Dokumentation: Nightmare at School



Inhaltsverzeichnis

[1. Spielbeschreibung 3](#_Toc168517081)

[Was ist das für ein Spiel? 3](#_Toc168517082)

[Um was geht es in dem Spiel? 3](#_Toc168517083)

[Anmerkungen 3](#_Toc168517084)

[2. Spielmechanik 4](#_Toc168517085)

[Steuerung 4](#_Toc168517086)

[Visualisierung inkl. GUI (Graphic User Interface) 4](#_Toc168517087)

[Gesundheit (playerHP) 4](#_Toc168517088)

[3. Level-Design 5](#_Toc168517089)

[Nacht 1 5](#_Toc168517090)

[4. Assets 6](#_Toc168517091)

[Charaktere 6](#_Toc168517092)

[- PlayerCharacter 6](#_Toc168517093)

[- Mr. Musicman 6](#_Toc168517094)

[- Mr. Heisen 6](#_Toc168517095)

[- „Mr. Blaschke“ 6](#_Toc168517096)

[Environment (Umgebung) 6](#_Toc168517097)

[Eastereggs 7](#_Toc168517098)

[5. Skripte 8](#_Toc168517099)

[LoadingScene 9](#_Toc168517100)

[LoadingBehaviour.cs 9](#_Toc168517101)

[MainGame 12](#_Toc168517102)

[CameraBehaviour.cs 16](#_Toc168517103)

[MrMusicmanBehaviour.cs 18](#_Toc168517104)

[PlayerBehaviour.cs 19](#_Toc168517105)

[6. Tools 22](#_Toc168517106)

[Unity 22](#_Toc168517107)

[Blender 22](#_Toc168517108)

[7. Fazit 23](#_Toc168517109)

[Selbstreflexion und -einschätzung 23](#_Toc168517110)

# 1. Spielbeschreibung

## Was ist das für ein Spiel?

Unser Spiel ist ein Softcore-Horror-Game.

## Um was geht es in dem Spiel?

Jeder war mal ein Schüler und man kennt es, dass man Teste verhauen hat. Auch du gehörst dazu und machst es dir zur Aufgabe in die Schule einzubrechen und den Test zu klauen. Doch wusstest du nicht, dass Mr. Musicman auf dich in der Schule wartet. Dazu kommt noch, dass zahlreiche Hindernisse auf dich warten. Einzigarte Dialoge und Quests mit den Lehrern warten auf dich, die dir helfen Mr. Musicman zu überlisten!

Schaffst du es den Schülern zu helfen und dich mit den Lehrern anzufreunden?

Und am wichtigsten, komm nicht in den Chor!

## Anmerkungen

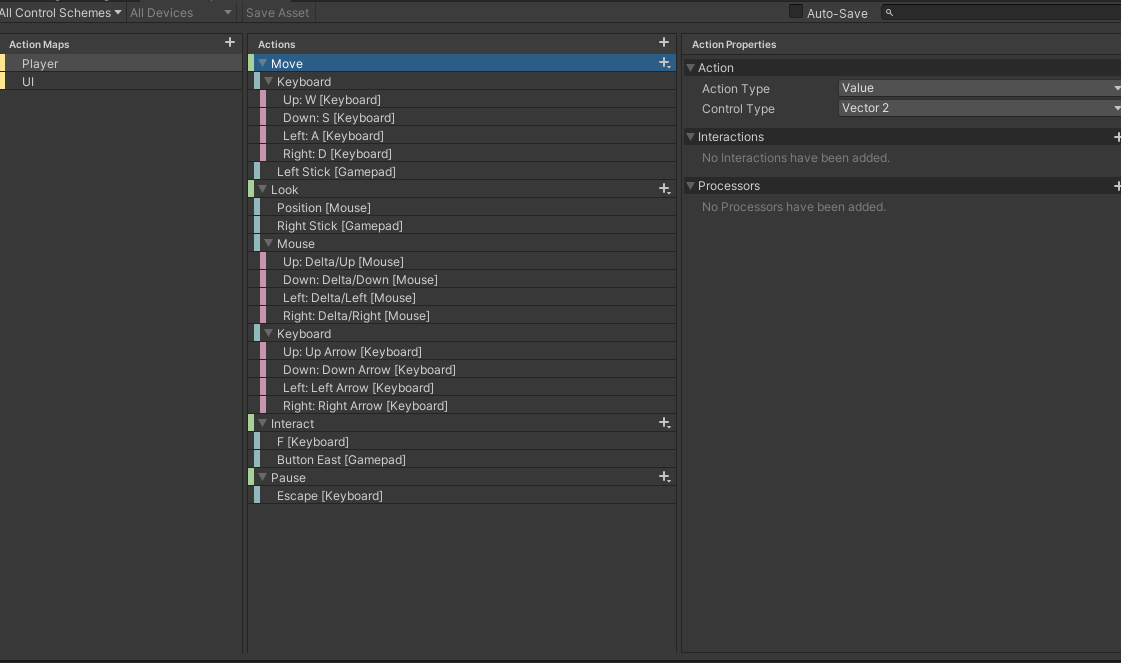
Nicht alles in dieser Test Version ist für die finale Version des Spiels vorgesehen. Das Spiel befindet sich derzeit in der Entwicklungsphase. Es wird eine verbesserte Version des Spiels geben, jedoch ist dies in diesem Zeitraum nicht möglich.

Nicht alle der oben genannten Tätigkeiten sind bereits implementiert (z.B. Dialoge).

# 2. Spielmechanik

## Steuerung

Der Spieler kann den Charakter durch verschiedene Tasten und andere Eingabemethoden kontrollieren.



Weitere Infos gibt es unter PlayerBehaviour.cs.

## Visualisierung inkl. GUI (Graphic User Interface)

Die Grafik wird durch Unity dargestellt und ist ein First-Person-View. Das heißt die Kamera steckt dem Charakter sozusagen „im Kopf“. Mehr dazu unter dem Abschnitt PlayerCharacter.

## Gesundheit (playerHP)

Der Spieler hat eine begrenzte Anzahl von Leben bevor der Spielstand zurückgesetzt wird.

Die Gesundheit wird verringert durch Begegnungen mit Mr. Musicman.

# 3. Level-Design

Die Level sind in Nächte unterteilt. Dabei stellt jede Nacht ein Level dar.

## Nacht 1

* Der Spieler hat das Ziel einen missglückten Physik Test zu stehlen.
* Dabei patrouilliert jedoch „Mr. Musciman“ auf dem Flur
* Der Spieler darf nicht in die Nähe dieses Charakters kommen
  + Falls dies geschieht ist das Spiel verloren

# 4. Assets

## Charaktere

### PlayerCharacter

* + - Dieser Charakter ist der vom Spieler durch Input gesteuerter Charakter.
    - In dem GameObject „Head“ ist die „Main Camera” eingesetzt um eine gute First-Person-Experience zu gewährleisten.

### Mr. Musicman

* + - Dies ist der Hauptgegner.
    - Er versucht den Spieler zu fangen und in den Chor zu bringen.
      * Falls dies geschieht ist das Spiel verloren.

### Mr. Heisen

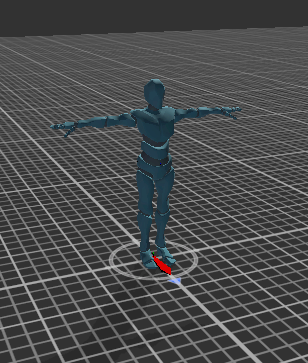
* + - Ein Cannabisabhängiger Chemielehrer.

### „Mr. Blaschke“

* + - Der einzige relativ konstant mit dem Spieler sympathisierender Charakter.
    - Kann den Spieler einmal vor dem Chor retten.

Die Charaktere sind derzeit nur temporär, aufgrund von technischen Pannen (Computerviren), wie bereits besprochen.

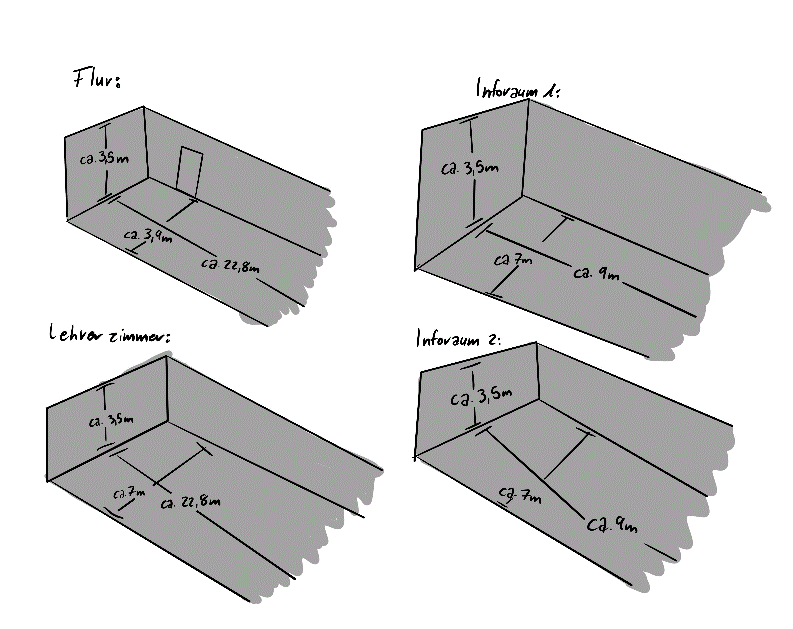
Die aktuellen Charaktere sind humanoide Figuren, welche von <https://www.mixamo.com> gratis zur Verfügung gestellt wurden. Teilweise sind die im Spiel genutzten humanoiden Animationen eine modifizierte Version von ebenfalls dieser Website.

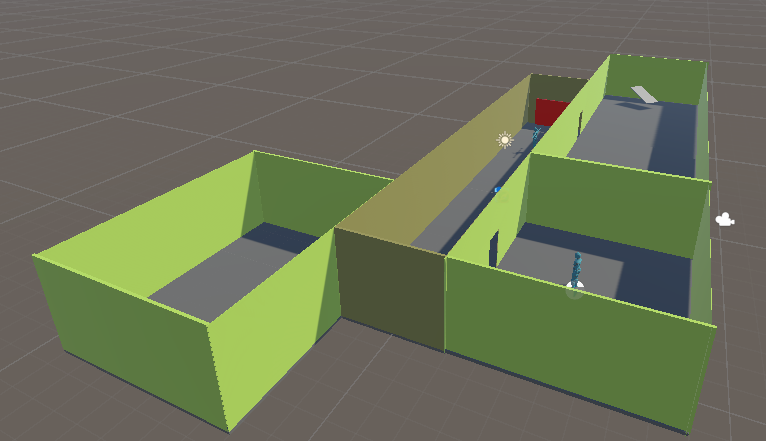


## Environment (Umgebung)

Das Spiel findet in einer Schule statt. Dementsprechend sind viele gestalterische Elemente gewöhnliche Dinge aus einer Schule. Zum Beispiel Tische und Stühle.

Die Spielumgebung ist mit den Programm Blender erstellt worden. Die einzelnen Böden sowie Wände bestehen aus dem Objekt „Cube (Würfel)“. Die Cubes wurden mit Hilfe von Messungen so gut wie möglich umgesetzt, um der Umgebung des RTG‘s zu entsprechen und eine Lebensechte Größe zu erhalten. Die Türdurchgänge von weiteren Cubes mithilfe von dem Tool „Boolean“ eingeschnitten worden und somit ermöglicht andere Räume zu betreten sowie zu verlassen.





## Eastereggs

Es gibt verschiedene versteckte Eastereggs (Überraschungen), welche an einigen Stellen im Spiel unter bestimmten Bedingungen erreichbar sind.

# 5. Skripte

Es ist sinnvoll dies nach den unterschiedlichen Szenen zu ordnen.

Noch einige Anmerkungen:

Wenn eine Klasse von MonoBehaviour ableitet ist, erbt diese einige Funktionen wie zum Beispiel Start(), Update(), Awake() und noch weitere. Die Funktion dieser ist bei jedem Skript als Summary beigefügt.

Wenn eine Klasse von einer Klasse abstammt, welche von MonoBehaviour abstammt, erbt diese ebenfalls sämtliche Funktionen.

## LoadingScene

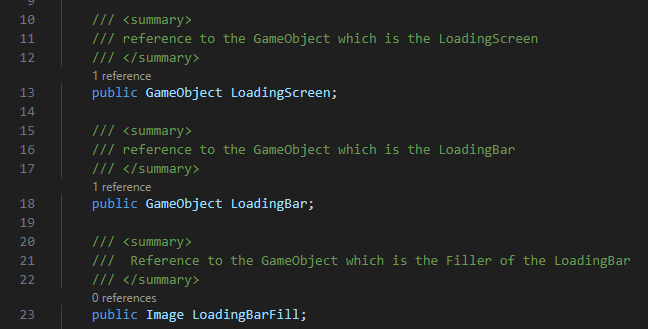
### LoadingBehaviour.cs

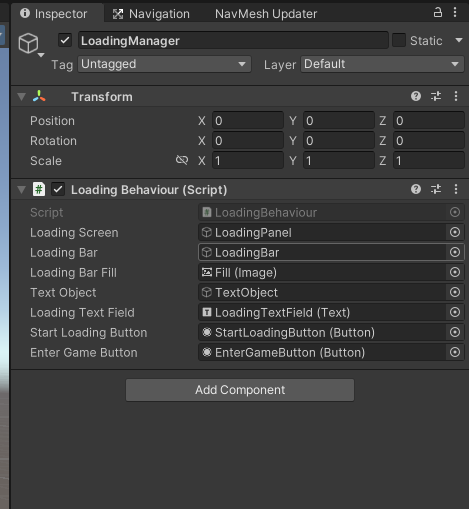
* + Steuert sämtliche möglichen Formen des synchronen und asynchronen ladens einer Szene.
  + Wird derzeit nur dazu verwendet die Szene MainGame zu laden. Dies geschieht asynchron. Das bedeutet das die Szene im Hintergrund geladen wird. Dies erlaubt es dem Spieler trotzdem mit der aktuellen Szene zu interagieren und zusätzlich kann die Szene auch während eines Zeitpunktes, mit niedriger Ressourcenauslastung geladen werden und dann bei Bedarf nahezu instantan aktiviert zu werden.
* using System.Collections;
* using System.Collections.Generic;
* using UnityEngine;
* using UnityEngine.UI;
* using UnityEngine.SceneManagement;
* public class LoadingBehaviour : MonoBehaviour
* {

Es wird zusätzlich zu den Standard Namespaces noch die von Unity bereitgestellten für UI und für SzenenMangement genutzt.

Als erstes werden Lokale Objekte erstellt um eine Referenz zu verschiedenen UI-Elementen herzustellen. Die Referenz ist zugewiesen im Unity Inspector.

Beispiel Ausschnitt aus dem Code:

  
Screenshot vom Inspector:

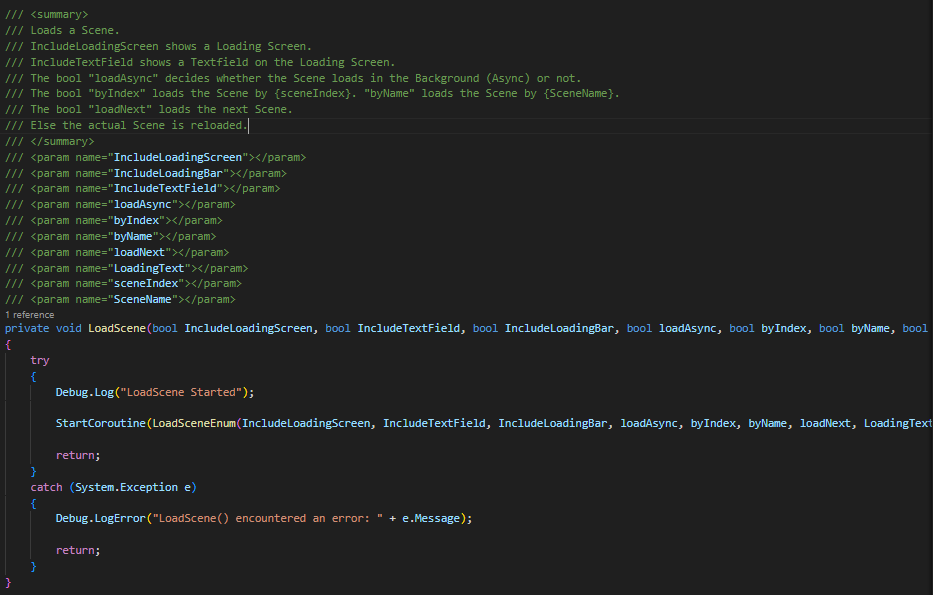


Als erstes wird dem Button StartLoadingButton ein Listener hinzugefügt, der bei Aktivierung des Buttons die Funktion LoadButton() aufruft.

Diese Funktion übergibt dort definierte Parameter wie die Szene geladen werden soll. Diese werden dann an die Funktion LoadScene() übergeben, welche aufgerufen wird sobald der Button gedrückt wird.

LoadScene() ruft den IEnumerator LoadSceneEnum() auf und übergibt die Parameter an diesen. Auch wird der Button StartLoadingButton ausgeblendet.

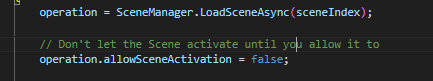
Dieser zusätzliche Aufwand ist nötig um einen einfacheren Aufruf der Funktion zu gewährleisten.



In der Funktion wird dann der Ladevorgang der neuen Szene gestartet. Die Parameter welche übergeben werden definieren ob

* + Die Szene asynchron geladen wird
    - Ein Lade Bildschirm angezeigt werden soll (Nur bei asynchronem laden möglich)
      * Ein Textfeld angezeigt werden soll, welches dem Spieler einige Tipps gibt.
      * Ein Ladebalken angezeigt werden soll? (Dieser zeigt den realen Ladefortschritt / 0.99f an um eine vollständige Füllung zu ermöglichen.
  + Die Szene durch den Namen einer Szene, durch den Index einer, oder ob einfach die nächste geladen werden soll. Falls keiner dieser Parameter den Wert „true“ hat wird die aktuelle Szene neu geladen. (Dies kann zu einer Endlosschleife führen!)

Der Szene wird beim asynchronen laden die Berechtigung genommen sich selbst nach dem Laden zu aktivieren.



Hierbei ist operation die derzeitige AsyncOperation, also der Ladevorgang.

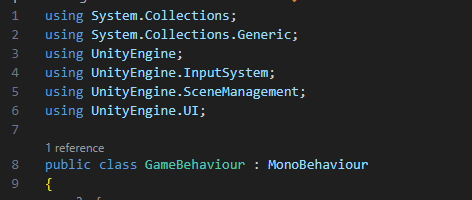
Nachdem der Ladevorgang bei dem asynchronem laden 90% erreicht hat, wird ein weiterer Button für den Spieler angezeigt. Dieser erlaubt bei Aktivierung der neuen Szene zu starten. Ansonsten würde der Spieler ewig in der Szene LoadingScene verweilen, wenn dieser nicht gedrückt würde.

Das gesamte Skript ist mit Fallbacks versehen. Dies bedeutet das falls ein Fehler auftritt dieser ausgegeben wird und behandelt wird um einen Absturz des Spiels zu verhindern, oder andere Auswirkungen auf dem Spieler, welche nicht erwünscht sind zu vermeiden.

## MainGame

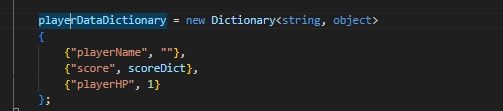
#### GameBehaviour.cs

* + Ist für das Speichern von Spieler-Daten zuständig
  + Behandelt die Parameter, welche das Verlieren und das Gewinnen des Spielers auslösen.



Diese Klasse nutzt das von Unity bereitgestellte neue Input System, sowie Funktionen für Szenen Management und UI (UserInterface). Zusätzlich ist die Klasse von MonoBehaviour abgeleitet.





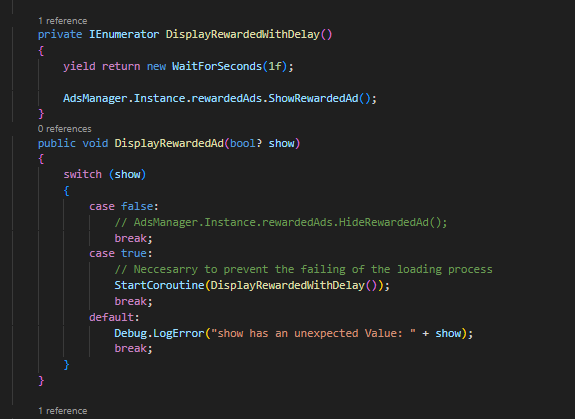
Der untere Screenshot zeigt einen Ausschnitt aus der Awake() Funktion, welche bei der Initialisierung der Szene aufgerufen wird. Dort werden die Spieler spezifischen Daten auf den Standard gesetzt.

Dort kann auch eine Verbindung zu Unity Cloud, einer Datenbank für Spielerdaten, hergestellt werden und damit einen vorherigen Speicherstand zu laden.

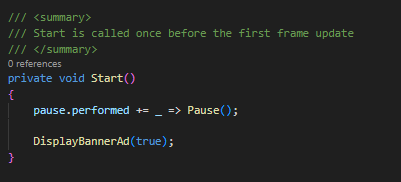
Diese Funktion ist jedoch derzeit deaktiviert.

Auch werden in diesem Skript die Funktionen für das Anzeigen von Werbungen implementiert.

Beispiel:



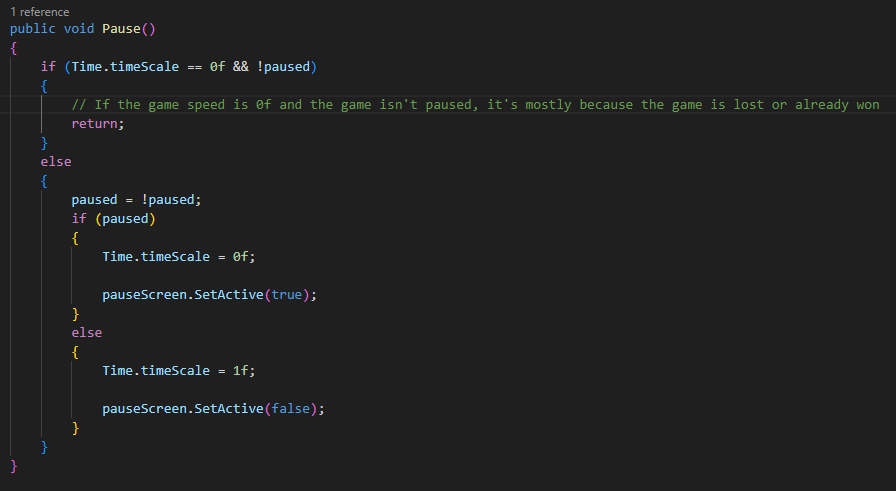
Ein sogenannter BannerAd wird angezeigt sobald die Szene MainGame geöffnet wurde.

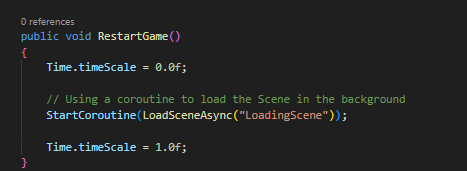


An diesem Screenshot ist ebenfalls erkennbar, dass sobald der Spieler durch das Auslösen der Pausenfunktion (zum Beispiel durch das drücken von der Escape-Taste), diese Funktion aufruft, welche das Spiel pausiert.

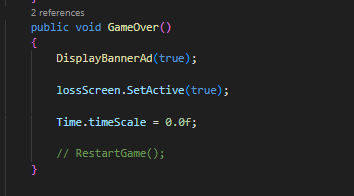
Dort wird zunächst überprüft ob das Spiel bereits verloren oder gewonnen ist um das unendliche Fortfahren durch den Spieler zu verhindern.

Falls dies nicht der Fall ist wird das Spiel angehalten und dem Spieler wird ein Bildschirm angezeigt, also ein Pausen-Bildschirm.





Hier wird die Funktion RestartGame() dargestellt. Diese wird aufgerufen um das Spiel neu zu starten. Zum Beispiel nach dem das Spiel verloren, oder gewonnen wurde.



Die Funktion GameOver() wird aufgerufen sobald der Spieler verloren hat. Dies geschieht, wenn der Spieler kein Leben mehr hat. Bei dieser Testversion des Spiels geschieht dies sobald „Mr. Musicman“ den Spieler erreicht.

Sobald die Funktion aufgerufen wird geschieht folgendes:

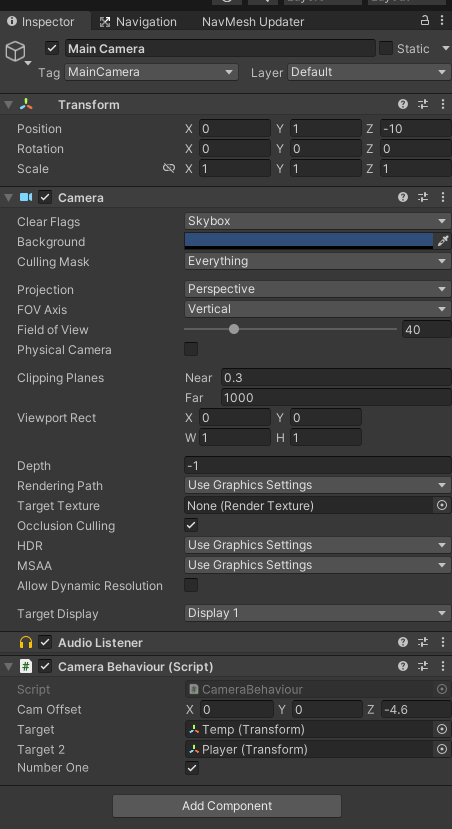
* + Ein BannerAd wird angezeigt
  + Der „lossScreen“ wird angezeigt
    - Dort hat der Spieler die Möglichkeit das Spiel neu zu starten durch das drücken eines Buttons.
  + Die In-Game-Zeit wird angehalten

Bei dem Gewinnen des Spiels durch das Einsammeln des Physik-Testes geschieht dasselbe, jedoch mit dem Unterschied, dass der „winScreen“ angezeigt wird.

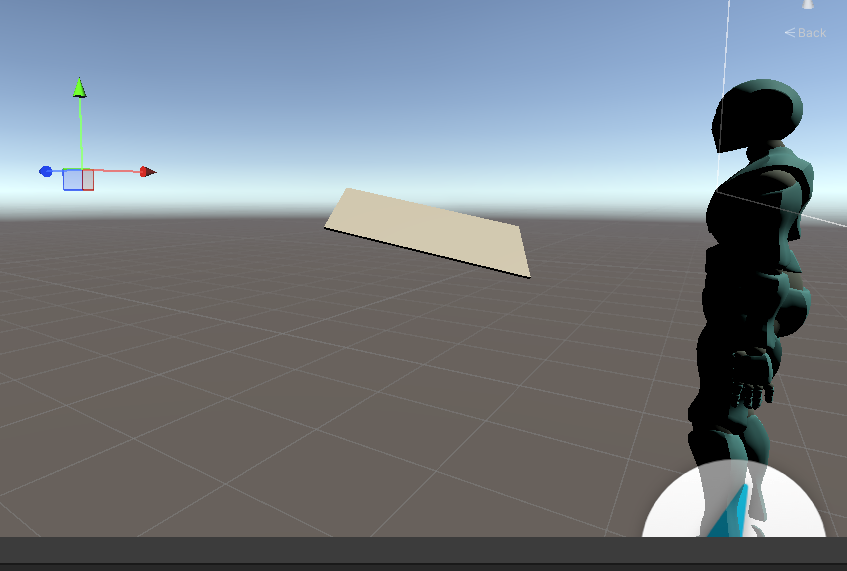
### CameraBehaviour.cs

* + Kontrolliert die imaginäre Kamera, welche in den Kopf des Spielers eingesetzt ist
    - Dies gewährleistet die First-Person-Experience





Hierbei guckt die Kamera das Gameobject Temp an. Dies befindet sich 4.6 Einheiten vor dem Kopf des Spielers. Dies gewährleistet den First-person-view.



Das Objekt Temp ist ohne physische imaginäre Visualisierung. (Man kann es nicht sehen).

Das Objekt Temp ist an der Stelle wo die 3 Pfeile sich Kreuzen.

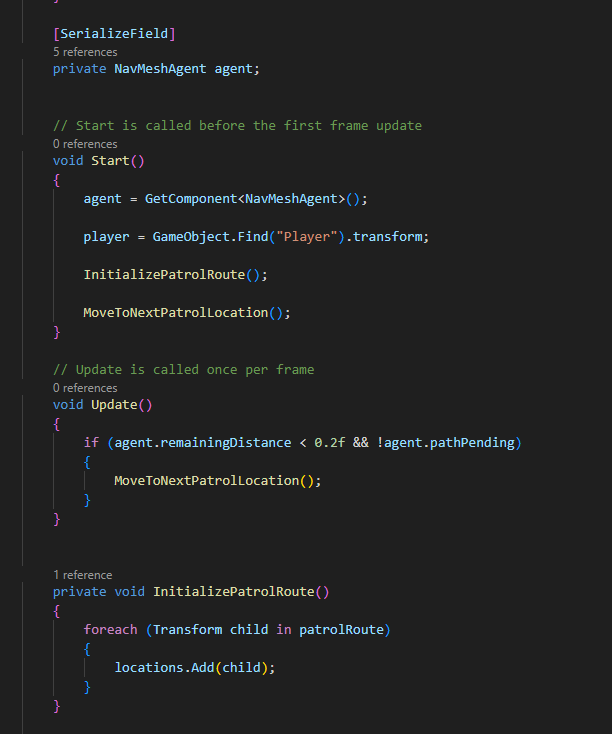
### MrMusicmanBehaviour.cs

* + Steuert wie der Charakter „Mr. Musicman“ sich verhält

Mr. Musicman wird hier erstmal der einfachhalbarkeit nur „Enemie“ genannt.

Enemie wird durch einen Pathfinding Algorithmus, welcher von Unity bereitgestellt wurde, gesteuert.

Dabei verfolgt Enemie zunächst eine Route zwischen 2 Punkten. Er Patrouliert auf dem Hauptflur. Sobald ein Spieler in der Nähe registriert wird, ist das neue und einzige Ziel das Enemie zu erreichen versucht der Spieler. Sobald dies geschieht ist das Spiel verloren.

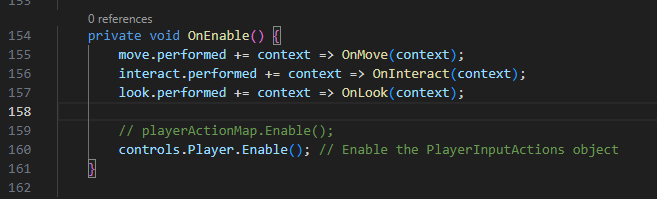


### PlayerBehaviour.cs

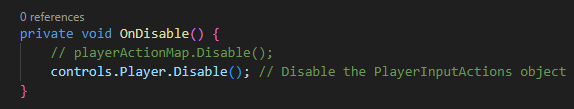
* + Steuert wie der Charakter des Spielers sich verhält
  + Die Bewegung des Spielers wird durch Animationen gesteuert, welche die Bewegung steuern
    - Die Animation werden gesteuert durch Variablen, welche in diesem Skript angesteuert werden:

Sobald der Spieler eine Taste drückt, welche in der Liste unter Steuerung enthalten ist wird ein Event aufgerufen. Dies geschieht durch Unity.

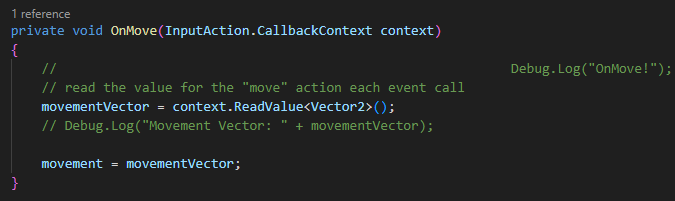
In diesem Skript werden für jede dieser Actions, spezifische Funktion aufgerufen. Damit dies geschieht wird bei der Initialisierung der Szene ein Lambda-Ausdruck verwendet, damit jedes Mal sobald das Event aufgerufen wird die Funktion ebenfalls aufgerufen wird. Man könnte dies als Verknüpfung bezeichnen. Zudem werden noch die Werte des Inputs durch den Spieler in der jeweiligen Form als Parameter weitergereicht. Zum Beispiel einen Vector2 (einen zweidimensionalen Vektor) für die Bewegung.



Auch wird in der OnEnable() Funktion die Eingabe durch den Spieler erlaubt. Die Eingabe wird dem Spieler verweigert, falls der Spielercharakter verschwinden sollte. Dies wird vermutlich nicht geschehen, jedoch kann in zukünftigen Versionen des Spiels mit zusätzlichen Extra-Leveln dies aus verschiedenen Gründen beabsichtigt passieren.



Sobald der Spieler durch Input versucht den Charakter zu bewegen (z.B. Linker Stick auf einem Gamepad, oder WASD) wird diese Funktion aufgerufen:

,

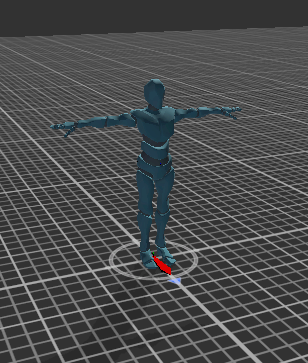
Diese nutzt den Vector2, welcher weitergereicht wurde und setzt die Lokale Variable des Typs Vector2 auf diese Werte.



Da der Charakter durch Animation gesteuert wird, welche durch Parameter gesteuert werden, ist dieser Vector2 dazu da, um die Bewegungs-Werte abzufragen beziehungsweise diese auf neue zu setzen.

Wenn der X-Wert des Vektors positiv ist bewegt sich der Charakter aus dessen Sicht nach vorne. Das heißt im Vergleich zu einem Menschen, in die Richtung in welche der Kopf übersteht. (Kinn). Dies wäre in Richtung des Blauen Pfeiles in der Abbildung.

Bei einem negativen X-Wert bewegt sich der Charakter noch hinten. Dies wäre in Richtung des Orangen Pfeiles in der Abbildung.

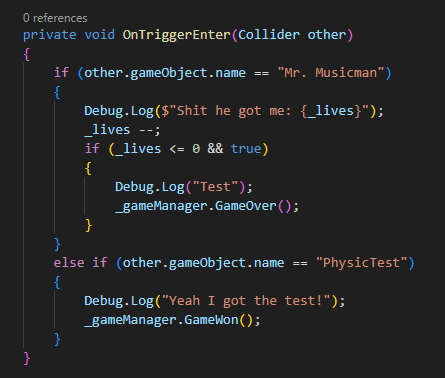




Bei dieser Grafik ist in Kreis beschrieben, auf welchem ein Roter Pfeil aufliegt. In Grün sind einige der Gradzahlen der Drehung aufgeschrieben. Je nachdem auf welche Gradzahl der Pfeil hinzeigen würde und somit auch der Charakter, ist die Gradzahl der Drehung des Charakters um die Y-Achse.

Bei einem positiven Y-Wert dreht sich der Charakter um die Y-Achse im Uhrzeigersinn (Gradzahl > 0°).

Bei einem negativen Y-Wert dreht sich der Charakter um die Y-Achse gegen den Uhrzeigersinn (Gradzahl < 0°).



Der obige Screenshot zeigt die Funktion OnTriggerEnter(). Dies ist eine von Unity bereitgestellte Funktion und wird ausgelöst wenn der Charakter ein anderes Objekt berührt.

Da jedes Objekt dies auslöst wird zuerst überprüft ob das Objekt „Mr. Musicman“ ist. Dies geschieht zuerst da keinem ein Physik Test nützt, wenn man in dem Chor dafür geht. Falls das andere Objekt „Mr. Musicman“ ist wird dem Spieler ein Leben abgezogen. Daraufhin wird überprüft ob der Spieler noch genug Leben hat, dass heißt mehr als eins. Falls der Spieler dies nicht hat ist das Spiel verloren und die Funktion GameOver() wird aufgerufen. Siehe GameBehaviour.cs.

Falls das Objekt nicht „Mr. Musicman“ ist und der Physik Test ist, dann ist das Spiel gewonnen und die Funktion GameWon() wird aufgerufen. Siehe GameBehaviour*.cs.*

# 6. Tools

## Unity

Unity ist eine leistungsstarke und vielseitige Spieleentwicklungsplattform, die Entwicklern die Erstellung interaktiver 2D- und 3D-Inhalte ermöglicht. Sie unterstützt eine Vielzahl von Plattformen (> 30) wie PCs, Konsolen, mobile Geräte, AR/VR-Systeme und WEBGL. Unity zeichnet sich durch eine benutzerfreundliche Oberfläche, umfassende Dokumentation und eine aktive Community aus. Es bietet einen Asset Store mit vorgefertigten Assets und nutzt C# als Programmiersprache. Jedoch sindd auch andere möglich. Neben der Spielentwicklung wird Unity für Simulationen, AR/VR-Anwendungen, Architekturvisualisierungen und interaktive Filme verwendet. Diese Flexibilität macht Unity zu einer bevorzugten Wahl für viele Entwickler und Kreative.

## Blender

Blender ist eine kostenlose und Multi-Format-Kompatible 3D-Grafiksoftware, die eine breite Palette von Funktionen für die Erstellung von 3D-Modellen, Animationen, Texturen und mehr bietet. Mit seinen leistungsstarken Werkzeugen ist Blender sowohl für Anfänger als auch für professionelle Künstler und Entwickler geeignet. Es ermöglicht die Modellierung von Objekten, das Rigging von Charakteren, das Animieren, das Texturieren, das Rendering und die Post-Produktion. Die Anwendungsbereiche von Blender sind vielfältig: von der Film- und Spieleindustrie über die Architekturvisualisierung bis hin zur wissenschaftlichen Visualisierung und Bildung. Kreative nutzen Blender für die Erstellung von 3D-animierten Filmen, Videospielen, Architekturvisualisierungen, digitalen Kunstwerken und vielem mehr. Durch seine benutzerfreundliche Oberfläche, die umfangreiche Dokumentation und die aktive Community ist Blender eine beliebte Wahl für viele, die hochwertige 3D-Inhalte erstellen möchten, ohne auf teure Softwarelizenzen zurückgreifen zu müssen.

# 7. Fazit

## Selbstreflexion und -einschätzung

Nach unserer Bewertung über unser Projekt würden wir dieses, in einem Notenbereich von 2 bis 2- einordnen.

Dies folgt aufgrund einiger Umstände:

Aufgrund eines Computervirus und der Tatsache, dass von den 3d Modellen kein Backup erstellt wurde und diese durch den Virus verloren gingen.

Ein weiterer Grund ist, die fehlende Komplexität, sowie die fehlende Erfüllung der persönlichen Anforderungen, sowie den Vorstellungen des Teams.