wurden drei Animationen erstellt, welche auch verwendet werden. Es war dem Team ein Anliegen, die Animationen der Objekte eigenhändig zu erstellen, da dies ein zentraler Punkt in der Spieleentwicklung ist. Sie mögen nicht immer so rund und perfekt aussehen wie in aktuellen Xbox-, PC oder Playstation-Spielen, doch sie sind mit Liebe und Sorgfalt kreiert und arrangiert worden.

5.2 Spielstand

Wie vorhergehend in diesem Dokument bereits erklärt wurde, hat man auf eine globale State Machine verzichtet. Dafür haben wir die Animationen genauer angeschaut und jeden Level in seine eigene State Machine verwandelt. Der Status jedes Levels wird dann zusammen mit der aktuellen Punktzahl als „Spielstand" bezeichnet. Dieses Paket kann dann auf die Festplatte gesichert und beim Starten des Spiels neu geladen werden.

Jeder Level hat folgende Zustände:

- Gruppierungs-State
    - abstract, nur zur Gruppierung der States nach Level benötigt
    - Erbt von „BaseState“
- NotAllowed-State
    - Level wird in der Levelübersicht nicht angezeigt
    - Level kann nicht besucht werden
- NotStarted-State
    - Level wird in Level-Übersicht gelb angezeigt
    - Level ist spielbar
    - Implementiert das Interface „LevelPlayable“
- Started-State
    - Level wird in Level-Übersicht gelb angezeigt
    - Level ist spielbar
    - Implementiert das Interface „LevelPlayable“
    - Beinhaltet möglicherweise Zusatzinformationen (z.B. Anzahl Versuche)
- Finished-State
    - Level wird in Level-Übersicht grün angezeigt
    - Level ist spielbar
    - Implementiert das Interface „LevelCompleted“

Zusätzlich dazu gibt es die bereits erwähnte abstrakte Klasse „BaseState“, von welcher alle States im Spiel erben. Die beiden Interfaces „LevelPlayable“ und „LevelCompleted“ dienen der Anzeigesteuerung in der Level-Übersicht. So werden nur spielbare (implementieren „LevelPlayable“) oder abgeschlossene (implementieren „LevelCompleted“) Level überhaupt angezeigt.

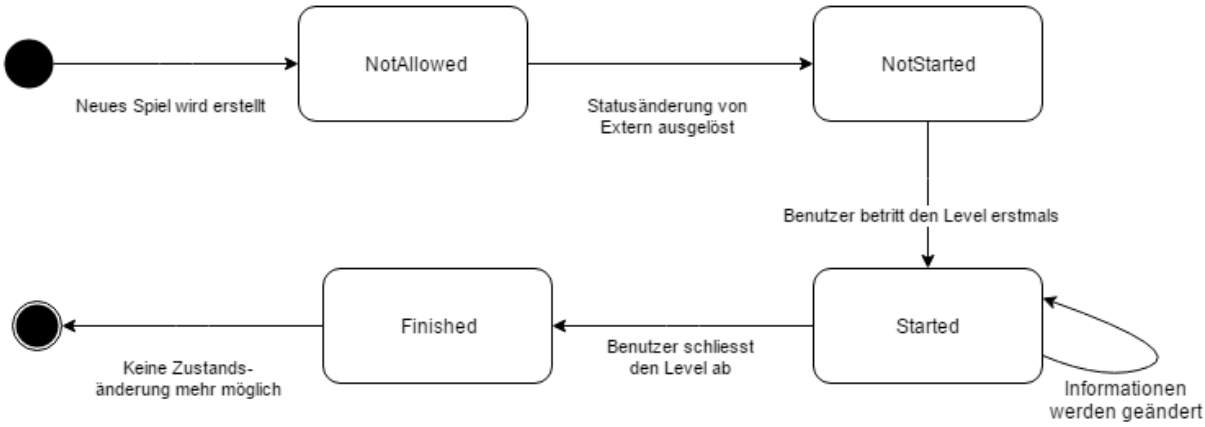

Abbildung 3 Konkrete Abfolge der Status innerhalb eines Raumes

Zustandsübergänge werden über statische Methodenaufrufe auf die Klasse „GameMemory“ erreicht.

6 Probleme und Lösungen

Eine Vielzahl an Eigenheiten von Unity haben dem Projektteam Probleme bereitet. Die Lösung einiger Probleme wurde im Wiki vom Github-Projekt (https://github.com/schas2/Project1/wiki) festgehalten. In den folgenden Abschnitten werden einige kurz erläutert.

6.1 Bauen eines Raumes

Da das geplante Spiel aus mehreren Räumen bestehen soll, liegt es nahe, dass in einem ersten Schritt ein solcher erstellt werden sollte. Dies ist jedoch nicht so einfach. Das Objekt „Plane“ wird als Boden in das Spiel gesetzt und „Quads“ als Wände, nicht etwa „Cubes“. Da die Game-Objekte noch keine Collider haben, muss von allen Seiten geschaut werden, dass die Wand keinen Abstand mehr vom Boden hat.

6.2 Collision

Eine wichtige Mechanik in Computerspielen ist die Auswertung von Kollisionen zwischen Spielelementen. So soll zum Beispiel verhindert werden, dass die Spielfigur sich durch die Wände bewegen kann.

In Unity gibt es zur Auswertung von Kollisionen eine eigene Komponentengruppe, die verschiedene Formen von „Colliders“ mitbringt. Diese Collider-Komponenten

können den entsprechenden Game Object hinzugefügt werden. Leider ist die Handhabung dieser Collider-Komponenten in Unity etwas umständlich gelöst worden. Zusätzlich zur Collider-Komponente wird ebenfalls die Rigidbody-Komponente benötigt. Der „Rigidbody“ ist eigentlich für die Physiksimulation von Unity zuständig. Wenn man nur seine Fähigkeiten zur Auswertung von Kollisionen benötigt, müssen die Physikeigenschaften am Rigidbody abgeschaltet werden.

6.3 Importieren von Objekten

Eine weitere kleinere Hürde stellt sich, sobald ein Objekt, eine Textur, oder eben eine Spielfigur für das Game gesucht wird. Standardmässig enthält Unity keine Gegenstände, aber viele können zusätzlich heruntergeladen werden. Dazu kann der Asset Store von Unity verwendet werden. Sucht man dort nach dem gewünschten Objekt, so findet sich sicherlich etwas Passendes. Allerdings meist nicht gratis. Da das Projektbudget stark beschränkt war, wurde beschlossen, auf die oft schöneren Dinge zu verzichten und mit den kostenlosen zurechtzukommen.

Wird ein Objekt aus dem Asset Store heruntergeladen und dann in das Projekt importiert, kommt dieses nicht als einzelne Datei. Zu dem Modell selber gehören Texturen, Material, Mesh und je nachdem noch zahlreiche andere Dateien. Wenn das Objekt in die Szene übernommen werden soll, kann die .fbx-Datei (erkennbar am Play-Symbol) auf das Spielfeld gezogen und dort platziert werden. Möglicherweise muss dem neuen Objekt nur ein Collider hinzugefügt werden und alles funktioniert, aber oft wurde das Objekt für eine ältere Unity-Version erstellt und es gibt Probleme. Es kann vorkommen, dass das Objekt gar nicht sichtbar ist. Öfter ereignete sich, dass zuerst alles zu klappen scheint. Das Objekt hat sogar schon einen Collider und somit auch die Möglichkeit, dass der Charakter damit interagiert; aber es gibt keine Reaktion. Dies liegt daran, dass bei den meisten heruntergeladenen Objekten durch die verschiedenen Unity-Versionen eine Verschiebung der Collider stattfindet. Das Spieleobjekt ist zwar am gewünschten Ort, aber der dazugehörige Collider befindet sich an anderen Koordinaten. Sobald das Problem bekannt ist, lässt es sich leicht beheben, indem der Collider an die gleichen Koordinaten wie das dazugehörige Objekt verschoben wird.

6.4 Kamera folgt der Spielfigur

In beinahe jedem Spiel ist es essentiell, dass der dargestellte Bildausschnitt auf die Spielfigur des Spielers fokussiert. Insbesondere bei 3D-Spielen ist aber die Kameraverfolgung der Spielfigur nicht immer ganz trivial. Unity bietet aber für dieses Problem eine gute Lösung. Im einfachsten Fall macht man aus der Kamera, die die Spielfigur verfolgen soll ein „Kind"-Objekt, indem man die Kamera in der Unity-Hierarchie zu einem Unterelement der Spielfigur macht. Somit vollführt die Kamera die gleichen Transformationen wie die Spielfigur, ohne dass man eine Zeile Code schreiben muss.

Dieser Ansatz hat aber Limitationen. Soll zum Beispiel eine rollende Kugel verfolgt werden, würde die Kamera die Rotation der Kugel ebenfalls durchführen und beim Spieler für Übelkeit sorgen. In diesem Fall kann ein Skript die Kameraführung übernehmen.

```
public class CameraController : MonoBehaviour
{
    public GameObject player;

    private Vector3 offset;

    void Start()
    {
        offset = transform.position - player.transform.position;
    }

    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

Abbildung 3 Skript zur Kamerakontrolle

Dieses Skript speichert beim Start des Spiels die relative Position der Kamera zur Spielfigur als 3D-Vektor. Die überschriebene Methode „LateUpdate" holt die neue Position der Spielfigur und positioniert die Kamera relativ zur Spielfigur neu. Die Methode „LateUpdate" wird bewusst verwendet, damit die Kameraneupositionierung möglichst nach der Positionsänderung der Spielfigur durchgeführt wird.

6.5 Navigation

Eine oft auftauchende Aufgabe in Spielen ist, dass man seine Spielfigur an einen bestimmten Ort auf dem Spielfeld hinbewegen will. Bei Computerspielen mit Maussteuerung hat sich eingebürgert, dass man auf den Ort klickt, wo sich die Spielfigur hinbewegen soll, um die Figur zu veranlassen sich an diesen Ort zu bewegen.

Bei Unity wird diese Art der Maussteuerung standardmässig mit Raytracing und einem Navigation Agent gelöst; beides Komponenten, die Unity von Haus aus beinhaltet. Der Navigation Agent kann einem beliebigen Game Object in Unity hinzugefügt werden. Die Komponente berechnet den Pfad für das Game Object und kann gegebenenfalls auch Hindernissen ausweichen, indem jeweils ein neuer Pfad berechnet wird. Um die Pfadberechnungen während der Laufzeit des Spiels zu vereinfachen, wird ein Navigation Mesh erzeugt. Das Navigation Mesh kennt alle statischen (d.h. zur Laufzeit nicht veränderlichen) Objekte im Spiel und beinhaltet alle „begehbaren“ Flächen im Spiel. Mithilfe dieses Navigation Mesh können die verwendeten Navigation Agent sich auf dem Spielfeld orientieren.

Damit der Navigation Agent ein Game Object bewegt, muss ihm zuerst ein Zielpunkt angegeben werden. Dies kann mit Raytracing bewerkstelligt werden. Als erstes erstellt man dazu ein Ray-Objekt, das von der Kamera aus Richtung Mauszeiger zeigt. Dieses Ray-Objekt wird der Raytracing-Funktion von Unity mitgegeben. Raytracing verfolgt diesen «Strahl» und kontrolliert, ob es zu einer Kollision mit einem anderen Game Object kommt. Falls es eine solche Kollision gibt, können die Koordinaten ausgelesen und dem Navigation Agent übergeben werden. Der Navigation Agent berechnet dann mithilfe des Navigation Mesh den Pfad zum Zielpunkt und bewegt das Game Object dorthin.

6.6 Verwendung mehrerer Szenen

Die Spielwelt in Unity besteht aus einer oder mehrerer sogenannter „Scenes“. Es ist naheliegend, pro Level in einem Spiel eine eigene Scene zu verwenden. Umso erstaunlicher ist die unintuitive Handhabung des Scene-Wechsels in Unity. So werden

die Scenes von Unity über eine Id verwaltet, die nur im „Build Settings“-Menü einsehbar ist. Diese Id kann auch nicht manuell gestzt werden, sondern wird durch die Reihenfolge der Scenes im „Build Settings“-Menü bestimmt.

6.7 Skript-Beispiel in eigenes Programm integrieren

Wer in Unity etwas erstellen will, ist wahrscheinlich nicht der Erste, der etwas in der Art und Weise macht. Daher kann man davon ausgehen, dass jedes Problem schon von einem Mitglied der Unity-Community gelöst wurde. Das tönt also gut – ob in Stack Overflow (Forum für Programmierprobleme) oder auf der offiziellen Unity-Hilfe-Seite: Lösungen sind vorhanden. Nicht ganz einfach ist es nun, diese Lösungen in das eigene Programm zu integrieren. Dabei gibt es verschiedenste Probleme. Nachfolgend werden zwei davon genauer beschrieben.

6.7.1 Verschiedene Unity-Versionen

Die Lösungen mögen noch so zahlreich zu finden sein, doch ein Grossteil ist in der aktuellen Unity-Version gar nicht mehr anwendbar. Die Umgebung scheint sich sehr schnell zu ändern und auch Lösungen, welche vor einem Jahr noch funktioniert haben, bringen heute nicht mehr die erwünschte Wirkung. So sucht man in der Regel eher lange in den zahlreichen Antworten.

6.7.2 Aufbau

Der Aufbau der Applikationen in Unity ist nicht so, wie man es sich von gewöhnlichen Haus- und Büroanwendungen gewohnt ist. Wenig funktioniert linear: Anfragen, Antworten und sonstige Events bewegen sich zwischen unzähligen Objekten und man fragt sich bei der Lösungsfindung des Öfteren, wo genau man die pfannenfertige Lösung implementieren muss.

6.8 Vergleich mit herkömmlichen Applikationen

Das Projektteam hatte vor dieser Applikation durchaus schon Programmiererfahrung, doch einer Umgebung dieser Art sind wir noch nicht begegnet. Wir sind es uns gewohnt, dass die Dinge etwas linearer ablaufen als in Unity. Um auf einem Objekt Methoden aufzurufen, braucht man in unserer Welt eine Referenz auf das Ziel. In Unity sind die Skripte in Game Objects eingepackt und es reicht vollends, dieses

Objekt zu kennen und eine Art Nachricht zu senden. Diese Nachricht wird vom Game Object dann interpretiert und als Methodenaufruf im Skript verarbeitet. Es ist auch möglich, auf einer Objektreferenz eine Methode direkt aufzurufen.

Wir kannten Multithreading und Parallelität. In den bisher bekannten Projekten war aber immer noch eine gewisse Gebundenheit an eine zentrale Ausführungslinie gegeben. Threads werden zum Beispiel normalerweise irgendwo gestartet und laufen einfach ab dem Zeitpunkt selbstständig. Der Ursprung ist aber offensichtlich ein Haupt-Thread, welcher die verschiedenen Threads startet. In Unity macht eigentlich jedes Objekt von Anfang an was es will und lässt sich dabei nicht gross stören. Die meisten Objekte werden regelmässig aufgerufen (Update-Funktion), nur um zu schauen, ob sie gerade etwas machen möchten. Eine Szene in Unity besteht also nicht aus Objekten, welche von einem Hauptobjekt aus erstellt wurden und eventuell noch verwaltet werden. Eine Unity-Szene besteht aus einer wilden Schar unabhängiger Objekte, welche untereinander kommunizieren und deren Lebensspanne nicht von einer zentralen Instanz diktiert wird.

Diese Umstellung war sehr interessant und spannend. Wir haben die Unterschiede nicht gewertet, sondern uns einfach damit abgefunden, dass die Softwareentwicklung in Unity anders funktioniert, als wir es gewohnt waren.

7 Schlussfolgerung

Das Projekt 1 hat uns ermöglicht, uns im Rahmen des Unterrichts mit der Unity Game Engine bekannt zu machen. Uns ist bewusst geworden, was mit Unity alles möglich ist und wir konnten vieles ausprobieren. Rückblickend betrachtet würden wir uns möglicherweise etwas mehr auf die Geschichte fokussieren und die Steuerung nicht über Mausklicks, sondern über die Pfeiltasten realisieren. Obwohl diese Punkte im Hinterkopf herumschwirren, sind wir zufrieden mit dem Resultat. Die Hauptsache – den Spass am Projekt – haben wir erreicht. Zudem haben wir ein lauffähiges Spiel entwickelt, welches wir bei zukünftigen Projekten als Referenz verwenden können. In Zukunft werden wir schneller beim Entwickeln sein, da wir uns mittlerweile besser in der Entwicklungsumgebung auskennen. Wir wissen die mächtige Physik-Umgebung einzusetzen, kennen die wichtigen Menüpunkte des Unity-Studios, kennen uns im Unity-Ökosystem aus und werden dadurch effizienter. Wir haben wertvolle Erfahrungen gesammelt, welche im weiteren Verlauf des Studiums zumindest für zwei Projektmitarbeiter, welche CPVR als Spezialisierung gewählt haben, nützlich sein werden.

8 Glossar

Adventure Room:	Spiel, bei dem eine Gruppe von Leuten sich freiwillig in einen Raum sperren lässt und dann durch Rätsellösen probiert, innerhalb einer Stunde auszubrechen. Zum Beispiel: http://bern.adventurerooms.ch/infoar/
Blender:	Open Source Programm zur Erstellung von 3D-Objekten (u.a.). Siehe: https://www.blender.org/
Collider:	Eine Komponente bei Unity, welche die physikalische Reaktion beim Zusammenprallen von Objekten ermöglicht.
Idle:	Zustand eines Spielcharakters, wenn dieser sich nicht bewegt.
K.I.:	Künstliche Intelligenz. In diesem Projekt zusätzlich der durch den Spieler gesteuerte Charakter (im Körper Roboter Kyle).
Kyle (Roboter):	Roboter-Charakter, der als gratis-Asset im Unity Store heruntergeladen werden kann (enthält noch keine Animationen).
Material:	Materials können als Oberfläche von Objekten verwendet werden (und werden dafür auf das Mesh gelegt). Sie enthalten Informationen zur Oberfläche. Ein Material enthält daher auch eine Texture.
Mesh:	Netz aus Knoten und Kanten, welches die Dimensionen eines 3D-Objekts darstellt.
Raytracing:	Ein Objekt in Unity, welches dazu verwendet werden kann, im Spiel imaginäre Pfeile abzuschiessen und zu erkennen, welche Spieleobjekte davon getroffen werden. Kann zum Beispiel verwendet werden, um zu erkennen, ob zwei Objekte eine freie Sicht zwischen sich haben oder ob Objekte die Sicht versperren.

Rigidbody:	Durch das Hinzufügen eines Rigidbody zu einem Game Object in Unity, wird dieses von dessen Physik Engine berücksichtigt.
State Machine:	Eine Darstellung von Zuständen und Übergängen.
Texture:	Besteht aus Informationen zur Oberflächenbeschaffenheit und kann einem Objekt zugeordnet werden. Je nach Texture kann das Objekt so eine raue oder eine flauschige Oberfläche, mit oder ohne Glanz besitzen.
Thread:	Abgekoppelter Prozess/Ablauf in der Programmierung; mehrere Threads können parallel ausgeführt werden.
Unity Asset Store:	Ein in Unity integrierter Store, der Objekte enthält, die in ein Unity-Projekt importiert werden können. Diese Objekte können von anderen Usern erstellt worden sein und viele kosten etwas.

9 Quellen

S. Schär, M. Utz, S. Zumstein, „Project1“, 2016, Github-Repository, https://github.com/schas2/Project1

R. M. Fujimoto, „Parallel and Distributed Simulation Systems“, 2000, John Wiley & Sons, Inc. New York, NY, USA

Unity Manual, https://docs.unity3d.com/Manual/index.html

9.1 Verschiedene Quellen mit Antworten zu Unity-Problemen

Hier einige Beispiele von Lösungshilfen, welche im Internet zu den verschiedenen Problemen zu finden sind und von uns für das Projekt zu Rate gezogen wurden.

http://answers.unity3d.com/questions/47826/best-way-to-send-variablesdata-to-other-Skripts.html

https://unity3d.com/de/learn/tutorials/projects/roll-ball-tutorial/displaying-score-and-text?playlist=17141

https://docs.unity3d.com/Manual/PositioningGameObjects.html

https://gamedevelopment.tutsplus.com/tutorials/how-to-save-and-load-your-players-progress-in-unity--cms-20934