Kantonsschule im Lee xxxxxxxxxxx

Fachschaft Informatik xxxxxxxxxx

Rychenbergstrasse 140 xxxxxxxxxxxx

8400 Winterthur

Maturitätsarbeit HS 2018/19

# SimpleRPG

**Ein einfaches Rollenspiel in Unity 3D**

### Erschaffen eines Spiels von Null

XXXXXXX Klasse XX

Betreuer: XXXXXX Eingereicht am 7. Januar 2019

an der Kantonsschule im Lee in Winterthur

# Inhaltsverzeichnis

1. [Einleitung 1](#_TOC_250057)
2. [Grundlagen 2](#_TOC_250056)

[2.1 Werkzeuge . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2](#_TOC_250055)

[2.1.1 Unity 3D . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2](#_TOC_250054)

[2.1.2 Blender . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3](#_TOC_250053)

[2.1.3 LaTeX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4](#_TOC_250052)

[2.1.4 MonoDevelop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4](#_TOC_250051)

[2.1.5 Git . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4](#_TOC_250050)

[2.1.6 astah UML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4](#_TOC_250049)

[2.1.7 Paint . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250048)

[2.2 Programmiersprache und Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250047)

[2.2.1 C# . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250046)

[2.2.2 Unity Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250045)

2.2.2.1 Wichtigste Basisklassen . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2.2.2 Input-Manager . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

[2.2.3 Physik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7](#_TOC_250044)

2.2.3.1 Rigidbody . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.2.3.2 Kollisionen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

1. [Prozess 9](#_TOC_250043)

[3.1 Allgemein . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250042)

[3.1.1 Arbeitsdokumentation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250041)

[3.1.2 Inspirationsquellen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250040)

[3.1.3 Vorgehen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250039)

[3.1.4 Code-Umbau . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250038)

* 1. [Problemstellungen und deren Umsetzung 10](#_TOC_250037)
     1. [Spielführung 10](#_TOC_250036)
        1. Menü (Klasse) 11
        2. GameData (Klasse) 12
        3. Head-Up-Display (Klasse HUD) 12
     2. [Spieler 15](#_TOC_250035)
        1. Player (Klasse) 15
        2. Drehbewegung des Spielers (Klasse MouseLookAtIt) 17
     3. [Animationen 17](#_TOC_250034)
        1. Die verschiedenen Animationen 18
        2. Erstellen einer Animation 18
        3. Animator 19
     4. [Waffen 19](#_TOC_250033)
        1. Stats (Klasse) 20
        2. WeaponHit (Klasse) 20
        3. WeaponManager (Klasse) 21
     5. [Figuren 22](#_TOC_250032)
        1. InteractibleObject (Klasse) 22
        2. Zufallsbasiertes Generieren von Objekten 23
        3. Autonomer Gegner NPC (Klasse) 24
        4. TheThirdKind (Klasse) 26

i

Inhaltsverzeichnis ii

* + 1. [Aufgaben 26](#_TOC_250031)
       1. Quest (Klasse) 26
       2. Quests (Klasse) 27
    2. [Fehlersuche mit Konsolenausgabe 27](#_TOC_250030)
  1. [Versionsverwaltung 29](#_TOC_250029)
  2. [Verteilung und Test 29](#_TOC_250028)
     1. [Verteilung über Github 29](#_TOC_250027)
     2. [Testphase 29](#_TOC_250026)

1. [Resultate 30](#_TOC_250025)
   1. [Erkenntnisse 30](#_TOC_250024)
   2. [Ausbaumöglichkeiten 30](#_TOC_250023)
      1. [Erweiterung des Spiels 30](#_TOC_250022)
      2. [Spielerlebnis vertiefen 30](#_TOC_250021)
      3. [Weitere Verbreitung 30](#_TOC_250020)
      4. [Betriebssysteme 30](#_TOC_250019)
2. [Schluss 32](#_TOC_250018)
   1. [Reflexion 32](#_TOC_250017)
   2. [Download und Kontakt 32](#_TOC_250016)
   3. [Danksagung 32](#_TOC_250015)
3. [Skripte 33](#_TOC_250014)
   1. [GameData.cs 33](#_TOC_250013)
   2. [HUD.cs 35](#_TOC_250012)
   3. [Player.cs 36](#_TOC_250011)
   4. [InteractibleObject.cs 40](#_TOC_250010)
   5. [Menu.cs 41](#_TOC_250009)
   6. [MouseLookAtIt.cs 42](#_TOC_250008)
   7. [NPC.cs 42](#_TOC_250007)
   8. [Stats.cs 45](#_TOC_250006)
   9. [TheThirdKind.cs 46](#_TOC_250005)
   10. [WeaponHit.cs 48](#_TOC_250004)
   11. [WeaponManager.cs 49](#_TOC_250003)
   12. [Quest.cs 51](#_TOC_250002)
   13. [FreedomQuest.cs 52](#_TOC_250001)
   14. [Quests.cs 52](#_TOC_250000)

Erstaunlicherweise ist dies tatsächlich mein dritter Anlauf einer Maturaarbeit. Beim ersten Mal wollte ich eine Geschichte schreiben. Dass das nicht geklappt hat, war nicht weiter tragisch, denn ich hatte noch nicht so viel Energie darin investiert. Beim zweiten Mal war es anders. Ich hatte schon gut 5 Monate lang recherchiert und programmiert, um eine KS imLee-App zu erschaffen. Neben Features wie z.B. individuelle Meldungen bei ausgefallenen Lektionen, sollte auch ein persönlicher Stundenplan dabei sein.

Nach einer Mathematikstunde nahm mich mein Lehrer und damaliger Betreuer XXXXX zur Seite und überbrachte mir die schlechte Nachricht: Die Extranet-Server, von denen ich die Infos jeweils abholte, sollten in zwei Wochen abgeschaltet werden. Um die neuen Server benutzen zu können, müsste man sich mit der Zentrale in Zürich in Verbindung setzen. An dieser Stelle musste ich die Arbeit verwerfen, da es zu rechtlichen Problemen mit den Betreibern der Intranet-Version kam (dies wäre die zweite mögliche Quelle für meine Daten gewesen).

Bei der Wahl des neuen Themas (und somit auch indirekt des Titels) war es mir wichtig, dass der Name Programm ist. SimpleRPG steht dafür, dass es ein einfaches Spiel ist, in welchem die Schlüsselelemente eines Rollenspiels (RPG = Role Playing Game) vorhanden sind.

Warum die Betonung auf einfach? Aktuell ist es der Fall, dass bei nahezu jeder Videospielentwicklung auf einen Programmierer rund zehn Grafiker kommen. Ich als Einzelperson wollte aber mehr programmieren als Graphiken gestalten, es sollte schliesslich eine ICT und keine BG Arbeit sein. Ich hätte die Option gehabt, Unity Assets, eine Sammlung von vorgefertigten Elementen, Figuren, Umgebungselementen etc. herunter zu laden. Diese wären aber nicht selbst gemacht und somit nicht vereinbar damit, dass ich das Spiel selber von Grund auf selbst erstellen wollte.

Und so kam es zu meinem finalen Maturprojekt: SimpleRPG ein Spiel, welches das Genre des RPGs auf seine Grundbausteine herunterbricht. Geschmückt mit einer Geschichte rund um die Spielfigur geht es um Aufgaben, das Sammeln von Gegenständen und die Steigerung der Fähigkeiten während des Verlaufs.

1

## Werkzeuge

Im Folgenden beschreibe ich die Werkzeuge, die ich für die Erstellung meiner Maturaarbeit gebraucht habe.

### Unity 3D

Unity 3D lernte ich bereits im Alter von 11 Jahren kennen. Damals tat ich mich sehr schwer mit der englischen Sprache, konnte aber schon mit der Hilfe eines Buches und meines Vaters kleinere Spiele programmieren. Dazu kommt, dass Unity im Unterschied zu anderen (teils kostenpflichtigen) Programmen recht einfach zu handhaben ist. Durch diese Einfachheit büsst es aber nichts an Optionen und Möglichkeiten ein. Deshalb habe ich mich für Unity als Programmierumgebung meiner Maturaarbeit entschieden. Die Umgebung ist folgendermassen aufgebaut1:

Abbildung 2.1: Unity Benutzeroberfläche

Scene View (1)

Dieses Fenster ermöglicht das interaktive Bearbeiten von Szenen und Objekten. Man kann sich in der Spielwelt überall hin und durch alle Objekte bewegen. Um Objekte zu editieren, kann man sie mit X, Y und Z Achsen positionieren, skalieren und drehen.

Game View (2)

Dieses Fenster zeigt eine Vorschau des Spiels. Sobald *Play* gedrückt wird, beginnt man das Spiel. Um Ver- änderungen auszutesten, kann man während des Spiels pausieren, Veränderungen vornehmen und das Spiel mit diesen Veränderungen fortsetzen. Sobald der Spielvorgang durch ein zweites betätigen des *Play* Buttons gestoppt wird, werden diese Veränderungen wieder rückgängig gemacht.

1Unity. *Unity Overview*. 2018. URL: https://docs.unity3d.com/Manual/UnityOverview.html.

2

Hierarchy (3)

Dieses Fenster zeigt alle in der aktuell geöffneten Szene existierenden Objekte und deren Hierarchiestruktur.

Inspector (4)

Dieses Fenster zeigt alle öffentlichen Parameter und Komponenten des aktuell ausgewählten Objekts (Game- Object) an.

Project Browser (5)

Dieses Fenster dient der Navigation durch die Projektdateien (Assets). Alternativ wird in diesem Bereich die Konsole angezeigt, ein Fenster mit allen Outputs (mittels Debug.Log), Warnungen und Fehlermeldungen.

Game-Engine

Unity liefert nicht nur die Programmierumgebung, sondern auch eine selbständige Game-Engine2, ein spezi- elles Programm, welches die grundlegende Funktionalität für den selbständigen Ablauf und die Steuerung des Spieles zur Verfügung stellt, ohne dass es dafür die Programmierumgebung braucht.

### Blender

Blender3 ist eine frei verfügbare 3D Modellierungssoftware. 3D Modelle lassen sich damit viel besser und genauer bearbeiten als mit dem Standard Unity Editor. Zum Beispiel wäre das Erstellen einer komplexen Blüte in Unity ohne sehr grossen Zeitaufwand nicht möglich gewesen.

Abbildung 2.2: 3D Modell der Blume in Blender

2Unity. *Unity Glossary*. 2018. URL: https://docs.unity3d.com/Manual/Glossary.html.

3Blender Foundation. *Blender*. 2018. URL: [https://www.blender.org](http://www.blender.org/).

### LaTeX

Gemäss Vorgabe verfasste ich den schriftlichen Teil der Arbeit in LaTeX4, basierend auf einer Vorlage der Kantonsschule Wattwil5. Als Editor verwendete ich Texmaker6.

### MonoDevelop

Scripts wurden in MonoDevelop7 entwickelt. Diese Umgebung verfügt über einen hochwertigen Debugger, welcher die Fehlersuche zur Laufzeit stark erleichtert.

Abbildung 2.3: Breakpoint im MonoDevelop Debugger mit Anzeige der Variablen

### Git

Für die Versionsverwaltung verwendete ich das weitverbreitete Programm Git8. Als Oberfläche kam Source- tree9 zum Einsatz. Ich erstellte mir einen Studentenzugang auf Github10. Damit wurde die Kommunikation und das schnelle Austauschen mit meiner Betreuungsperson einfacher, da man sich nicht mehr für alles treffen musste.

### astah UML

UML Diagramme zeichnete ich mit der gratis Studentenversion von astah UML11.

4The LaTeX Project. *LaTeX*. 2018. URL: [https://www.latex-project.org](http://www.latex-project.org/).

5Simon Schälli. *Kantonsschule Wattwil - Maturaarbeit Template*. 2016. URL: https : / / www . overleaf . com / latex / templates/kantonsschule-wattwil-maturaarbeit-template/pswptsnytrgx.

6Pascal Brachet. *Texmaker*. 2018. URL: [www.xm1math.net/texmaker.](http://www.xm1math.net/texmaker)

7MonoDevelop Project. *MonoDevelop*. 2018. URL: [https://www.monodevelop.com](http://www.monodevelop.com/).

8Git. *Git*. 2018. URL: https://git-scm.com.

9Atlassian. *Sourcetree*. 2018. URL: [https://www.sourcetreeapp.com](http://www.sourcetreeapp.com/).

10GitHub Inc. *GitHub*. 2018. URL: https://github.com.

11ChangeVision Inc. *astah UML*. 2018. URL: https://astah.net/editions/uml-new.

### Paint

Jegliche Texturen habe ich mit dem auf jedem Windows-PC vorinstallierten Microsoft Paint erstellt, da es sehr einfach zu handhaben ist, und ich keine weiteren Funktionalitäten als die eines einfachen Malprogrammes brauchte.

## Programmiersprache und Framework

### C#

C# ist eine objektorientierte Programmiersprache, die von Microsoft entwickelt wurde12. Unity kann sowohl in C# als auch in Javascript programmiert werden. Letzteres kannte ich schon, also habe ich die neue Sprache gewählt, um mein Basiswissen der Informatik zu vergrössern.

### Unity Framework

Die Funktionalitäten (Klassen, Skripts etc.) bauen auf dem von Unity (für nicht kommerzielle Zwecke gratis) zur Verfügung gestellten Framework auf. Ein Framework ist eine Sammlung von vorgefertigten Klassen, deren Verwendung enorm viel Zeit einspart, da sie meistens einwandfrei funktionieren und somit insgesamt weni- ger Zeit für die Fehlersuche aufgewendet werden muss. Gleichzeitig dient es als Grundlage für die eigenen Erweiterungen.13 Hier sind die zwei meist benutzten Basisklassen aufgeführt:

* + - 1. **Wichtigste Basisklassen**

Abbildung 2.4: Basisklassen in Unity

GameObject

14 GameObject ist die Basisklasse für alle Objekte, die auf der Benutzeroberfläche erstellt werden. Die wich- tigste Eigenschaft eines GameObject ist transform. Dieses beinhaltet die Position, Drehung und Skalierung im Raum. GetComponent<Komponentenklasse> ist die meist benutzte Methode. Sie liefert die Komponen- te z.B. das Kollisionsobjekt oder Skripts des GameObject. SetActive aktiviert oder deaktiviert das Objekt. Ein inaktives GameObject wird unsichtbar und erhält keine Aufrufe mehr.

12Microsoft Corporation. *Leitfaden für C#*. 2018. URL: https://docs.microsoft.com/de-de/dotnet/csharp/.

13Unity. *Unity Script Reference*. 2018. URL: https://docs.unity3d.com/ScriptReference/index.html.

14Unity. *Unity Script Reference, GameObject*. 2018. URL: https : / / docs . unity3d . com / ScriptReference / GameObject.html.

MonoBehaviour

15 MonoBehaviour ist die Basisklasse wenn ein GameObject mit einem Skript erweitert wird. Die zwei geerbten Methoden die standardmässig überschrieben werden sind:

Start() wird zu Laufzeitbeginn einmal aufgerufen. Es wird benutzt um Initialisierungen durchzuführen.

Update() wird jedes Mal aufgerufen bevor ein neues Bild des Spiels berechnet wird. In abgeleiteten Klassen werden in dieser Methode deshalb Elemente wie Bewegungen fortlaufend aktualisiert. SendMessage übermit- telt einen Aufruf an alle Komponenten eines Objekts.

* + - 1. **Input-Manager**

Der Inputmanager16 dient der virtuellen Steuerung des Spielers. Er kann z.B. nach Achsen (vorwärts, seit- wärts) abgefragt werden und deren Intensität mitliefern. Die Zuordnung von Tasten zu Achsen kann dabei angepasst werden:

Abbildung 2.5: Der Inputmanager mit horizontaler und vertikaler Achse, zugeordnet an die Tasten a-d/s-w Die Abfrage sieht im Code folgendermassen aus:

1 void Update ()

2 {

3 (...)

4 // x und z Koordinaten Bewegung

5 float x = Input.GetAxis ("Horizontal") \* Time.deltaTime \* speed;

15Unity. *Unity Script Reference, MonoBehaviour*. 2018. URL: https : / / docs . unity3d . com / ScriptReference / MonoBehaviour.html.

16Unity. *Unity Manual, Input Manager*. 2018. URL: https:// docs. unity3d. com/ Manual/ class- InputManager. html.

6 float z = Input.GetAxis ("Vertical") \* Time.deltaTime \* speed;

7 (...)

8 }

Listing 2.1: Abfrage der X- und Z-Achsen im Code

### Physik

* + - 1. **Rigidbody**

Jedes GameObject, auf welches sich physikalische Kräfte wie Gravitation auswirken, besitzt einen Rigidbody (=starrer Körper)17.

Abbildung 2.6: Rigidbodykomponente

Hier wird die Masse des Players auf 1 festgelegt. Dies wird unter Anderem beim Berechnen des Springens verwendet.

* + - 1. **Kollisionen**

Die Kollisionserkennung ist einer der wichtigsten Bestandteile eines Spiels. Sie wird nicht nur für die Geltend- machung eines Treffers verwendet, sondern sie entscheidet auch, wann ein Element im Sichtfeld eines Spielers ist und wann nicht. Die Hauptkomponente für das Feststellen einer Kollision ist der Collider. Dessen Umrisse werden in der IDE definiert:

17Unity. *Unity Manual, Rigidbody Overview*. 2018. URL: https : / / docs . unity3d . com / Manual / RigidbodiesOverview.html.

Abbildung 2.7: Kollisionskomponente des Katanas

Hier ist der Collider in Grün zu sehen. Man sieht, dass er nicht die ganze Waffe abdeckt, da nur die Klinge als Trefferzone gelten soll. Um diese Kollision einem Skript zu übermitteln, gibt es zwei Methodengruppen:

**OnCollisionEnter/Stay/Exit**

wird ausgelöst, sobald ein physikalischer Treffer zwischen zwei Körpern festgestellt wird.

**OnTriggerEnter/Stay/Exit**

wird ausgelöst, wenn ein bestimmter Collider getroffen wird. Dies muss hierbei nicht zwischen zwei Körpern stattfinden, sondern kann zum Beispiel beim Eintreten in ein Sichtfeld geschehen(Abschnitt 3.2.5.3).

# Prozess

## Allgemein

### Arbeitsdokumentation

Ich hatte jeden Tag ein Heft in meiner Schultasche mit mir getragen, in welchem ich auftauchende Ideen so- fort notierte. Wenn ich dann wieder an meinem Computer sass, wurden diese Ideen nochmals durchgeschaut und entweder verworfen oder implementiert. Ich sorgte dafür, dass ich jede Woche mindestens einmal dar- an arbeitete und auch für längere Zeit dran blieb. In einer Textdatei journal/quellen.txt führte ich ein Arbeitsjournal in welchem ich festhielt, wann ich woran gearbeitet habe. Über die Versionskontrolle war ich abgesichert für den Fall, dass Daten verloren oder kaputt gehen würden. Auch wenn ich beim Entwickeln fest- stellte, dass die Version von vor zwei Tagen besser funktionierte als die jetzige, konnte ich auf diese zurück wechseln. Ebenso wurde der Entwicklungsfortschritt automatisch erfasst.

### Inspirationsquellen

Was das Erfinden der Objekte und Geschichte angeht, diente mir vor allem meine rege Fantasie als Inspiration. Doch ab und an half auch diese nichts, gerade wenn es um komplizierte Funktionalitäten ging (z.B. Sicht- Vektoren für Interaktionen und der Sprung des Spielers).

In diesen Fällen konsultierte ich zunächst eine Webseite namens Udemy (www.udemy.com). Diese bietet eine grosse Spannweite an Kursen. Mein Vater hatte mir dort vor ein paar Monaten einen Programmierkurs für C# geschenkt, welcher sich nun als nützlich erweisen sollte. Dieser Kurs war eine auf sich aufbauende Videoserie, in welcher ein 2D Spiel von A-Z entwickelt wurde. Da mein Spiel jedoch 3D sein sollte und der Udemy Kurs beim besten Willen nicht alle Thematiken abdecken konnte, blieb mir immer noch die öffentliche Tutorialwebseite und das Forum von Unity.1

### Vorgehen

Nachdem ich die grundlegenden Objekte wie den Spieler und die einfachen Funktionen wie Laufen erstellt hatte, ging es daran, das Spiel zu erweitern. Ich bin nicht einer der wenigen, die sich ein Skript im Kopf überle- gen und dies dann fehlerfrei beim ersten Versuch zum laufen kriegen können, noch nicht. Wenn es also an die Umsetzung einer Idee ging, schrieb ich den Code in ein schon existierendes Skript. Falls es nicht funktionierte arbeitete ich daran bis es geklappt hat oder ich wählte einen neuen Ansatz.

Sobald funktional alles glatt lief, musste der Code aufgeräumt werden. Dabei wurden die noch vorhandenen Spuren der nicht erfolgreichen Versuche entfernt, der gewünschte Code allenfalls bereinigt und vereinfacht. Das Aufräumen oder Auslagern passierte entweder dadurch, dass ich den Code manuell geändert bzw. den getesteten Code in ein neues Skript hinein kopierte, oder indem ich den Code mit Hilfe von MonoDevelop umgebaut habe.

### Code-Umbau

Das Umbauen von Code ist vor allem in der objektorientierten Programmierung sehr wichtig, was dem Vorgang seinen eigenen Namen einbrachte: Refactoring. Ins Deutsche wurde es als „Refaktorisieren“ übersetzt, was eigentlich falsch ist, da das Umbauen von Code wenig mit dem Faktorisieren der Mathematik zu tun hat. Treffender wäre der Begriff „Restrukturierung“.

1Unity. *Unity 3D Tutorial*. 2018. URL: https://unity3d.com/de/learn/tutorials.

9

Umbenennen

Wenn ich im Schwung vorwärts arbeite, leidet meine Rechtschreibung. Ebenfalls neige ich dazu, während des Kommentierens in andere Sprachen zu fallen. Ich habe auch beim Kodieren manchmal die Gross- und Klein- schreibung nicht eingehalten. Standard ist aber, dass alle Namen von Methoden gross anzufangen, diejenigen von Variablen und Eigenschaften hingegen klein.

Beim Umbenennen wird Folgendes gemacht: Wenn ich nun ein rein textbasiertes Suchen und Ersetzen ma- chen würde, könnte es passieren, dass an gewissen Stellen fälschlicherweise ebenfalls ersetzt wird und dann könnte nicht mehr kompiliert werden. Deswegen hat MonoDevelop einen eigenen Mechanismus, welcher NUR genau die angegebene Variable/Methode/Eigenschaft verändert, und diese überall.

Das einzige, was manuell angepasst werden muss, ist der Dateiname des Skripts beim Umbenennen der Hauptklasse (Hauptklasse und Dateiname müssen übereinstimmen) und deren Referenzen in der Unity Umge- bung.

Vereinfachen

Beim Entwickeln kann es passieren, dass mehrmals hintereinander die gleichen Aufrufs-Ketten stehen. Aus Gründen der Lesbarkeit und Geschwindigkeit ist es sinnvoll, diese in eine Variabel zwischenzuspeichern. Dabei unterstützt einen die Monodevelop IDE:

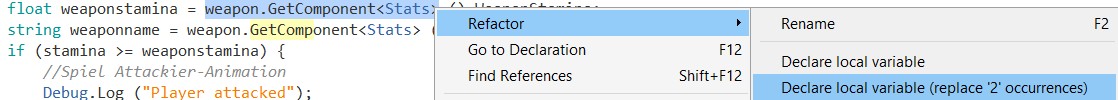


Abbildung 3.1: Vereinfachen des GetComponent<> Aufrufs Das Resultat schaut folgendermassen aus:

1 var stats = weapon.GetComponent<Stats>();

2 float weaponstamina = stats.WeaponStamina;

3 string weaponname = stats.WeaponName;

Listing 3.1: Code nach Vereinfachung

Aufteilen

Wenn es grosse Code-Teile waren, die neu entstanden, dann sollten die entsprechenden Teile eigenständige Klassen werden. Denn ein einziges riesiges Skript, welches das ganze Spiel steuert ist, weder besonders gut für das rasche Arbeiten, noch wird es von anderen Programmierern/Programmiererinnen gerne gesehen, da die Übersichtlichkeit verloren geht. So wurde während dem Entwickeln des Waffenwechsels im Playerskript der Code immer grösser. Nach einer Zeit ergab es Sinn, diesen Teil des Codes in ein eigenes Skript auszulagern.

## Problemstellungen und deren Umsetzung

In den nächsten Abschnitten werden Bereiche des Spiels und die Funktionen, welche dabei von den jeweiligen Klassen umgesetzt werden, aufgeführt. Skripte werden jeweils nach dem Namen der Hauptklasse bezeichnet (mit der Endung .cs).

### Spielführung

Damit ein Spiel wirklich gut benutzbar wird, benötigt es gewisse Funktionen, die den Spieler unterstützen und durch das Spiel begleiten:

* + - * Um nicht jedes Mal neu anfangen zu müssen, gibt es ein Menü mit der Möglichkeit des Unterbruchs und Wiederaufnahme des Spiels, verbunden mit einer Speicherfunktion des Spielstands.
      * Das Head-Up-Display (HUD) hat die Aufgabe, dem Spieler alle nötigen Informationen über den Zustand seiner Spielfigur zu liefern.
      1. **Menü (Klasse)**

Das Menü beinhaltet vier Elemente: Einen Canvas, also eine Art „Plache“, die für uns hier als Hintergrundab- dunklung fungiert. Dann gibt es 3 Buttons:

Abbildung 3.2: Menü

New Game

Dies setzt alle gespeicherten Spieldaten zurück und startet das Spiel neu.

1 // Startet ein neues Spiel

2 public void OnButtonNewPressed()

3 {

4 player.gameData.InitNewGame();

5 }

Listing 3.2: New Game

Save Game

Dieser Knopf bewirkt, dass bestimmte Daten wie die X-, Y-, Z-Position des Spielers, Gegners sowie wichti- ge Spieldaten wie Gesundheitszustand etc. in eine Datei geschrieben werden. Bei einem Neustart des Spiels werden diese Angaben wieder eingelesen und übernommen.

1 // Speicherfunktion

2 public void OnButtonSavePressed()

3 {

4 Debug.Log("Speichern");

5 player.gameData.SaveGame();

6 }

Listing 3.3: Save Game

End Game

Der Knopf führt einen Unity Befehl namens Application.quit aus, welcher das ganze Programm beendet.

1 // Beendet das Spiel

2 public void OnButtonEndPressed()

3 {

4 Debug.Log("Spiel Beendet");

5 Application.Quit();

6 }

Listing 3.4: End Game

* + - 1. **GameData (Klasse)**

Um die Daten des aktuellen Spiels zu speichern oder diejenigen des letzten Spiels zu laden, schuf ich die Klasse GameData. Mit Hilfe der PlayerPref Klasse schreibt oder liest sie übergebene Daten in oder aus eine/r Datei. Im Folgenden zeige ich die fürs Speichern verwendete Reihe von Aufrufen.

1 public void SaveGame()

2 {

3 PlayerPrefs.SetInt ("GameState", 0);

4 player.SaveState (this);

5 foreach (GameObject go in enemies) {

6 NPC npc = go.GetComponent<NPC>();

7 npc.SaveState(this);

8 }

9 theThirdKind.SaveState(this, player);

10 }

Listing 3.5: Methode zur Speicherung des Spielstandes in GameData

1 public void SaveState (GameData gameData)

2 {

3 gameData.SaveTransform("player", transform);

4 gameData.SaveFloat("playerhealth", health);

5 gameData.SaveFloat("playerstamina", stamina);

6 gameData.SaveInt("playerrp", regenerationPoints);

7 }

Listing 3.6: Methode SaveState in Player

1 public void SaveTransform (string scope, Transform transform)

2 {

3 Vector3 position = transform.position;

4

5 PlayerPrefs.SetFloat (scope + "X", position.x);

6 PlayerPrefs.SetFloat (scope + "Y", position.y);

7 PlayerPrefs.SetFloat (scope + "Z", position.z);

8 }

Listing 3.7: Methode SaveTransform in GameData

* + - 1. **Head-Up-Display (Klasse HUD)**

Ein Head-up-Display ist eine Anzeigefläche, die sich nicht aus dem Sichtfeld bewegt, auch wenn der Benutzer seinen Kopf neigt oder schwenkt. Somit sind die Informationen in jeder Situation ablesbar.

Abbildung 3.3: HUD

Aufgabenbereich (1)

Der Aufgabenbereich befindet sich oben rechts. Darin wird die jeweils aktuelle Aufgabe angezeigt.

Zustandsdisplay (2)

Das Zustandsdisplay befindet sich oben links im HUD. Es besteht aus zwei Schiebereglern Slider, einem für die Gesundheit player.health und einem für die Ausdauer player.stamina des Spielers. Ein Schiebereg- ler ist ein 2D Balken, der einen Wert verkörpert. Man kann ihm in der Entwicklungsumgebung einen minimalen und einen maximalen Wert geben. Hier ist das bei beiden 0-100:

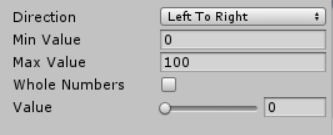


Abbildung 3.4: Schieberegler Konfiguration

Abgefüllt werden diese Balken durch das HUD-Skript, welches bei jedem Update Aufruf die aktuellen Werte aus dem Player ausliest und an die Schieberegler weitergibt.

1 public class HUD : MonoBehaviour

2 {

3 (...)

4 // Gesundheit Schieberegler

5 private Slider healthSlider;

6 // Ausdauer Schieberegler

7 private Slider staminaSlider;

8 (...)

9 // Statusinformationen aus Spieler übernehmen und anzeigen

10 void Update()

11 {

12 (...)

|  |  |  |  |
| --- | --- | --- | --- |
| 13 |  |  | healthSlider.value = player.health; |
| 14 |  |  | staminaSlider.value = player.stamina; |
| 15 |  |  | (...) |
| 16 |  | } |  |
| 17 | } |  |  |

Listing 3.8: Schieberegler aktualisieren

Kommunikationsbereich (3)

Der Kommunikationsbereich befindet sich im unteren Drittel des Bildschirmes. Normalerweise ist er deaktiviert und damit unsichtbar. Aktiviert wird er dann, wenn eine Methode, z.B. ein Interact, das Attribut lastTalk des Players neu setzt.

1 void Interact(Player p)

2 {

3 (...)

4 p.lastTalk = "You must pick the two flowers ...";

5 (...)

6 }

Listing 3.9: Kommunikation setzen

Dann erscheint eine Sprechblase mit dem entsprechenden Text. Im folgenden Codeausschnitt wird gezeigt, wie das Ein- und Ausschalten der Sprechblase und das Aktualisieren des beinhalteten Textes funktioniert. Dies geschieht in der periodisch aufgerufenen Update Methode. Zum besseren Verständnis: Time.time liefert die Zeit in Sekunden seit Spielstart.

1 public class HUD : MonoBehaviour

2 {

3 // Die Spielerinstanz, welche alle Informationen liefert

4 public Player player;

5

6 // Feld, welches die Sprechblase beinhaltet

7 private GameObject talking;

8 // Textobjekt, welches den gesprochenem Text anzeigt

9 private Text talkingText;

10 // Zeitpunkt, als die letzte Blase angezeigt wurde, 0 wenn keine

11 // angezeigt wird

12 private int speechDisplayedTime = 0;

13 // Darstellungszeit der Sprechblase

14 private const int speechDisplayDuration = 15;

15

16 (...)

17

18 // Update wird pro frame einmal aufgerufen

19 // Statusinformationen aus Spieler übernehmen und anzeigen

20 void Update()

21 {

22 (...)

23 string lastTalk = player.lastTalk;

24 // wenn mit dem Spieler seit letztem Mal gesprochen wurde

25 if (lastTalk.Length > 0) {

26 // anzeigen der Sprechblase

27 player.lastTalk = "";

28 talking.SetActive(true);//hier wird die Sprechblase aktiviert

29 talkingText.text = lastTalk;

30 // Zeitpunkt des Anzeigens merken

31 speechDisplayedTime = (int)Time.time;

32 } else {

33 // kein neuer Text, überprüfe ob die Sprechblase

34 // wieder versteckt werden soll

35 if (speechDisplayedTime > 0) {

36 if ((int)Time.time - speechDisplayedTime >

37 speechDisplayDuration) {

38 speechDisplayedTime = 0;

39 talking.SetActive(false);//hier wird die Sprechblase deaktiviert

40 talkingText.text = "";

41 }

}

|  |  |  |  |
| --- | --- | --- | --- |
| 42 |  |  |  |
| 43 |  |  | } |
| 44 |  | } |  |
| 45 | } |  |  |

Listing 3.10: Sprechblase ein- und ausblenden

### Spieler

Das 3D Modell des Spielers besteht ausschliesslich aus Würfeln und Quadern:

Abbildung 3.5: Player 3D Modell Die Funktionalität für den Spieler ist auf verschiedene Klassen verteilt:

* + - 1. **Player (Klasse)**

In dieser Klasse finden sich die Funktionen für den Zustand des Spielers (z.B. Gesundheit und Ausdauer), die Fortbewegung und die Interaktionen mit anderen Objekten.

Gesundheitszustand

Der Gesundheitszustand wird im HUD angezeigt und kann über einen Methodenaufruf z.B. von Waffen verän- dert werden. Der Player wie auch der NPC haben beide in ihrem eigenen Script das Setup ihrer Gesundheit.

1 // Gesundheit

2 public float health = 100f;

Listing 3.11: Gesundheit als öffentliche Variable

Um die Gesundheit zu verändern benutze ich eine Methode, in welcher der Wert, um den die Gesundheit geändert werden soll, in den Aufruf mitgegeben wird. Dieser kann sowohl positiv als auch negativ sein. Er wird dann zur vorherigen Gesundheit addiert.

Dafür, dass die Gesundheit nicht über 100 geht und falls sie unter 0 geht, man auch tatsächlich stirbt, sorgen die if-Statements. Für den NPC sieht es so gut wie gleich aus:

1 public void ChangeHealth(float change)

2 {

3 health += change;

4

5 if (health > 100.0f)

6 health = 100.0f;

7 else if (health < 0.0f) {

8 Debug.Log("YOU ARE DEAD");

9 model.SetActive(false);//lässt das gameObject verschwinden

10 }

11 }

Wichtig ist, dass alle Veränderungen über diese Methode gemacht werden, damit es nur einen Ort gibt, an welchem Entscheide über Tod gefällt werden können.

Laufen

Der hier gezeigte Codeausschnitt beinhaltet das Abfragen der Bewegungssteuerung und deren Umsetzung. Danach wird die Laufanimation (siehe Abschnitt 3.2.3.1) initialisiert.

1 // x und z Koordinaten Bewegung

2 float x = Input.GetAxis ("Horizontal") \* Time.deltaTime \* speed;

3 float z = Input.GetAxis ("Vertical") \* Time.deltaTime \* speed;

4

5 // Animationen

6 anim.SetFloat ("forward", z \* 3);

7 anim.SetBool ("Walking", true);

8 (...)

9 transform.Translate (x, 0, z);

Listing 3.12: Laufen

Springen

Um zu springen muss der Spieler den Boden berühren. Ob sich etwas unter ihm befindet, wird mit einem Raycast überprüft, einem virtuellen Strahl. Wenn dieser Raycast etwas trifft, ist Springen erlaubt. Der Sprung erfolgt, indem ich dem Spieler eine Anfangsgeschwindigkeit in Y-Richtung (=Kraftstoss aus der Physik) gebe. Den Rest übernimmt Unitys Physik-Komponente.

1 // raycast für "isgrounded"

2 RaycastHit hit;

3 (...)

4 if (Input.GetAxis ("Jump") > 0f) {

5 // isgrounded: Vektor Richtung Boden mit Länge 1.

6 // Wenn er etwas trifft, ist isgrounded true (was bedeutet,

7 // dass springen möglich ist). Wenn nicht, dann nicht.

8

wn,

|  |  |  |  |
| --- | --- | --- | --- |
| 9 |  | if | (Physics.Raycast (transform.position, Vector3.do |
| 10 |  |  | Debug.DrawLine (transform.position, hit.point); |
| 11 |  |  | print (hit.distance); |
| 12 |  |  | Vector3 power = rigid.velocity; |
| 13 |  |  | power.y = 5f; |
| 14 |  |  | rigid.velocity = power; |
| 15 |  | } |  |
| 16 | } |  |  |

out hit, 1)) {

Listing 3.13: Springen

Kämpfen

Beim Angriff wird im Player zunächst überprüft, ob überhaupt noch genügend Kraft für den Einsatz der individuellen Waffe vorhanden ist. Wenn ja, dann wird sie verwendet und die Ausdauer des Players um den zugehörigen Betrag vermindert.

1 // Angriff

2 if (Input.GetButtonDown ("Fire1")) {

3 // hole und vergleiche Waffenwerte

4 GameObject weapon = GetComponent<WeaponManager>().getActiveWeapon();

5 var stats = weapon.GetComponent<Stats>();

6 float weaponstamina = stats.WeaponStamina;

7 string weaponname = stats.WeaponName;

8 if (stamina >= weaponstamina) {

9 // Spiel Attackier-Animation

10 Debug.Log("Player attacked");

11 if (weaponname == "Katana" || weaponname == "Bo")

12 anim.Play ("Katana 0");

13 else if (weaponname == "Hands")

14 anim.Play ("Katana 0");

15 ChangeStamina (-weaponstamina);

16 }

17 }

Listing 3.14: Angriff

Die weiteren Abläufe beim Kampf des Spielers werden im Abschnitt 3.2.4 beschrieben.

* + - 1. **Drehbewegung des Spielers (Klasse MouseLookAtIt)**

Im Gegensatz zu allen anderen Code-Ausschnitten, die mit einer Art Bewegung zu tun haben, geht es hier nicht um eine Verschiebung, sondern um eine Drehung. Dieser Code sorgt dafür, dass sich für eine Links/Rechts Drehung die Spielfigur selbst bewegt, für eine Auf/Ab Bewegung hingegen nur die Kamera (Kopf).

1 // Ermittle die Maus-Position

2 rotationY += Input.GetAxis ("Mouse X") \* sensitivityY;

3 rotationX += Input.GetAxis ("Mouse Y") \* sensitivityX;

4 rotationX = Mathf.Clamp (rotationX, minimumX, maximumX);

5 rotationY = Mathf.Clamp (rotationY, minimumY, maximumY);

6

7 // Bewege das Gameobject für die Links-/Rechts-Bewegung

8 transform.localEulerAngles = new Vector3 (0, rotationY, 0);

9 // und die Kamera für die Auf-/Ab-Bewegung

10 cam.transform.localEulerAngles = new Vector3 (-rotationX, 0, 0);

Listing 3.15: Drehbewegungen

### Animationen

In Unity Animationen zu erstellen braucht etwas Geschick und viel Geduld. Jede der drei Spezies hat drei Animationen.

* + - 1. **Die verschiedenen Animationen**

Lauf-Animation

Die Lauf-Animation besteht daraus, dass ein Fuss gehoben und nach vorne bewegt, dann gesenkt wieder nach hinten bewegt wird. Diese Animation wird versetzt auf den zweiten Fuss kopiert, so dass immer, wenn der eine Fuss hinten ist, der andere vorne ist und umgekehrt. Die grösste Schwierigkeit bei dieser Animation war es, die beiden Füsse auf einander abzustimmen. Dazu kam, dass die Fortbewegungsgeschwindigkeit des Objekts ungefähr mit der Fussbewegung der Animation übereinstimmen sollte. Die Figur TheThirdKind hat zwar eine Laufanimation, benutzt sie jedoch aktuell nicht, da sie sich noch nicht bewegt.

Untätig-Animation

Die Untätig-Animation (oder auch Idle-Animation) war die einfachste Animation zum Erstellen, da sie rein aus einem synchronen heben und senken der Hände besteht.

Angriffs-Animation

Diese Animation war die schwierigste, da sie neben einer Positionsveränderung auch eine Drehung verlangte. Das Problem war, dass bei einer Drehung der Hand diese aus ihrer Form fiel und die in der Hand gehaltene Waffe ebenso falsch skaliert wurde. Dieses Problem behob ich mit Feintuning an den Kurven.

* + - 1. **Erstellen einer Animation**

Um eine Animation neu hinzuzufügen gibt man ihr zuerst einen Namen. Danach öffnet sich ein leeres Fenster des Unity-Animators2 . Dort hinein kann man GameObjekte ziehen und wählen, ob man Rotation, Proportion oder Position verändern will (es gibt noch viele andere Optionen, die man verändern kann, aber diese habe ich nicht benutzt).

Abbildung 3.6: Animationskurven

In diesem Beispiel handelt es sich um eine Laufanimation, welche nur die Füsse kontrolliert. Links sieht man die einzelnen Elemente und deren X-, Y- und Z-Koordinaten mit dazugehöriger Farbe. Rechts sind die Kurven, welche beschreiben, wie sich diese Koordinaten im Ablauf der Animation verändern (zum Beispiel hier sichtbar: Bewegung nur in Y und Z Koordinaten, die X-Koordinate bleibt unverändert (rot)). Man sieht hier besonders, wie sich die Füsse nicht ruckartig, sondern fliessend und einander entgegengesetzt bewegen.

2Unity. *Unity Manual, Animation Window Guide*. 2018. URL: https : / / docs . unity3d . com / Manual / AnimationEditorGuide.html.

* + - 1. **Animator**

Der Animator ist das Element, welches den Wechsel zwischen verschiedenen Animationszuständen (Idle, Wal- king etc.) steuert. Zeiger können von einem Zustand auf einen anderen verweisen. Diese Übergänge kann man wiederum an Bedingungen koppeln.

Abbildung 3.7: Animator mit Zuständen und deren Übergängen

### Waffen

Es gibt drei Waffen: Das Katana, das Bo, und die Fäuste. Jede Waffe hat verschiedene Werte was Schaden, Ausdauerkosten und Reichweite angeht.

Abbildung 3.8: Katana

Abbildung 3.9: Bo

* + - 1. **Stats (Klasse)**

Diese Klasse führt die Werte der Waffen. Die jeweiligen Werte werden in der Programmierumgebung deklariert. Diese verwenden dann die anderen beteiligten Klassen.

1 public string WeaponName = "Katana";

2 public float WeaponDamage = 35;

3 public float WeaponDefense = 15;

4 public float WeaponRange = 3;

5 public float WeaponSpeed = 3;

6 public float WeaponStamina = 10;

Abbildung 3.10: Bo Stats Werte

* + - 1. **WeaponHit (Klasse)**

Jede Waffe hat einen Collider (siehe Abschnitt 2.2.3.2). Dieses Element löst ein OnTriggerEnter Ereignis aus, sobald es mit einem anderen Collider in Berührung kommt. Das hier verwendete Script ermittelt das Kennzeichen des getroffenen Objekts und fügt Schaden zu, entweder dem Gegner oder dem Spieler selbst, je nachdem ob es sich dabei um ein Element mit dem Tag „Enemy“ oder mit dem Namen „Player“ handelt. Der Wert des Schadens berechnet sich aus der Waffenwirkung, welche es aus den Stats der jeweiligen Waffe ausliest. Dieser wird dann dem getroffenen Objekt zugefügt.

1 damage = Self.GetComponent<Stats>().WeaponDamage;

2

3 void OnTriggerEnter(Collider col)

4 {

5 Debug.Log("Hit: " + col.name + " Weapon: " + weaponname);

6 if (col.tag == "Enemy") {

7 col.gameObject.GetComponentInParent<NPC>().ChangeHealth(-damage);

8 Debug.Log(damage + " Damage applied to " + col.name);

9 } else if (col.name == "Player") {

10 col.gameObject.GetComponentInParent<Player>().ChangeHealth(-damage);

11 Debug.Log(damage + " Damage applied to " + col.name);

12 }

13 }

Listing 3.16: Waffentreffer

* + - 1. **WeaponManager (Klasse)**

Der WeaponManager verwaltet die verschiedenen Waffen und wird verwendet, um die aktive Waffe zu wech- seln3.

1 void Update()

2 {

3 // Wechseln der Waffe

4 if (isplayer == true) {

5 if (Input.GetAxis("1") > 0f) {

6 SetActiveWeapon(Katana);

7 equipped = "Katana";

8 }

9

10 if (Input.GetAxis("2") > 0f) {

) {

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | SetActiveWeapon(Bo); equipped = "Bo"; | | | |
| 12 |
| 13 |  |  | } | |
| 14 |  |  |  | |
| 15 |  |  | if | (Input.GetAxis("3") > 0f |
| 16 |  |  |  | SetActiveWeapon(Hands); |
| 17 |  |  |  | equipped = "Hands"; |
| 18 |  |  | } |  |
| 19 |  | } |  |  |
| 20 | } |  |  |  |

Listing 3.17: Wechsel der Waffen je nach Taste

1 private void SetActiveWeapon (GameObject activeWeapon)

2 {

3 for (int j = 0; j < weaponList.Count; j++) {

4 if (weaponList [j] == activeWeapon) {

5 for (int i = 0; i < enabledWeaponList.Count; i++) {

6 if (enabledWeaponList [i] == activeWeapon) {

7 weaponList [j].SetActive (true);

8 weaponInHand = activeWeapon;

9 }

10 }

11 } else {

12 weaponList [j].SetActive (false);

13 }

14 }

3Unity Forum. *Help with multiple weapons switching*. 2018. URL: https : / / forum . unity . com / threads / solved - missing-prefabs-with-freshly-downloaded-project-windows.411721/.

15 }

Listing 3.18: Aktivieren einer Waffe

Die Methode SetActiveWeapon hat die Aufgabe die activeWeapon zu aktivieren und alle anderen zu deaktivieren, sonst würden sie gleichzeitig erscheinen.

### Figuren

* + - 1. **InteractibleObject (Klasse)**

Alle GameObjekte, mit denen interagiert werden kann, sind von dieser Klasse oder erben von dieser. Die Funktion Interact sendet einen Strahl vom Spieler aus. In der Entwicklungsumgebung kann dies während des Spiels dargestellt werden (1).

Abbildung 3.11: Vektorabfrage bei Interaktion mit TheThirdKind

Wenn dieser Strahl etwas trifft, wird überprüft, ob das getroffene GameObject den Tag Interactive hat. Falls ja, wird eine Nachricht Interact an das getroffene Objekt geschickt. Falls der Tag nicht vorhanden ist, passiert nichts.

1 // Interaktion (Standard button "E")

2 if (Input.GetKeyDown("e")) {

3 Vector3 forward = transform.TransformDirection(Vector3.forward);

4

5 if (Physics.Raycast(transform.position, forward, out interactHit, 10)) {

6

7 Debug.DrawLine(transform.position, interactHit.point);

8 if (interactHit.collider.gameObject.tag == "Interactive") {

9 // sende eine Nachricht

10 interactHit.collider.gameObject.SendMessage("Interact", (Player)this);

11 // aktualisiere Aufgabe - wenn vorhanden

12 quests.Interacted(interactHit.collider.gameObject);

13 }

14 }

15 }

Listing 3.19: Auslösen der Interaktion

Im getroffenen Objekt wird damit, falls vorhanden, die Methode Interact aufgerufen. Diese kann je nach Art des Objekts eine andere Handlung bewirken (sogenannte Polymorphie im objektorientierten Programmie- ren). So wird z.B. gesammelt oder gesprochen:

Sammeln

Wie jedes gute RPG braucht auch das meinige eine klassische Sammel-Aufgabe. In diesem Spiel geht es zu- nächst um das Sammeln von Blumen. Sobald der Spieler mit den Blumen interagiert, verschwinden sie: sie wurden gepflückt. Dies passiert, weil die oben beschriebene Nachricht Interact bei dem Objekt Blume ein Skript ausführt, welches das Objekt deaktiviert.

1 void Interact(Player p)

2 {

3 gameObject.SetActive(false);

4 }

Listing 3.20: Standardimplementation von Interact

Reden

Es gibt nur ein einziges GameObject das mit dem Player redet, und das ist TheThirdKind. Diese Klasse ist eine Unterklasse von InteractibleObject, und kann deshalb beim Aufruf von Interact in einer anderen Form reagieren.

Abbildung 3.12: Interact in der Klasse InteractibleObject und die abgeleitete Klasse TheThirdKind

Das Gesprochene erscheint in einer Sprechblase. Um dies auszulösen, muss lastTalk des Spielers geändert werden. Dies passiert, wenn der Spieler mit dem TheThirdKind Objekt interagiert

* + - 1. **Zufallsbasiertes Generieren von Objekten**

Mit der Methode Instantiate können neue Objekte einer bestimmten Klasse an einem gewünschten Ort neu erstellt werden. Ich verwende dies für die Erstellung von Blumen an zufälligen Orten im Bereich des Landes von TheThirdKind.

1 void Start()

2 {

3 (...)

4 for (int i = 1; i < 21; i++) {

5 GameObject go = (GameObject)Instantiate(flower, new Vector3 (Random.Range(370,650), 0, Random.Range(120,390)),

6 Quaternion.identity);

7 go.name = "FlowerRed" + i;

8 Debug.Log(go.name + " at " + go.transform.position.x + " " + go.transform.position.y + " " + go.transform.position.z);

|  |  |  |
| --- | --- | --- |
| 9 |  |  |
| 10 |  | } |
| 11 |  | (...) |
| 12 | } |  |

Listing 3.21: Generieren von Blumen beim Start (Player.cs)

* + - 1. **Autonomer Gegner NPC (Klasse)**

NPC bedeutet Non-Player-Character. Der Gegner war in der Anfangsphase der Programmierung dieses Spie- les der einzige Nicht-Spieler-Character, deswegen blieb der Klassenname. Der NPC ist der Antagonist meines Spiels. Er ist neben dem Player selber das komplizierteste Objekt. Das 3D Modell des NPC besteht nur aus Sphären.

Abbildung 3.13: NPC 3D Modell

Aktivieren

Damit der NPC auf den Spieler aufmerksam wird, muss er ihn zuerst sehen. Dafür hat der NPC einen Collider, welcher sein Sichtfeld darstellt:

Abbildung 3.14: Sichtfeld des NPCs

Sobald der Spieler diesen Collider einmal ausgelöst hat, wird der Bewegungscode aktiviert.

1 public bool hasseenplayer = false;

2 private void Start()

3 {

4 player = GameObject.FindGameObjectWithTag("Player");

5 }

6

7 // Den Spieler entdecken

8 void OnTriggerEnter(Collider fov)

9 {

10 if (fov.name == "Player") {

11 Debug.Log("has seen");

12 hasseenplayer = true;

13 }

14 }

Bewegen

Der NPC richtet sich nach dem Spieler aus und geht so lange auf ihn zu, bis die X und Y Koordinaten auf eine Differenz von 1 genau übereinstimmen.4 Dann spielt der NPC seine Angriffs-Animation ab.

1 if (hasseenplayer == true) {

2 // wenn Spieler in Reichweite, angreifen (fighting = true )

3 Vector3 forward = transform.TransformDirection(Vector3.forward);

4

5

6 if (Physics.Raycast(transform.position, forward, out hit, 10)) {

7

8 Debug.DrawLine(transform.position, hit.point);

"Player") {

|  |  |  |
| --- | --- | --- |
| 9 | if | (hit.collider.gameObject.tag == |
| 10 |  | fighting = true; |
| 11 |  | Debug.Log("NPC is attacking"); |
| 12 | } else { | |
| 13 |  | fighting = false; |
| 14 |  | anim.ResetTrigger("Attack"); |
| 15 | } |  |
| 16 |  |  |
| 17 | if | (fighting == true) { |
| 18 |  | // Animation Angriff starten |
| 19 |  | anim.SetTrigger("Attack"); |
| 20 |  |  |

21 } else {

22 anim.SetBool("iswalking", true);

23

24 if (player.transform.position.x > (transform.position.x - 1)) {

25 // Gehe nach rechts

26 transform.position += new Vector3(Speed \* Time.deltaTime, 0, 0);

27

28 } else {

29 // Gehe nach links

30 transform.position -= new Vector3(Speed \* Time.deltaTime, 0, 0);

31 }

32 // Gehe zu Player’s Z-Koordinate

33 if (player.transform.position.z > transform.position.z) {

34 // Gehe hoch

35 transform.position += new Vector3(0, 0, Speed \* Time.deltaTime);

36 } else {

37 // Gehe runter

38 transform.position -= new Vector3(0, 0, Speed \* Time.deltaTime);

39 }

40 //https://docs.unity3d.com/ScriptReference/Transform.LookAt.html

4Joseph Manley. *How do I make an NPC move in Unity*. 2016. URL: https://[www.quora.com/How-](http://www.quora.com/How-) do- I- make- an- NPC-move-in-Unity.

|  |  |  |  |
| --- | --- | --- | --- |
| 41 |  |  | transform.LookAt(target); |
| 42 |  | } | |
| 43 |  | } else { | |
| 44 |  |  | anim.SetBool("iswalking", false); |
| 45 |  | } |  |
| 46 | } |  |  |

* + - 1. **TheThirdKind (Klasse)**

TheThirdKind ist ein Schlüsselobjekt des Spiels. Er ist ein Weiser, der den Spieler leitet. Zu ihm muss man zu Beginn gehen, um die Blumen-Aufgabe zu erhalten. Er sollte der aus vielen RPGs bekannten Auftraggeber sein, der einen quer durch die Welt schickt um ein paar Rohstoffe zu sammeln. Sein 3D Modell besteht nur aus Kegeln.

Abbildung 3.15: TheThirdKind 3D Modell

### Aufgaben

* + - 1. **Quest (Klasse)**

Eine Quest ist eine einzelne Aufgabe. Sie hat einen beschreibenden Text und ein zugehöriges GameObject, das über seinen Namen festgelegt wird.

1 public class Quest

2 {

3 // Eine Aufgabe mit dem namen des Zielobjekts und einer Aufgabenbeschreibung.

4 public Quest(string nameString, string taskString)

5 {

6 name = nameString;

7 task = taskString;

8 done = false;

9 }

10

11 virtual public bool IsDone()

12 {

13 return done;

14 }

15

16 public bool Done()

17 {

18 done = true;

19 return IsDone();

20 }

Listing 3.22: Quest Constructor

Bei der Interaktion wird die Done Methode aufgerufen, die IsDone Methode gibt an, ob die Aufgabe wirk- lich erfüllt wurde. In der Subklasse FreedomQuest ist das z.B. erst der Fall, wenn alle Gegner erledigt sind:

1 public class FreedomQuest : Quest

2 {

3 public static int npcsAlive;

4

5 public FreedomQuest(string taskString) : base("A1", taskString)

6 {

7 }

8

9 public override bool IsDone()

10 {

11 return npcsAlive < 1;

12 }

13

14 public override string Task()

15 {

16 return "Fight the remaining\n" + npcsAlive + " red figures";

17 }

18 }

Listing 3.23: FreedomQuest Subklasse

* + - 1. **Quests (Klasse)**

Diese Klasse führt die Liste aller Aufgaben, ob erledigt oder nicht. Sie aktualisiert den HUD mit der Beschrei- bung der aktuellen Aufgabe und gibt bei Interaktion mit einem GameObject an, ob für dieses eine noch offene Quest vorhanden istGetActiveQuest. Aus Gründen der Einfachheit ist sie ein Attribut der Player Klasse, deswegen kommunizieren die GameObjekte über die entsprechenden Methoden von Player:

1 // Fügt der Liste eine neue Aufgabe hinzu

2 public void AddQuest(Quest q)

3 {

4 quests.AddQuest(q);

5 }

6

7 // Sucht nach einer Aufgabe per Name

8 public Quest GetQuest(string name)

9 {

10 return quests.GetQuest(name);

11 }

12

13 // Sucht nach einer aktiven Aufgabe per Name

14 public Quest GetActiveQuest(string name)

15 {

16 return quests.GetActiveQuest(name);

17 }

Listing 3.24: Schnittstelle für Aufgaben in der Klasse Player

### Fehlersuche mit Konsolenausgabe

Bei der Fehlersuche bietet der Debugger eine Menge an Unterstützung: Werte von Variablen und Parame- tern werden angezeigt, der Weg des Aufrufs kann nachverfolgt werden etc. Doch leider wird damit das Spiel komplett unterbrochen und die Maus wird an einen anderen Ort bewegt. Dadurch entstehen Störungen, die

es unmöglich machen, gewissen Problemen wirklich auf die Schliche zu kommen. Abhilfe schafft hier die Möglichkeit mit Debug.Log direkt aus dem Code Nachrichten in die Konsole zu schreiben.

1 void OnTriggerEnter(Collider col)

2 {

3 if (col.tag == "Enemy") {

4 (...)

5 Debug.Log(damage + " Damage applied to " + col.name);

6 } else if (col.name == "Player") {

7 (...)

8 Debug.Log(damage + " Damage applied to " + col.name);

9 }

10 }

Listing 3.25: Konsoleausgaben beim Waffeneinsatz

## Versionsverwaltung

Die Versionsverwaltung funktionierte meist einwandfrei. Nur beim Klonen des Projekts auf einen Laptop stellte sich das Problem, dass das Projekt dort nicht mehr geöffnet werden konnte. Eine Suche im Forum ergab, dass Blender installiert sein muss, sonst kann das Projekt von Unity nicht geöffnet werden.5

## Verteilung und Test

### Verteilung über Github

Da ich die Versionskontrolle dieser Maturaarbeit von Anfang an mit Github durchgeführt habe, konnte sich theoretisch jeder, der meinen Github-Namen kannte, meine komplette Arbeit herunterladen.[5] Ich erzählte also in der Schulklasse von meinem Projekt und einige meiner Freunde fragten, ob sie es downloaden könnten. Leider schieden sofort einige Kandidaten aus, da sie kein Windows-Gerät besassen. Aus den restlichen drei Interessierten wurden die SimpleRPG-Beta-Tester.

### Testphase

Diese informierten mich schon früh über Abstürze, Bugs und Unschönheiten. An ihnen konnte ich ebenfalls die Effektivität meines README.MD Textes testen, welches ihnen Instruktionen für das korrekte Starten des Spiels und Informationen über die Steuerung lieferte. Oft redeten wir in den kurzen Pausen zwischen den Lektionen über mein Spiel. Das gab mir eine Plattform, wo ich meine weiterführenden Ideen präsentieren und sofort Feedback einholen konnte. Das half mir, mich auf Inhalte zu fokussieren, die von der Mehrheit auch gemocht wurden.

5Unity Forum. *Missing Prefabs with freshly downloaded project*. 2018. URL: https://forum.unity.com/threads/help- with-multiple-weapons-switching.465702/.

# Resultate

## Erkenntnisse

Meine grösste Erkenntnis war, wie schwierig das Abschätzen von Zeiten für Programmierarbeiten ist. Beim Programmieren ist es anders als beim Schreiben eines Buches wo man sich vornehmen kann, an einem Tag eine bestimmte Anzahl an Seiten zu schreiben, so dass man bis zum Ende der Frist genug Seiten hat. Der Aufwand für ein einzelnes Problem kann spontan zehnmal höher ausfallen als geplant. Anders herum habe ich auch einige Features viel schneller erledigt als gedacht. Um diesem nicht planbaren Faktor entgegen zu wirken entschied ich mich dazu, die Code-Arbeiten vorzuziehen. Dieses Vorgehen hat nun dazu geführt, dass ich einen guten Code habe, aber leider weniger Spielinhalt als ich es mir wünschte.

Beim Lösen eines Problems war einer meiner besten Ansprechpartner das Internet. Leider gibt dieses nicht nur eine Antwort auf eine Frage, sondern ich bekomme 100 Antworten, von welchen 80 ähnliche Probleme haben, aber nur 20 das gleiche. Von diesen 20 bekomme ich dann 3-6 verschiedene Lösungsvorschläge und muss ausprobieren, welcher am besten für mein Projekt geeignet ist.

## Ausbaumöglichkeiten

### Erweiterung des Spiels

Hätte ich mehr Zeit, wären sicherlich mehr Aufgaben und mehr Interaktionen meine erste Wahl. Mehr sich autonom bewegende Charaktere sind denkbar. Weitere Möglichkeiten sähe ich im Hinzufügen von zusätzlichen Waffen. Auch die Fähigkeit, die Schläge des NPCs zu blocken, hätte ich mit ein wenig mehr Zeit implementiert. Die Animation dazu habe ich schon erstellt.

### Spielerlebnis vertiefen

Um das Spielerlebnis schöner zu gestalten, nähme ich zunächst die Lichtgestaltung hinzu. Punktuelle Licht- quellen in Form von Fackeln an Häusern und glühende Feuer brächten eine ganz spezielle Atmosphäre ins Erleben. Sound-Effekte für alle Fortbewegungsarten und Kampfaktionen wären ein weiteres Element. Musik und Videosequenzen könnten die Geschichte begleiten. Da ich die vorgefertigen Assets nicht benutzen wollten, wäre das Erstellen von Bäumen ein weiter Schritt, um die Umgebung zu gestalten. Auch Gewässer, nicht zuletzt als zusätzliches Hinderniss, könnten für Abwechslung sorgen.

### Weitere Verbreitung

Mir ging die Idee durch den Kopf, das Spiel auf Indiegame-Seiten online zu stellen . Da es aber im aktuellen Stand nichts nie Dagewesenes oder Bahnbrechendes ist, habe ich mich dagegen entschieden. Für den Fall, dass ich damit Geld verdienen wollen würde, könnte man mit Google-ads auch Werbung schalten. Ich halte aber von dieser Form der Werbung nichts, ausserdem ist dies eine Maturaarbeit und kein Free-To-Play Pay-to-win Spiel (ein Mechanismus, durch den man sich mit Mikrotransaktionen einen unfairen Vorteil im Spiel erkaufen kann).

### Betriebssysteme

Mit Unity kann man so gut wie jedes Betriebssystem ansteuern. Dies führt dazu, dass ich das Spiel theoretisch auch für alle Geräte zur Verfügung stellen könnte. Dennoch habe ich mich dazu entschieden, es zunächst nur auf Windows auszulegen. Warum?

Mobiltelefone haben zu kleine Bildschirme und keine Tastatur, also fallen diese Geräte weg, wenn das Spiel nicht speziell dafür angepasst wird. Dann bleiben noch die verschiedenen Betriebssysteme für Laptops oder

30

*4 Resultate* 31

Desktop PCs. Von den dreien habe ich mich für Windows entschieden, weil ich dies selber am meisten benutze und meine „Zielgruppe“, also Gamer, fast ausschliesslich Windows verwenden, da schlicht die meisten Spiele nur für dieses OS zugänglich sind. Also entschloss ich, mich der Masse anzuschliessen und ebenfalls nur Windows zu verwenden. Dennoch: ein macOS Testlauf ist geglückt: Es ist möglich, das Projekt als macOS Version zu erstellen und unter dem System laufen zu lassen. Ein erster Durchlauf hat an sich einwandfrei funktioniert. Für ausgiebige Tests fehlte mir aber die Zeit.

## Reflexion

Nach nun mehr als einem halben Jahr Arbeit soll ich das Ergebnis bewerten.

Das Spiel, das ich in dieser Zeit erschaffen habe beinhaltet fast alle Elemente, die ich mir vorgenommen hatte. In meinen Augen ist das Spiel jedoch noch lange nicht „vollendet“, da ich immer noch mehr dazu ergänzen könnte. Doch dafür bräuchte ich noch mehr Zeit, und die Zeit habe ich nun nicht mehr. Im Grossen und Ganzen bin ich jedoch sehr zufrieden. Ich konnte ein Projekt machen, das mir Spass bereitete und bei dem ich mit Leidenschaft dabei war. Selbst, wenn mich einige Probleme manchmal zur Weissglut treiben konnten.

Unity ist ein Profi-Werkzeug, mit welchem auch teure kommerzielle Projekte durchgeführt werden. Das heisst, es ist sehr vielfältig und lässt den Benutzer somit so gut wie alles erstellen. Als „Amateur“ ist es ent- sprechend schwieriger, komplexe Effekte wie z.B. ein Feuer zu erstellen, da in diesem rund 30 Parameter für die Umsetzung des Partikelfeldes zur Verfügung stehen. In solchen Momenten war ich öfters als erwartet ge- zwungen, Anleitungen zu konsultieren, um mich nicht im Herumprobieren zu verlieren.

Die Tatsache, dass die nötige Dauer um ein Problem im Code zu lösen nicht schon vorher bekannt und für mich nur sehr ungenau abschätzbar ist, erschwerte mir das Zeitmanagement mehr, als ich es erwartet hatte. Es sind die Ungewissheiten, die mir am Meisten zu schaffen machten.

Ich habe während der Arbeit auch einige neue Dinge gelernt. Zum Beispiel kannte ich LaTeX vorher noch nicht. Dieses Werkzeug gefällt mir besonders, da die darin vorhandenen Grundfunktionen aus reinem Inhalt (alles in aus Text) gut aussehende Dokumente erstellen. Der Umgang mit den mir schon bekannten Werkzeugen lieferte zusätzliche Erfahrung, wodurch ich diese nun noch besser beherrsche.

## Download und Kontakt

Das GitHub Repository befindet sich in https://github.com/XXXX Für Rückfragen: [XXXXXXX](mailto:elias.csomor@stud.ksimlee.ch)

## Danksagung

Grosser Dank geht an Herrn XXXX, meinen Betreuer, der mich in dem Labyrinth der Möglichkeiten und Schwer- punkte begleitet hat. Ich bedanke mich auch bei meinen ehemaligen Betreuern der ersten 2 Maturaarbeits- Versuche, auch wenn es nicht geklappt hat. Dank auch an die „Nerds“ aus meiner Klasse und aus dem Mint- Labor, die mir als Spieletester geholfen oder mir die 3D Modellierung in Blender näher gebracht haben. Schliesslich will ich meinem Vater danken, der mich bei dieser Maturaarbeit sehr unterstützt hat.

32

## GameData.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class GameData

6 {

7 private GameObject[] enemies;

8 private TheThirdKind theThirdKind;

9

10 public GameData(Player p)

11 {

12 player = p;

13 enemies = GameObject.FindGameObjectsWithTag("Enemy");

14 GameObject[] interactives = GameObject.FindGameObjectsWithTag("Interactive");

15 foreach (GameObject go in interactives) {

16 if (go.name == "TheThirdKind") {

17 theThirdKind = go.GetComponent<TheThirdKind>();

18 break;

19 }

20 }

21 }

22

23 private Player player;

24 // Neues Speicherfile

25 public void InitNewGame()

26 {

27 player.InitState();

28 foreach (GameObject go in enemies) {

29 NPC npc = go.GetComponent<NPC>();

30 npc.InitState();

31 }

32 theThirdKind.InitState(player);

33 }

34 // Laden des Speicherfiles

35 public void LoadGame()

36 {

37 if (!PlayerPrefs.HasKey("GameState")) {

38 InitNewGame();

39 }

40 else {

41 player.LoadState(this);

42 foreach (GameObject go in enemies) {

43 NPC npc = go.GetComponent<NPC>();

44 npc.LoadState(this);

45 }

46 theThirdKind.LoadState(this, player);

47 }

48 }

49 // Schreiben des Speicherfiles

33

50 public void SaveGame()

51 {

52 PlayerPrefs.SetInt("GameState", 0);

53 player.SaveState(this);

54 foreach (GameObject go in enemies) {

55 NPC npc = go.GetComponent<NPC>();

56 npc.SaveState(this);

57 }

58 theThirdKind.SaveState(this, player);

59 }

60 // Speichern der Koordinaten

61 public void SaveTransform(string scope, Transform transform)

62 {

63 Vector3 position = transform.position;

64

65 PlayerPrefs.SetFloat(scope + "X", position.x);

66 PlayerPrefs.SetFloat(scope + "Y", position.y);

67 PlayerPrefs.SetFloat(scope + "Z", position.z);

68 }

69

70 public void SaveFloat(string scope, float f)

71 {

72 PlayerPrefs.SetFloat(scope, f);

73 }

74

75 public void SaveInt(string scope, int i)

76 {

77 PlayerPrefs.SetInt(scope, i);

78 }

79

80 public void SaveBool(string scope, bool b)

81 {

82 PlayerPrefs.SetInt(scope, b ? 1 : 0);

83 }

84

85 // Laden der Position

86 public void LoadTransform(string scope, Transform transform)

87 {

88 Vector3 position = new Vector3(0, 0, 0);

89 if (PlayerPrefs.HasKey(scope + "X")) {

90 position.x = PlayerPrefs.GetFloat(scope + "X");

91 position.y = PlayerPrefs.GetFloat(scope + "Y");

92 position.z = PlayerPrefs.GetFloat(scope + "Z");

93 transform.position = position;

94 }

95 }

96

97 public float LoadFloat(string scope, float def)

98 {

99 if (PlayerPrefs.HasKey(scope))

100 return PlayerPrefs.GetFloat(scope);

eturn

|  |  |  |
| --- | --- | --- |
| 101 |  | else |
| 102 |  | r |
| 103 | } |  |
| 104 |  |  |

def;

105 public int LoadInt(string scope, int def)

106 {

107 if (PlayerPrefs.HasKey(scope))

108 return PlayerPrefs.GetInt(scope);

109 else

110 return def;

111 }

112

113 public bool LoadBool(string scope, bool def)

114 {

115 if (PlayerPrefs.HasKey(scope))

116 return PlayerPrefs.GetInt(scope) != 0;

117 else

return def;

|  |  |  |
| --- | --- | --- |
| 118 |  |  |
| 119 |  | } |
| 120 |  |  |
| 121 | } |  |

## HUD.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4 using UnityEngine.UI;

5

6 public class HUD : MonoBehaviour

7 {

8 // Die Spielerinstanz, welche alle Informationen liefert

9 public Player player;

10

11 // Feld, welches die Sprechblase beinhaltet

12 private GameObject talking;

13 // Textobjekt, welches den gesprochenem Text anzeigt

14 private Text talkingText;

15 // Zeitpunkt, als die letzte Blase angezeigt wurde, 0 wenn keine angezeigt wird

16 private int speechDisplayedTime = 0;

17 // Darstellungszeit der Sprechblase

18 private const int speechDisplayDuration = 15;

19

20 // Gesundheit Schieberegler

21 private Slider healthSlider;

22 // Ausdauer Schieberegler

23 private Slider staminaSlider;

24 // Textobjekt welches das Geld anzeigt das der Spieler hat.

25 private Text moneyText;

26

27 // Initialisierung

28 void Start()

29 {

30 GetComponent<Canvas>().enabled = true;

31

32 talking = transform.Find("Talking").gameObject;

33 talkingText = transform.Find("Talking/Text").gameObject.GetComponent<Text>();

34 talking.SetActive(false);

35

36 healthSlider = transform.Find("HealthSlider").gameObject.GetComponent<Slider>();

37 staminaSlider = transform.Find("StaminaSlider").gameObject.GetComponent<Slider>();

38 moneyText = transform.Find("Money/Text").gameObject.GetComponent<Text>();

39 }

40

41 // Update wird pro frame einmal aufgerufen

42 // Statusinformationen aus Spieler übernehmen und anzeigen

43 void Update()

44 {

45 moneyText.text = player.regenerationPoints + " RP";

46 healthSlider.value = player.health;

47 staminaSlider.value = player.stamina;

48

49 string lastTalk = player.lastTalk;

50 // wenn mit dem Spieler seit letztem Mal gesprochen wurde

51 if (lastTalk.Length > 0) {

52 // anzeigen der Sprechblase

53 player.lastTalk = "";

54 talking.SetActive(true);

55 talkingText.text = lastTalk;

56 // Zeitpunkt des Anzeigens merken

57 speechDisplayedTime = (int)Time.time;

58 }

59 else {

60 // kein neuer Text, überprüfe ob die Sprechblase wieder versteckt werden soll

61 if (speechDisplayedTime > 0) {

62 if ((int)Time.time - speechDisplayedTime > speechDisplayDuration) { speechDisplayedTime = 0; talking.SetActive(false); talkingText.text = "";

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63 |  | | | | |
| 64 |
| 65 |  |  |  |  |  |
| 66 |  |  |  |  | } |
| 67 |  |  |  | } |  |
| 68 |  |  | } |  |  |
| 69 |  | } |  |  |  |
| 70 | } |  |  |  |  |

## Player.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using System.Security.Cryptography.X509Certificates;

4 using UnityStandardAssets.CrossPlatformInput;

5 using UnityEngine;

6

7 // Steuerung der Spielfigur

8 public class Player : MonoBehaviour

9 {

10 private CharacterController m\_CharacterController;

11

12 // Auf dem boden?

13 private bool isGrounded;

14

15 // blockt der Spieler?

16 public bool isBlocking;

17

18 // Laufgeschwindigkeit

19 public float speed = 3f;

20 //private float towardsY = 0f;

21

22 // Sprungkraft

23 public float sprungkraft = 5f;

24

25 // Gesundheit

26 public float health = 100f;

27

28 // Ausdauer

29 public float stamina = 100f;

30

31 public int regenerationPoints = 0;

32

33 // Das Graphische Modell, ua für drehung in Laufrichtung

34 public GameObject model;

35

36 // Zeiger auf die animations komponente der spielfigur

37 private Animator anim;

38

39 // Physikkomponente

40 private Rigidbody rigid;

41

42 // Der Winkel zu dem sich die Figur um die eigene Achse (=Y) drehen soll

43 public float towardsY = 90f;

44

45 public string lastTalk;

46

47

48 public GameData gameData;

49

50 public Quests quests;

51

52 public GameObject flower;

53

54 // Abkürzungen etablieren

55 void Start()

56 {

57 gameData = new GameData(this);

58 gameData.LoadGame();

59 for (int i = 1; i < 21; i++) {

60 GameObject go = (GameObject)Instantiate(flower, new Vector3(Random.Range(-50, 58), 0, Random.Range(403, 566)),

|  |  |  |
| --- | --- | --- |
| 61 |  | Quaternion.identity); |
| 62 |  | go.name = "FlowerRed" + i; |
| 63 |  | Debug.Log(go.name + " at " + go.transform.position.x + " " + |
| 64 |  | go.transform.position.y + " " + go.transform.position.z); |
| 65 |  | } |
| 66 |  |  |
| 67 |  | rigid = GetComponent<Rigidbody>(); |
| 68 |  | m\_CharacterController = GetComponent<CharacterController>(); |
| 69 |  | anim = GetComponent<Animator>(); |
| 70 |  |  |
| 71 | } |  |
| 72 |  |  |

73 void Update()

74 {

|  |  |  |
| --- | --- | --- |
| 75 | // | ChangeStamina (Time.deltaTime \* 5); |
| 76 | if | (health < 50 && regenerationPoints > 0) { |
| 77 |  | int p = Mathf.Min(regenerationPoints, 10); |
| 78 |  | ChangeHealth(p); |
| 79 |  | regenerationPoints -= p; |
| 80 | } |  |
| 81 | if | (stamina < 50 & regenerationPoints > 0) { |
| 82 |  | int p = Mathf.Min(regenerationPoints, 10); |
| 83 |  | ChangeStamina(p); |
| 84 |  | regenerationPoints -= p; |

85 }

86 // überprüfe ob lebendig

87 // x und z Koordinaten Bewegung

88 float x = Input.GetAxis("Horizontal") \* Time.deltaTime \* speed;

89 float z = Input.GetAxis("Vertical") \* Time.deltaTime \* speed;

90

91

92 // Animationen

93 anim.SetFloat("forward", z \* 3);

94 anim.SetBool("Walking", true);

95 // raycast für "isgrounded"

96 RaycastHit hit;

97 // raycast für "Interact"

98 RaycastHit interactHit;

99

100 transform.Translate(x, 0, z);

101

102 // Springen

103 if (Input.GetAxis("Jump") > 0f) {

104 // isgrounded: Vektor Richtung Boden mit Länge 1.

105 // Wenn er etwas trifft, ist isgrounded true (was bedeutet,

106 // dass springen möglich ist). Wenn nicht, dann nicht.

107

wn,

|  |  |  |  |
| --- | --- | --- | --- |
| 108 |  | if | (Physics.Raycast(transform.position, Vector3.do |
| 109 |  |  | Debug.DrawLine(transform.position, hit.point); |
| 110 |  |  | print(hit.distance); |
| 111 |  |  | Vector3 power = rigid.velocity; |
| 112 |  |  | power.y = 5f; |
| 113 |  |  | rigid.velocity = power; |
| 114 |  | } |  |
| 115 | } |  |  |
| 116 |  |  |  |

out hit, 1)) {

117 // Angriff

118 if (Input.GetButtonDown("Fire1")) {

119 // hole und vergleiche Waffenwerte

120 GameObject weapon = GetComponent<WeaponManager>().GetActiveWeapon();

121 float weaponstamina = weapon.GetComponent<Stats>().weaponStamina;

122 string weaponname = weapon.GetComponent<Stats>().weaponName;

123 if (stamina >= weaponstamina) {

124 //Spiel Attackier-Animation

125 Debug.Log("Player attacked");

126 if (weaponname == "Katana" || weaponname == "Bo")

127 anim.Play("Katana 0");

128 else if (weaponname == "Hands")

129 anim.Play("Katana 0");

130

131 ChangeStamina(-weaponstamina);

132 }

133 }

134

135 // Interagieren (Standard "E")

136 if (Input.GetKeyDown("e")) {

137 Vector3 forward = transform.TransformDirection(Vector3.forward);

138

139 if (Physics.Raycast(transform.position, forward,

140 out interactHit, 10)) {

141 Debug.DrawLine(transform.position, interactHit.point);

142 if (interactHit.collider.gameObject.tag == "Interactive") {

143 // sende eine Nachricht

144 interactHit.collider.gameObject.SendMessage("Interact",

145 (Player)this);

146 // Aufgabe aktualisieren - wenn sie existiert

147 quests.Interacted(interactHit.collider.gameObject);

148 }

149 }

150

151 }

152

153 if (Input.GetAxis("Fire2") > 0f) {

154 // spiel Attackier-Animation

155

156 isBlocking = true;

157 anim.SetBool("Blocking", true);

158 }

159 else {

160 isBlocking = false;

161 anim.SetBool("Blocking", false);

162 }

163

164 // if (isBlocking == true) {

165 // Debug.Log ("Player Blocked");

166 // anim.Play ("Block");

167 // }

168 }

169

170 public void InitState()

171 {

172 transform.position = new Vector3(429, 0, 226);

173 transform.rotation = Quaternion.identity;

174 health = 100f;

175 stamina = 100f;

176 regenerationPoints = 0;

177 gameObject.SetActive(true);

178 quests.Clear();

179 }

180

181 public void LoadState(GameData gameData)

182 {

183 gameData.LoadTransform("player", transform);

184 health = gameData.LoadFloat("playerealth", 100f);

185 stamina = gameData.LoadFloat("playerstamina", 100f);

186 regenerationPoints = gameData.LoadInt("playerrp", 0);

187 }

188

189 public void SaveState(GameData gameData)

190 {

191 gameData.SaveTransform("player", transform);

192 gameData.SaveFloat("playerealth", health);

193 gameData.SaveFloat("playerstamina", stamina);

194 gameData.SaveInt("playerrp", regenerationPoints);

195 }

196

197 public void ChangeHealth(float change)

198 {

199 health += change;

200

201 if (health > 100.0f)

202 health = 100.0f;

203 else if (health < 0.0f) {

204 Debug.Log("YOU ARE DEAD");

205 gameData.InitNewGame();

206 }

207 }

208

209 public void ChangeStamina(float change)

210 {

211 stamina += change;

212

213 if (stamina > 100.0f)

214 stamina = 100.0f;

215 else if (stamina < 0.0f)

216 stamina = 0.0f;

217 }

218

219 public void AddRegenerationPoints(int extra)

220 {

221 regenerationPoints += extra;

222 }

223

224 // Fügt der Liste eine neue Aufgabe hinzu

225 public void AddQuest(Quest q)

226 {

227 quests.AddQuest(q);

228 }

229

230 // Sucht nach einer Aufgabe per Name

231 public Quest GetQuest(string name)

232 {

233 return quests.GetQuest(name);

234 }

235

236 // Sucht nach einer aktiven Aufgabe per Name

237 public Quest GetActiveQuest(string name)

238 {

return quests.GetActiveQuest(name);

|  |  |  |
| --- | --- | --- |
| 239 |  |  |
| 240 |  | } |
| 241 |  |  |
| 242 | } |  |

## InteractibleObject.cs

1 using System.Collections;

2 using System.Collections;

3 using System.Collections.Generic;

4 using UnityEngine;

5

6 public class InteractibleObject : MonoBehaviour

7 {

8

9

10 // Objekt bekommt einen Interaktionsaufruf

11 void Interact(Player p)

12 {

13 p.AddRegenerationPoints(15);

14 gameObject.SetActive(false);

15 }

16 }

## Menu.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class Menu : MonoBehaviour

6 {

7

8 public Player player;

9

10 void Start()

11 {

12 GetComponent<Canvas>().enabled = false;

13

14 }

15

16 /// <summary>

17 /// Wahr, wenn die Taste bereits zuvor als gedrückt erkannt wurde

18 /// Nötig, um Mehrfachauswertungen der Menütasten zu verhindern

19 /// </summary>

20 private bool keyWasPressed = false;

21

22 void Update()

23 {

24

25 if (Input.GetAxisRaw("Menu") > 0f) {

26 if (!keyWasPressed)

27 GetComponent<Canvas>().enabled =

!GetComponent<Canvas>().enabled;

28

29 keyWasPressed = true;

30 }

31 else

32 keyWasPressed = false;

33 }

34

35 // Beendet das Spiel

36 public void OnButtonEndPressed()

37 {

38 Debug.Log("Spiel beendet");

39 Application.Quit();

40 }

41 // Startet ein neues Spiel

42 public void OnButtonNewPressed()

43 {

44 player.gameData.InitNewGame();

45 }

46

47

48 //Speicherfunktion

49 public void OnButtonSavePressed()

50 {

51 Debug.Log("Speichern"); player.gameData.SaveGame();

|  |  |  |
| --- | --- | --- |
| 52 |  |  |
| 53 |  | } |
| 54 |  |  |
| 55 |  |  |
| 56 | } |  |

## MouseLookAtIt.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class MouseLookAtIt : MonoBehaviour

6 {

7 // Setzen der Grenzen.

8 public float minimumX = -60f;

9 public float maximumX = 60f;

10 public float minimumY = -360f;

11 public float maximumY = 360;

12

13 public float sensitivityX = 15f;

14 public float sensitivityY = 15f;

15

16 // Verbinden des Kameraobjekts

17 public Camera cam;

18

19 float rotationY = 0f;

20

21 float rotationX = 0f;

22

23 void Start()

24 {

25 // Mauscursor festhalten

|  |  |  |
| --- | --- | --- |
| 26 |  | Cursor.lockState = CursorLockMode.Locked; |
| 27 |  | } |
| 28 |  |  |
| 29 |  | void Update() |
| 30 |  | { |
| 31 |  | // Maus Koordinaten kriegen |
| 32 |  | rotationY += Input.GetAxis("Mouse X") \* sensitivityY; |
| 33 |  | rotationX += Input.GetAxis("Mouse Y") \* sensitivityX; |
| 34 |  |  |
| 35 |  | rotationX = Mathf.Clamp(rotationX, minimumX, maximumX); |
| 36 |  | // Bewegen des GameObjekts |
| 37 |  | transform.localEulerAngles = new Vector3(0, rotationY, 0); |
| 38 |  | cam.transform.localEulerAngles = new Vector3(-rotationX, 0, 0); |
| 39 |  | } |
| 40 | } |  |

## NPC.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityStandardAssets.CrossPlatformInput;

4 using UnityEngine;

5

6 public class NPC : MonoBehaviour

7 {

8 // Körperteile

9 public GameObject AI;

10 public GameObject AITorso;

11 public GameObject AILeftFoot;

12 public GameObject AIRightFoot;

13 public GameObject AILeftHand;

14 public GameObject AIRightHand;

15

16 //Spieler Object

17 public GameObject player;

18

19 // Field-of-View (kann sehen)

20 public GameObject FOV;

21

22 // Initialisiere NPC Values

23 public float health = 100f;

24 public float stamina = 100f;

25 public float speed = 2.0f;

26 public bool hasseenplayer = false;

27 public bool fighting = false;

28

29 // Physikkomponente

30 private Rigidbody rigid;

31

32 // Animationskomponente

33 private Animator anim;

34 private Collider fov;

35

36 public Transform target;

37 public RaycastHit hit;

38

39 private Transform originalTransform;

40

41 // find Player src [https://www.quora.com/How-do-I-make-an-NPC-move-in](http://www.quora.com/How-do-I-make-an-NPC-move-in-Unity)-Unity#

42 void Start()

43 {

44 // Verbinde Komponenten

45 rigid = GetComponent<Rigidbody>();

46 anim = GetComponent<Animator>();

47 fov = FOV.GetComponent<Collider>();

48 player = GameObject.FindGameObjectWithTag("Player");

49 FreedomQuest.npcsCreated += 1;

50 name = "A1\_" + FreedomQuest.npcsAlive;

51 originalTransform = gameObject.transform;

52 }

53

54 void Update()

55 {

56 // wenn Gesundheit unter 0, zerstöre Objekt

57 if (health < 0) {

58 Debug.Log("NPC HEALTH: " + health);

59 Destroy(AI);

60 }

61

62 if (hasseenplayer == true) {

63 // wenn Spieler in Reichweite, angreifen (fighting = true )

64 Vector3 forward = transform.TransformDirection(Vector3.forward);

65 if (Physics.Raycast(transform.position, forward, out hit, 10)) {

66

67 Debug.DrawLine(transform.position, hit.point); "Player") {

|  |  |  |
| --- | --- | --- |
| 68 | if | (hit.collider.gameObject.tag == |
| 69 |  | fighting = true; |
| 70 |  | Debug.Log("NPC is attacking"); |
| 71 | } |  |
| 72 | else { | |
| 73 |  | fighting = false; |
| 74 |  | anim.ResetTrigger("Attack"); |

75 }

76

77 if (fighting == true) {

78 // Animation Angriff starten

79 anim.SetTrigger("Attack");

80 }

81 else {

82 // Animation Laufen starten

83 anim.SetBool("iswalking", true);

84

85 if (player.transform.position.x >

86 (transform.position.x - 1)) {

87 // nach rechts

88 transform.position +=

89 new Vector3(speed \* Time.deltaTime, 0, 0);

90

91 }

92 else {

|  |  |  |  |
| --- | --- | --- | --- |
| 93 |  | // nach links |  |
| 94 |  | transform.position -= |
| 95 |  | new Vector3(speed | \* Time.deltaTime, 0, 0); |
| 96 | } |  |  |
| 97 |  |  |  |

98 // in Richtung Spieler

99 if (player.transform.position.z > transform.position.z) {

100 // nach oben

101 transform.position +=

102 new Vector3(0, 0, speed \* Time.deltaTime);

103

104 }

105 else {

106 // nach unten

107 transform.position -=

108 new Vector3(0, 0, speed \* Time.deltaTime);

109 }

110 //

https://docs.unity3d.com/ScriptReference/Transform.LookAt.html

111 transform.LookAt(target);

112 }

113 }

114 else {

anim.SetBool("iswalking", false);

|  |  |  |  |
| --- | --- | --- | --- |
| 115 |  |  |  |
| 116 |  |  | } |
| 117 |  |  |  |
| 118 |  |  |  |
| 119 |  |  |  |
| 120 |  |  |  |
| 121 |  | } |  |
| 122 | } |  |  |
| 123 |  |  |  |

124 // Entdecken des Spielers

125 void OnTriggerEnter(Collider fov)

126 {

127 if (fov.name == "Player") {

128 Debug.Log("has seen");

129 hasseenplayer = true;

130 }

131

132 }

133

134 public void ChangeHealth(float change)

|  |  |  |  |
| --- | --- | --- | --- |
| 135 |  | { |  |
| 136 |  |  | health += change; |
| 137 |  |  |  |
| 138 |  |  | if (health > 100.0f) |
| 139 |  |  | health = 100.0f; |
| 140 |  |  | else if (health < 0.0f) { |
| 141 |  |  | Debug.Log(gameObject.name + " I’M DEAD"); |
| 142 |  |  | gameObject.SetActive(false); |
| 143 |  |  | FreedomQuest.npcsAlive -= 1; |
| 144 |  |  |  |
| 145 |  |  | Player p = player.GetComponent<Player>(); |
| 146 |  |  | // Dem Player extra Punkte geben |
| 147 |  |  | p.AddRegenerationPoints(10); |
| 148 |  |  | } |
| 149 |  | } |  |
| 150 |  |  |  |
| 151 |  | public void InitState() | |
| 152 |  | { | |
| 153 |  | health = 100f; | |
| 154 |  | stamina = 100f; | |
| 155 |  | hasseenplayer = false; | |
| 156 |  | fighting = false; | |
| 157 |  | gameObject.SetActive(true); | |
| 158 |  | transform.SetPositionAndRotation(originalTransform.position, | |
| 159 |  | originalTransform.rotation); | |
| 160 |  | } | |
| 161 |  |  | |
| 162 |  | public void LoadState(GameData gameData) | |
| 163 |  | { | |
| 164 |  | gameData.LoadTransform(name, transform); | |
| 165 |  | health = gameData.LoadFloat(name + "ealth", 100f); | |
| 166 |  | stamina = gameData.LoadFloat(name + "stamina", 100f); | |
| 167 |  | gameObject.SetActive(gameData.LoadBool(name + "active", true)); | |
| 168 |  | hasseenplayer = gameData.LoadBool(name + "hasseenplayer", false); | |
| 169 |  | fighting = gameData.LoadBool(name + "fighting", false); | |
| 170 |  | } | |
| 171 |  |  | |
| 172 |  | public void SaveState(GameData gameData) | |
| 173 |  | { | |
| 174 |  | gameData.SaveTransform(name, transform); | |
| 175 |  | gameData.SaveFloat(name + "ealth", health); | |
| 176 |  | gameData.SaveFloat(name + "stamina", stamina); | |
| 177 |  | gameData.SaveBool(name + "active", gameObject.activeSelf); | |
| 178 |  | gameData.SaveBool(name + "hasseenplayer", hasseenplayer); | |
| 179 |  | gameData.SaveBool(name + "fighting", fighting); | |
| 180 |  | } | |
| 181 |  |  | |
| 182 | } |  | |

## Stats.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class Stats : MonoBehaviour

6 {

7 // Animationen

8 public Animation AttackAnimation;

9 // Waffen Werte

10 public string weaponName = "Katana";

11 public float weaponDamage = 35;

12 public float weaponDefense = 15;

13 public float weaponRange = 3;

14 public float weaponSpeed = 3;

15 public float weaponStamina = 10;

16

17 }

## TheThirdKind.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class TheThirdKind : InteractibleObject

6 {

7 private GameObject flowerRed;

8 private GameObject flowerBlue;

9

10 void Start()

11 {

12 GameObject[] interactives = GameObject.FindGameObjectsWithTag("Interactive");

13 foreach( GameObject go in interactives ){

14 if ( go.name == "FlowerRed" ) {

15 flowerRed = go;

16 }

17 else if ( go.name == "FlowerBlue" ) { flowerBlue = go;

|  |  |  |  |
| --- | --- | --- | --- |
| 18 |  |  |  |
| 19 |  |  | } |
| 20 |  | } |  |
| 21 | } |  |  |
| 22 |  |  |  |

23 void Update()

24 {

25

26 }

27

28 private void CreateDummyQuest(Player p)

29 {

30 p.AddQuest(

31 new Quest("TheThirdKind", "You must search the wise green man"));

32 p.lastTalk = "You must search the wise green man";

33 }

34

35 private void CreateFirstQuest(Player p)

36 {

37 p.AddQuest(new Quest("FlowerRed", "Go and pick the flower behind the house"));

38 p.AddQuest(new Quest("FlowerBlue", "Go and pick the flower at the corner"));

39

40 p.AddQuest(new Quest("TheThirdKind", "You must come back to the wiser green man"));

41 p.lastTalk = "You must pick the two flowers and come back to me. The picked flowers give you regeneration points";

42 }

43

44 private void CreateSecondQuest(Player p)

45 {

46 p.AddQuest(new FreedomQuest("You must fight all red figures"));

47 p.lastTalk = "Great, now you must fight the red figures. Pick the flowers you find to get regeneration points";

48 }

49

50 void Interact(Player p)

51 {

52 Quest qFlowerRed = p.GetQuest("FlowerRed");

53 Quest qFlowerBlue = p.GetQuest("FlowerBlue");

54

55 if (qFlowerRed == null) {

56 CreateFirstQuest(p);

57 }

58 else if (qFlowerRed.IsDone() && qFlowerBlue.IsDone()) {

59 Quest enemies = p.GetQuest("A1");

60 // Beim ersten mal zurückkommen wenn die Aufgabe noch nicht erledigt ist

61 // wird einmal durchlaufen

62 if (enemies == null) {

63 CreateSecondQuest(p);

64 }

65 }

66 else if (qFlowerRed.IsDone() != qFlowerBlue.IsDone()) {

67 // Sonst wird diese Nachricht ausgelöst nachem die erste Aufgabe erledigt wurde.

p.lastTalk = "You have not yet picked both flowers";

|  |  |  |
| --- | --- | --- |
| 68 |  |  |
| 69 |  | } |
| 70 | } |  |
| 71 |  |  |

72 public void InitState(Player p)

73 {

74 CreateDummyQuest(p);

75 flowerBlue.SetActive(true);

76 flowerRed.SetActive(true);

77 }

78

79 public void LoadState(GameData gameData, Player p)

80 {

81 int level = gameData.LoadInt("thethirdkindlevel",0);

82 flowerBlue.SetActive(true);

83 flowerRed.SetActive(true);

84 CreateDummyQuest(p);

85

86 if ( level == 0 ) {

87 // setup;

88 }

89 else {

90 // mindestens level 1

91 p.GetActiveQuest("TheThirdKind").Done();

92

93 CreateFirstQuest(p);

94 Quest qFlowerRed = p.GetQuest("FlowerRed");

95 Quest qFlowerBlue = p.GetQuest("FlowerBlue");

96

97 if ( level == 2 ) {

98 qFlowerRed.Done();

99 flowerRed.SetActive(false);

100 }

else if ( level == 3 ) { qFlowerBlue.Done(); flowerBlue.SetActive(false);

|  |  |  |
| --- | --- | --- |
| 101 |  | |
| 102 |
| 103 |
| 104 |
| 105 |
| 106 |
| 107 |
| 108 |
| 109 |
| 110 |
| 111 |
| 112 |
| 113 |
| 114 |
| 115 |
| 116 |
| 117 |  | } |
| 118 | } |  |
| 119 |  |  |

}

else if ( level >= 4 ) { flowerBlue.SetActive(false); flowerRed.SetActive(false); qFlowerRed.Done(); qFlowerBlue.Done();

if ( level == 5 ) { p.GetActiveQuest("TheThirdKind").Done(); CreateSecondQuest(p);

FreedomQuest.npcsAlive = gameData.LoadInt("npcsalive", FreedomQuest.npcsCreated);

}

}

120 public void SaveState(GameData gameData, Player p)

121 {

Quest qFlowerRed = p.GetQuest("FlowerRed"); Quest qFlowerBlue = p.GetQuest("FlowerBlue");

|  |  |  |
| --- | --- | --- |
| 122 |  | |
| 123 |
| 124 |
| 125 |
| 126 |
| 127 |
| 128 |
| 129 |
| 130 |
| 131 |
| 132 |
| 133 |
| 134 |
| 135 |
| 136 |
| 137 |
| 138 |
| 139 |
| 140 |
| 141 |
| 142 |
| 143 |
| 144 |
| 145 |
| 146 |
| 147 |
| 148 |  | } |
| 149 | } |  |

int level = 0;

if (qFlowerRed == null) {

// level ist 0

}

else if (qFlowerRed.IsDone() && qFlowerBlue.IsDone()) { Quest enemies = p.GetQuest("A1");

if ( enemies != null ) level = 5;

else

}

level = 4;

else {

if (qFlowerRed.IsDone() != qFlowerBlue.IsDone()) { if ( qFlowerRed.IsDone() )

level = 2;

else if ( qFlowerBlue.IsDone() ) level = 3;

}

else

}

level = 1;

gameData.SaveInt("thethirdkindlevel",level); gameData.SaveInt("npcsalive",FreedomQuest.npcsAlive);

## WeaponHit.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class WeaponHit : MonoBehaviour

6 {

7 // Dieses Skript fügt Schaden in Höhe des WeaponDamage Werts

8 // einer Waffe einem Objekt zuzufügen

9

10 // Platzhalter für Waffenwerte

11 private float damage;

12 private string weaponname;

13 private float defense;

14 private float range;

15 private float speed;

16 private float stamina;

17

18 public GameObject Self;

19

20 // Das Element, welches die Trefferzone darstellt, muss einen Physik

21 // Collider haben

22

23 void Start()

24 {

25 // abfüllen der Stats

26 weaponname = Self.GetComponent<Stats>().weaponName;

27 damage = Self.GetComponent<Stats>().weaponDamage;

28 defense = Self.GetComponent<Stats>().weaponDefense;

29 range = Self.GetComponent<Stats>().weaponRange;

30 speed = Self.GetComponent<Stats>().weaponSpeed;

31 stamina = Self.GetComponent<Stats>().weaponStamina;

32 }

33

34 // Wenn der Weapon-Collider etwas trifft, überprüfe ob etwas mit dem Tag

35 // "Enemy" oder dem Namen "Player" getroffen wurde und füge Schaden zu

36 void OnTriggerEnter(Collider col)

37 {

38 Debug.Log("Hit: " + col.name + " Weapon: " + weaponname);

39 if (col.tag == "Enemy") {

40 col.gameObject.GetComponentInParent<NPC>().ChangeHealth(-damage);

41 Debug.Log(damage + " Damage applied to " + col.name);

42 }

43 else if (col.name == "Player") {

44 col.gameObject.GetComponentInParent<Player>().ChangeHealth(-damage);

45 Debug.Log(damage + " Damage applied to " + col.name);

}

|  |  |  |
| --- | --- | --- |
| 46 |  |  |
| 47 |  | } |
| 48 |  |  |
| 49 | } |  |

## WeaponManager.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 /// Diese Klasse verwaltet die Verfügbarkeit und Ausstattung von Waffen

6 public class WeaponManager : MonoBehaviour

7 {

8 public List<GameObject> weaponList;

9 public List<GameObject> enabledWeaponList;

10

11 /// Waffen

12 public GameObject Bo;

13 public GameObject Katana;

14 public GameObject Hands;

15

16 public string equipped;

17

18 private GameObject weaponInHand;

19

20 // Bool zur Unterscheidung von NPC und Player

21 public bool isplayer;

22

23 // inspiriert von

24 //

https://forum.unity.com/threads/help-with-multiple-weapons-switching.465702/

25

26 void Start()

27 {

28 weaponList = new List<GameObject>(); // alle Waffen

29 enabledWeaponList = new List<GameObject>(); // alle verfügbaren Waffen

30

31

32 weaponList.Add(Katana);

33 weaponList.Add(Bo);

34 weaponList.Add(Hands);

35

36

37 enabledWeaponList.Add(Katana);

38 enabledWeaponList.Add(Bo);

39 enabledWeaponList.Add(Hands);

40

41 // mit Waffe ausrüsten

42 setActiveWeapon(Katana);

43 }

44

45 void Update()

46 {

47 // Wechseln der Waffe

48 if (isplayer == true) {

49 if (Input.GetAxis("1") > 0f) {

50 setActiveWeapon(Katana);

51 equipped = "Katana";

52 }

53

54 if (Input.GetAxis("2") > 0f) {

) {

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 55 | setActiveWeapon(Bo); equipped = "Bo"; | | | |
| 56 |
| 57 |  |  | } | |
| 58 |  |  |  | |
| 59 |  |  | if | (Input.GetAxis("3") > 0f |
| 60 |  |  |  | setActiveWeapon(Hands); |
| 61 |  |  |  | equipped = "Hands"; |
| 62 |  |  | } |  |
| 63 |  | } |  |  |
| 64 | } |  |  |  |
| 65 |  |  |  |  |

66 // die activeWeapon aktivieren und anderen zu deaktivieren, sonst

67 // erscheinen sie gleichzeitig

68 private void setActiveWeapon(GameObject activeWeapon)

69 {

70 for (int j = 0; j < weaponList.Count; j++) {

71 if (weaponList[j] == activeWeapon) {

72 for (int i = 0; i < enabledWeaponList.Count; i++) {

73 if (enabledWeaponList[i] == activeWeapon) {

74 weaponList[j].SetActive(true);

75 weaponInHand = activeWeapon;

76 }

77 }

78 }

79 else {

weaponList[j].SetActive(false);

|  |  |  |  |
| --- | --- | --- | --- |
| 80 |  |  |  |
| 81 |  |  | } |
| 82 |  | } |  |
| 83 | } |  |  |
| 84 |  |  |  |

85 // füge eine Waffe zu der Liste der verfügbaren Waffen,

86 // wenn man darüber läuft

87 public void PickedUpWeapon(GameObject weaponPickedUp)

88 {

89 enabledWeaponList.Add(weaponPickedUp);

90 }

91

92 public GameObject GetActiveWeapon()

93 {

94 return weaponInHand;

95 }

96 }

## Quest.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class Quest

6 {

7 private string name;

8 private string task;

9 private bool done;

10

11 // Eine Aufgabe mit dem namen des Zielobjekts und einer Aufgabenbeschreibung.

12 public Quest(string nameString, string taskString)

13 {

14 name = nameString;

15 task = taskString;

16 done = false;

17 }

18

19 virtual public bool IsDone()

20 {

21 return done;

22 }

23

24 public bool Done()

25 {

26 done = true;

27 return IsDone();

28 }

29

30 public string Name()

31 {

32 return name;

33 }

34

35 virtual public string Task()

36 {

37 return task;

38 }

39 };

## FreedomQuest.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4

5 public class FreedomQuest : Quest

6 {

|  |  |  |
| --- | --- | --- |
| 7 |  | public static int npcsCreated; |
| 8 |  | public static int npcsAlive; |
| 9 |  |  |
| 10 |  | public FreedomQuest(string taskString) : base("A1", taskString) |
| 11 |  | { |
| 12 |  | npcsAlive = npcsCreated; |
| 13 |  | } |
| 14 |  |  |
| 15 |  | public override bool IsDone() |
| 16 |  | { |
| 17 |  | return npcsAlive < 1; |
| 18 |  | } |
| 19 |  |  |
| 20 |  | public override string Task() |
| 21 |  | { |
| 22 |  | return "Fight the remaining\n" + npcsAlive + " red figures"; |
| 23 |  | } |
| 24 | } |  |

## Quests.cs

1 using System.Collections;

2 using System.Collections.Generic;

3 using UnityEngine;

4 using UnityEngine.UI;

5

6 public class Quests : MonoBehaviour

7 {

8 private List<Quest> questlist;

9

10 public Text Questtext;

11

12 void Start()

13 {

14 questlist = new List<Quest>();

15 }

16

17 void Update()

18 {

19 bool everythingDone = true;

20

21 foreach (Quest q in questlist) {

22 if (!q.IsDone()) {

23 Questtext.text = q.Task();

24 everythingDone = false;

25 break;

26 }

27 }

28

29 if (everythingDone) {

Questtext.text = "Mission accomplished";

|  |  |  |
| --- | --- | --- |
| 30 |  |  |
| 31 |  | } |
| 32 | } |  |
| 33 |  |  |

34 public void Clear()

35 {

36 questlist.Clear();

37 }

38

39 public void Interacted(GameObject obj)

40 {

41 Quest q = GetActiveQuest(obj.name);

42

43 if (q != null) {

44 if (q.Done())

45 Questtext.text = "";

46 }

47 }

48

49 // fügt der Liste eine neue Aufgabe hinzu

50 public void AddQuest(Quest q)

51 {

52 questlist.Add(q);

53 }

54

55 // Sucht eine Aufgabe nach Name

56 public Quest GetQuest(string name)

57 {

58 foreach (Quest q in questlist) {

59 if (name.StartsWith(q.Name())) {

60 return q;

61 }

62 }

63 return null;

64 }

65

66 // Sucht eine aktive Aufgabe nach Name

67 public Quest GetActiveQuest(string name)

68 {

69 foreach (Quest q in questlist) {

70 if (!q.IsDone() && name.StartsWith(q.Name())) {

71 return q;

72 }

73 }

74 return null;

75 }

76 }

2.1 Unity Benutzeroberfläche . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2.2 3D Modell der Blume in Blender . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3 Breakpoint im MonoDevelop Debugger mit Anzeige der Variablen . . . . . . . . . . . . . . . 4 2.4 Basisklassen in Unity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.5 Der Inputmanager mit horizontaler und vertikaler Achse, zugeordnet an die Tasten a-d/s-w . . 6 2.6 Rigidbodykomponente . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7 2.7 Kollisionskomponente des Katanas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

* 1. Vereinfachen des GetComponent<> Aufrufs 10
  2. Menü 11
  3. HUD 13
  4. Schieberegler Konfiguration 13
  5. Player 3D Modell 15
  6. Animationskurven 18
  7. Animator mit Zuständen und deren Übergängen 19
  8. Katana 19

3.9 Bo 20

* 1. Bo Stats Werte 20
  2. Vektorabfrage bei Interaktion mit TheThirdKind 22
  3. Interact in der Klasse InteractibleObject und die abgeleitete Klasse TheThirdKind . 23 3.13 NPC 3D Modell 24
  4. Sichtfeld des NPCs 24
  5. TheThirdKind 3D Modell 26

54

[1] Atlassian. *Sourcetree*. 2018. URL: [https://www.sourcetreeapp.com](http://www.sourcetreeapp.com/). [2] Blender Foundation. *Blender*. 2018. URL: [https://www.blender.org](http://www.blender.org/). [3] Pascal Brachet. *Texmaker*. 2018. URL: [www.xm1math.net/texmaker.](http://www.xm1math.net/texmaker)

[4] ChangeVision Inc. *astah UML*. 2018. URL: https://astah.net/editions/uml-new.

[5] XXXX. *Maturaarbeit*. 2019.   
[6] Git. *Git*. 2018. URL: https://git-scm.com.

[7] GitHub Inc. *GitHub*. 2018. URL: https://github.com.

[8] Joseph Manley. *How do I make an NPC move in Unity*. 2016. URL: https://[www.quora.com/](http://www.quora.com/) How-do-I-make-an-NPC-move-in-Unity.

[9] Microsoft Corporation. *Leitfaden für C#*. 2018. URL: https:// docs. microsoft. com/ de- de/dotnet/csharp/.

[10] MonoDevelop Project. *MonoDevelop*. 2018. URL: [https://www.monodevelop.com](http://www.monodevelop.com/).

[11] Simon Schälli. *Kantonsschule Wattwil - Maturaarbeit Template*. 2016. URL: https://www.overleaf. com/latex/templates/kantonsschule-wattwil-maturaarbeit-template/pswptsnytrgx.

[12] The LaTeX Project. *LaTeX*. 2018. URL: [https://www.latex-project.org](http://www.latex-project.org/).

[13] Unity. *Unity 3D Tutorial*. 2018. URL: https://unity3d.com/de/learn/tutorials.

[14] Unity. *Unity Glossary*. 2018. URL: https://docs.unity3d.com/Manual/Glossary.html. [15] Unity. *Unity Manual, Animation Window Guide*. 2018. URL: https :/ / docs . unity3d . com /

Manual/AnimationEditorGuide.html.

[16] Unity. *Unity Manual, Input Manager*. 2018. URL: https:// docs. unity3d. com/ Manual/ class-InputManager.html.

[17] Unity. *Unity Manual, Rigidbody Overview*. 2018. URL: https://docs.unity3d.com/Manual/ RigidbodiesOverview.html.

[18] Unity. *Unity Overview*. 2018. URL: https://docs.unity3d.com/Manual/UnityOverview. html.

[19] Unity. *Unity Script Reference*. 2018. URL: https://docs.unity3d.com/ScriptReference/ index.html.

[20] Unity. *Unity Script Reference, GameObject*. 2018. URL: https://docs.unity3d.com/ScriptReference/ GameObject.html.

[21] Unity. *Unity Script Reference, MonoBehaviour*. 2018. URL: https : / / docs . unity3d . com / ScriptReference/MonoBehaviour.html.

[22] Unity Forum. *Help with multiple weapons switching*. 2018. URL: https://forum.unity.com/ threads/solved-missing-prefabs-with-freshly-downloaded-project-windows. 411721/.

[23] Unity Forum. *Missing Prefabs with freshly downloaded project*. 2018. URL: https : / / forum . unity.com/threads/help-with-multiple-weapons-switching.465702/.

55