SimpleGame - Module

by Oliver Ziegler

Version 15

Beschreibung

Dieses Modul besteht aus einfachen, miteinander verbundenen Systemen für eine direkte oder indirekte Steuerung, Objekt-Interaktionen, Variablen und geskriptete Abläufe durch Sequenzen und Aktionen. Zusätzliche Systeme umfassen ein Inventar und Konversationen mit auswählbaren Optionen.

1. Steuerungsmethoden
2. Kameras
3. Objekt-Auswahl und -Interaktion
4. Action-Sequenzen
5. Variablen
6. Trigger
7. Inventar
8. Konversationen
9. Schießen
10. Fallen lassen
11. Timer

1. Steuerungsmethoden

Es kann zwischen einer direkten Steuerung via WASD und einer Maus-Steuerung entschieden werden.

Direkte Steuerung

Der notwendige Komponent für diese Steuerung heißt **MovementPerKeyboard** und kann an jedes beliebige GameObject angehängt werden.

Hauptsächlich benötigt es ein Objekt, den Charakter, den es bewegen kann. Dafür muss der Charakter über einen Rigidbody-Komponenten verfügen.

Mit den Parametern **MoveAcceleration** (Beschleunigung), **MaxSpeed** (Höchstgeschwindigkeit) und **RotationSpeed** (Drehgeschwindigkeit) wird die Bewegung des Charakters angepasst. Die Höchstgeschwindigkeit kann wird mit der Funktion **SetMaxSpeed** zur Laufzeit angepasst.

Die Bewegungsrichtung des Charakters kann auf unterschiedliche Art berechnet werden.

CharacterDirection

Die Tasten A und D drehen den Spieler, während er mit W und S in seiner eigenen Blickrichtung bewegt wird.

CameraDirection

Die Tasten WASD bewegen den Spieler relativ zur Kamera in die angegebene Richtung, wobei sich der Spieler automatisch in die entsprechende Richtung dreht und so immer vorwärts läuft.

Um mit Objekten zu interagieren muss unter **MainInteractionKey** die entsprechende Taste ausgewählt werden.

Der **JumpKey** gibt die Taste an, mit der ein Spieler springen kann. (None, wenn nicht springen)
Die **JumpForce** ist die Kraft mit der der Spieler nach oben befördert wird. (Mit der Masse des **Rigidbody** anpassen.)

Nach einem Sprung kann der Charakter erst nach Ablauf eines internen Timers erneut springen **(Time Until Jump Again).** Diese Funktion kann ausgeschalten und durch eine eigene ersetzt werden, beispielsweise, wenn ein Collider einen Untergrund trifft. Dazu muss die entsprechende Zeit auf 0 gesetzt und die Funktion „**CanJumpAgain**" im MovementPerKeyboard Komponenten aufgerufen werden.

Indirekte Steuerung

Der benötigte Komponent für eine Point&Click Steuerung heißt **MovementPerMouse** und kann an jedes beliebige GameObject angehängt werden. Um die Eingabe zu realisieren benötigt das Script ein GameObject **PositionObject**, dass an die Position verschoben wird, auf die die Maus im 3D-Raum gerade zeigt. Es kann natürlich auch ausgeschalten oder versteckt werden.

Diese Projektion der Maus in den 3D-Raum funktioniert nur über die **Main Camera**.

Klickt der Spieler auf einen Ort, wird dieser Input als Bewegungsauforderung an den Spieler weitergegeben. Dafür muss das Player-Objekt über einen NavMeshAgent verfügen, der hier eingetragen wird.

2. Kameras

Um die Funktionalität des Modules komplett zu nutzen sollte eines der drei Kamera-Scripte verwendet, also an das GameObject „MainCamera" angeheftet werden. In verschiedenen Szenen können unterschiedliche Kameras verwendet werden. Jede Kamera kann per Befehl zu einer statischen Position wechseln.

CamStatic

Eine statische Kamera die per Befehl zwischen zwei Punkten über einen Zeitraum wechseln kann (Siehe ActionCamSwitch).

Ist die Funktion **RotateToFollow** ausgewählt, rotiert diese Kamera, so dass sie das angegebene Objekt **FollowObject** im Fokus hält.

CamFollow

Eine Kamera, die dem Spieler folgt und dabei den definierten Abstand hält. Im Feld **Player** muss das Objekt spezifiziert werden, dem die Kamera folgen soll. Zudem sollte sie in den gewünschten Abstand, bzw. die gewünschte Drehung relativ zum Spieler gebracht werden. Dieses Offset wird sie dann im Spielverlauf halten.

Die Variable **Speed** beschreibt die relative Geschwindigkeit, mit der die Kamera dem Spieler folgt. Ist der Wert weit unter 1 wird die Bewegung der Kamera gedämpft. Somit können schnelle Bewegungen des Charakters ausgeglichen werden.

CamRotate

Diese Kamera kann vom Spieler mit der Maus gedreht werden. Dazu erzeugt sie ein automatisch ein CameraRig, dass dem Spieler, der in Player definiert wird, folgt. Die eigentliche Kamera hält dann den Abstand zum verfolgten Charakter.

Die Drehung um die Y-Achse ist unbegrenzt und erfolgt in der Geschwindigkeit **RotateSpeed**.
Die Drehung um die X-Achse, also nach oben und unten, erfolgt in der Geschwindigkeit **VerticalRotateSpeed** und wird durch die Angaben „**VerticalRotationLimits**“ begrenzt. Der X-Wert entspricht der minimalen, der Y-Wert der maximalen Drehung.

Ist **InvertVerticalMouseRotation** aktiviert, dreht sich die Kamera um die X-Achse bei der Mausbewegung in die entgegensetzte Richtung.

Normalerweise wird die Drehung durch die Rotation des CameraRigs erreicht. Dadurch dreht sich die Kamera automatisch um das „verfolgte“ Objekt.
Sollte dies nicht gewünscht sein, wie vor allem in **First Person Ansicht**, kann die Option „**RotateCamItself“** aktiviert werden. Dann wird die Kamera auf der Stelle gedreht.

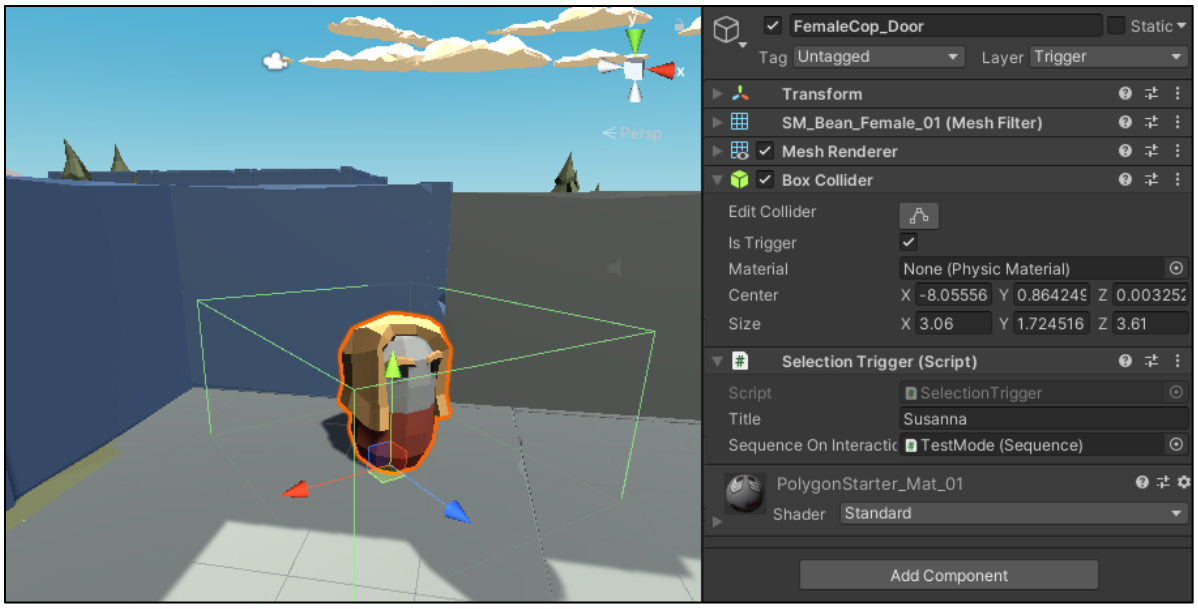
3. Objekt-Auswahl und -Interaktion

Beide Steuerungsmethoden haben entsprechende Komponenten mit denen Objekte in der Spielwelt ausgewählt werden können. Die Interaktion mit diesen Objekten erfolgt dann in der direkten Steuerung über den festgelegten **MainInteractionKey** und in der indirekten Steuerung über den linken Maus-Klick. In beiden Fällen wird die im Komponenten gespeicherte Sequenz **sequenceOnInteraction** gestartet (siehe ActionSequenzen)

Diese Auswahl-Komponenten heißen:

Direkte Steuerung: **SelectionTrigger**

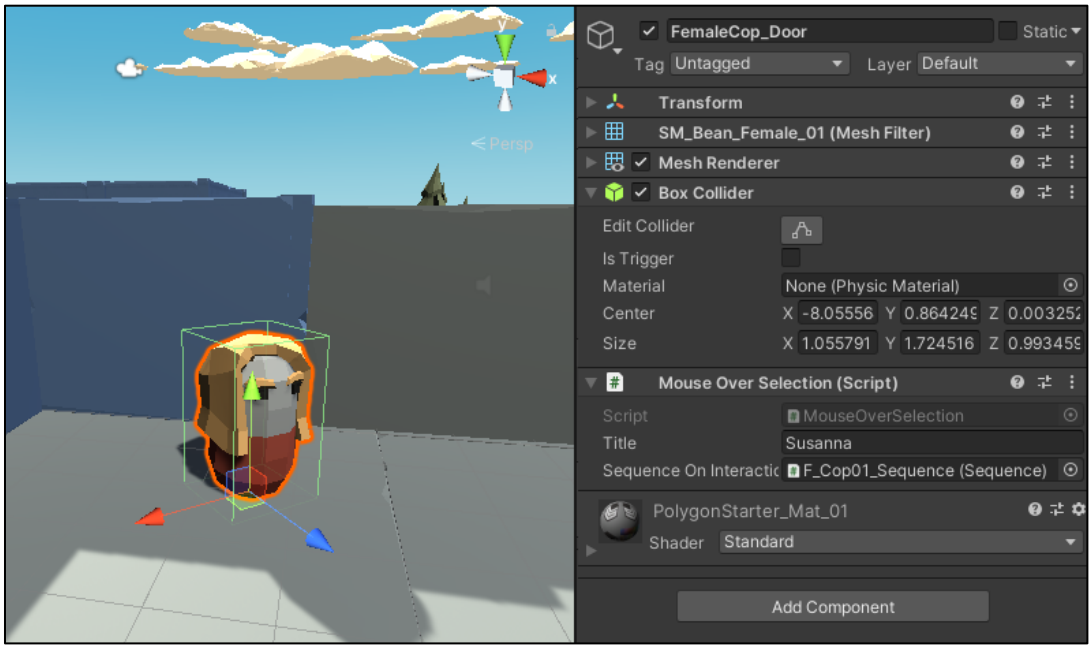
Dieser Komponent muss an dem zu erfassenden Objekt angebracht werden. Das GameObject benötigt ausserdem einen Collider der als „Trigger“ eingestellt ist. Läuft der Charakter in diesen Collider wird das Objekt ausgewählt und mit ihm kann interagiert werden. Verlässt der Charakter den Trigger ist dies nicht mehr möglich.



Indirekte Steuerung: **MouseOverSelection**

Dieser Komponent muss an dem zu erfassenden Objekt angebracht werden. Das GameObject benötigt außerdem einen Collider der NICHT als „Trigger“ eingestellt ist. Findet der **MovementPerMouse** – Komponent ein Objekt mit einem solchen Komponenten, wird das entsprechende Objekt ausgewählt. Ein Klick schickt dann den Spieler-Charakter nicht zu diesem Punkt, sondern löst die im Komponenten festgelegte Sequenz aus.

(Der Charakter läuft nicht zum Objekt, dass sollte in der Sequenz ggf. per **ActionWalkTo** eingerichtet werden.)

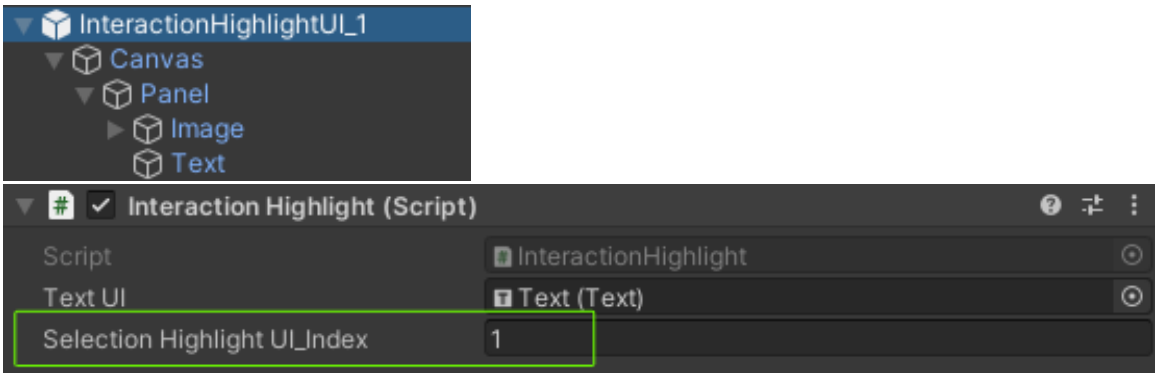


Tooltip zum ausgewählten Objekt

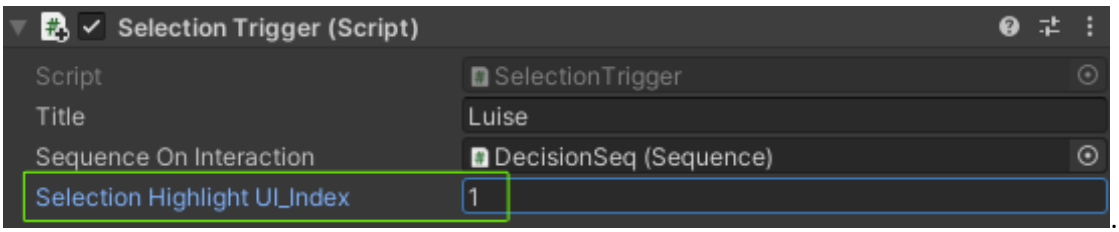
Um anzuzeigen, dass ein Objekt ausgewählt wurde gibt es eine vorgefertigte Interface-Lösung. Das Prefab **SelectionHighlightUI** wird automatisch im System registriert, wenn es sich in der Szene befindet. Wählt der Spieler jetzt ein Objekt aus (egal ob direkt oder indirekt), wird dieses UI-Canvas zum entsprechenden Objekt verschoben und zeigt den Namen des Objektes an. Das UI kann wie gewünscht verändert werden, beispielsweise um den Interaktions - Button anzuzeigen.

Unterschiedliche Tooltips/UIs

Es können auch **unterschiedliche UI**s dieser Art für **unterschiedliche Objekte** verwendet werden. Zu diesem Zweck das UI-Element eine Variable namens **`SelectionHighlightUI_Index.`** Es wird intern unter dieser Zahl gespeichert.



Um nun Festzulegen welches UI-Element ein selektierbares Objekt verwenden soll, kann im Komponenten von **selektierbaren Objekte** (wie SelectionTrigger und MouseOverSelection – Komponenten) ebenfalls ein **`SelectionHighlightUI_Index`** angegeben werden. Beim Auswählen dieses Objekts erscheint dann das UI, dass mit derselben Zahl gekennzeichnet wurde.



4. Action Sequenzen

Das Herzstück des Moduls ist eine Reihe von Actions, die Standard-Aktionen für geskriptete Abläufe in Spielen umsetzen sollen. Sie sind beliebig erweiterbar und, außer ihrer gemeinsamen Basis-Klasse, wenig besonders.

Das Modul fasst diese Actions jedoch zu Sequenzen zusammen, die dann nach einander abgearbeitet werden. Durch die Abarbeitung in dieser Sequenz können beispielsweise Verzögerungen im Ablauf eingebaut werden, um die Inszenierung des Spiels besser steuern zu können. Actions können zudem weitere Sequenzen aufrufen und dafür auch bestimmte Bedingungen festlegen.

Zur Festlegung und Abarbeitung der Sequenzen wird das GameObject – Komponenten System von Unity benutzt. Jedes GameObject kann mit dem Komponenten **Sequence** zu einer Sequenz gemacht werden. Jedes Objekt sollte nur einen dieser Komponenten enthalten.

Die in der Sequenz enthaltenen Aktionen werden ebenfalls als Komponenten an dasselbe GameObject angefügt. Ihre Parameter und ihre Reihenfolge im Inspector des GameObjects entsprechen der Abarbeitung zur Laufzeit, wenn die Sequenz gestartet wird.

Notwendige Vorbereitungen:

Damit das System funktioniert muss sich das Prefab **SimpleGameEngine** in der Szene befinden. Es wird automatisch so vorbereitet, dass es beim Wechsel der Szene zur Laufzeit nicht gelöscht wird.

Befindet sich in der nächsten Szene ein weiteres Prefab dieser Art, wird dieses neue Prefab beim Laden der Szene entfernt.

Actions allgemein

Actions können auch ohne Sequenzen immer über die Funktion **„ExecuteAction“** aufgerufen. Eine Nutzung beispielsweise durch das Event-System ist also möglich. Empfohlen wird aber das Anlegen und Nutzen von Sequenzen, auch für kleine Events.

Eigene Actions

Um eigene Actions anzulegen wurde das Script **ActionTemplate** hinzugefügt.
Dieses kann in ein neues Script kopiert werden. Ändern sie dann im Code den Namen „ActionTemplate“ zum Namen ihres Scripts.

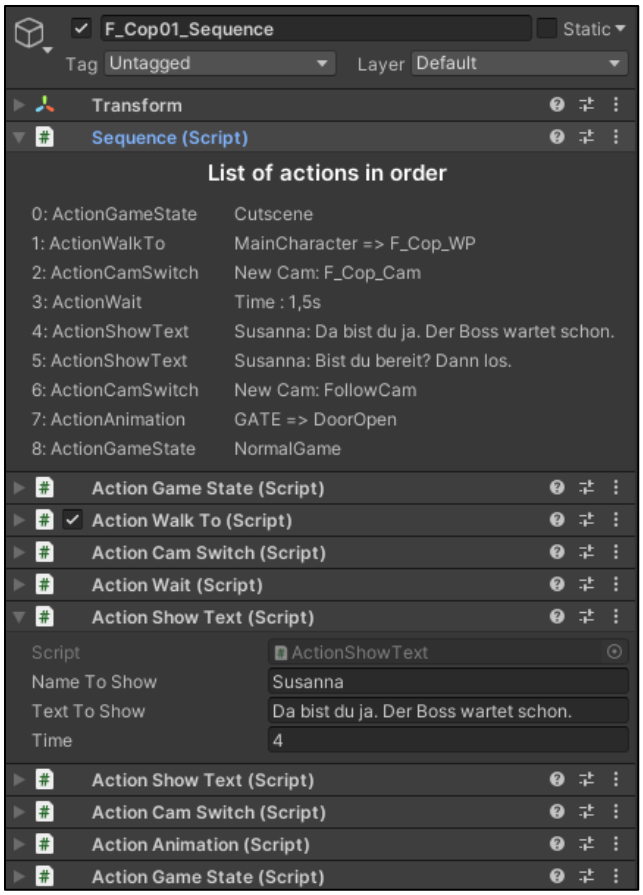
Sequence

Eine Sequenz sammelt automatisch alle Komponenten (die eine Action sind) des eigenen GameObjects in einer Liste und zeigt diese an. Die Reihenfolge der Actions im Inspector des GameObjects ist dabei auch die Reihenfolge der Abarbeitung der Sequenz (sofern nicht zwischendurch eine andere Sequenz gestartet wird)

Eine Sequenz kann über „allowedGameState“ auf einen einzelnen GameState fixiert werden. Liegt ein anderer GameState zum Zeitpunkt der Ausführung an, wird die Sequenz nicht abgespielt. Dies ist

hilfreich um Steuerungen oder Funktionen des normalen Spiels während einer Cutscene oder in einer Konversation zu ignorieren, ohne eine Abfrage tätigen zu müssen.

Um eine Sequenz durch ein Script oder einen Event zu starten, muss die Funktion **ExecuteCompleteSequence** aufgerufen werden. Die Sequenz wird dann an den SequenceHandler übergeben, der für die Abarbeitung zuständig ist.



Actions

(Weitere Aktionen in den Teil-Modulen **Inventar** und **Konversationen**)

ActionWait

Die Ausführung der Sequenz wird für **timeInSeconds** Sekunden unterbrochen.

ActionSetActivity

Das angegebene Objekt **Obj** wird dem Parameter **State** entsprechend aus- oder angeschalten.

ActionSound

Spielt den angegebenen AudioClip **Clip** mit der eingetragenen Lautstärke **Volume** ab. Wenn eingetragen, kann **waitForSeconds** gewartet werden, bis die Sequenz weiter ausgeführt wird.

ActionStartSequence

Startet die angegebene Sequenz **sequenceToStart**. Die aktuelle Sequenz wird unterbrochen, bis die neue abgeschlossen ist.

ActionSwitchScene

Wechselt zu der per Name **SceneName** angegebenen Szene. Stellen Sie sicher, dass der Name korrekt und die entsprechende Szene in den BuildSettings eingetragen ist.

ActionAnimation (Legacy Animation System)

Wenn der **Typ** „Play“ eingestellt ist, startet diese Action eine Animation namens **AnimationName** im angegebenen **AnimationComponent**. Der **WaitUntilEnded** Parameter pausiert gegebenenfalls die weitere Abarbeitung der Sequenz bis die Animation abgeschlossen ist.

Der **Typ** „Stop“ stoppt die Animation des angegebenen **AnimationComponent**.

ActionAnimatorParameter (Mechanim Animation System)

Verändert den Parameter **parameterName** im Animation Controller des Animators des angegebenen Objekts **AnimatedObject**. Der Name und der ausgewählte Parameter-Typ **Type** müssen exakt mit einem Parameter im Animation Controller übereinstimmen.

Für die Typen Float, Int (Integer) und Bool werden die Werte übernommen, die im Feld **Value** eingetragen werden. Ein Trigger kann nur „gesetzt“ werden und deaktiviert sich von allein.

ActionCamSwitch

(Damit diese Action funktioniert, muss die Haupt-Kamera des Spiels über ein **CamFollow** oder **CamStatic** Komponent verfügen, in dem ein Objekt eingetragen ist, dem sie folgen kann.)

Der **Typ** „Static Cam“ sorgt für eine Bewegung der Haupt-Kamera zu dem im **CamTarget** angegebenen Transform innerhalb von **time** Sekunden. Ist der Wechsel abgeschlossen, bewegt sich die Kamera nicht mehr.

Der **Typ** „BackToFollow“ sorgt für eine Bewegung der Hauptkamera zu ihrem angestammten Platz relativ zu ihrem „gefolgten“ Objekt innerhalb von **time** Sekunden. Die Kamera verfolgt dann wieder das Objekt/den Spieler.

ActionLayer

Verändert die von der Kamera sichtbaren Ebenen (CullingMask). Der Parameter **VisibleLayer** wird dabei in der Kamera komplett übernommen.

ActionGameState

Wechselt den internen **GameState**, wie angegebenen.

Im GameState *Cutscene* und *Conversation* sollten keine Bewegungen mehr an den Spieler weitergegeben und keine Interaktionen mehr möglich sein.

CamRotate:

Im NormalGame GameState ist die Maus dann in der Mitte des Bildschirms gelockt und versteckt.

Im Cutscene GameState ist die Maus gelockt und Eingaben werden nicht angenommen.

Im Conversation GameState werden keine Bewegungseingaben angenommen, die Maus ist aber freigegeben, um die Interaktion in der Konversation zu ermöglichen.

ActionShowText

Notwendig: UI-TextPanel mit TextPanel-Component. (Vorgefertigt im Prefab TextPanelCanvas)

Zeigt den in **TextToShow** angegebenen Text im TextPanel für **time** Sekunden an. Ist ein Name im **NameToShow** Parameter eingetragen, wird dieser ebenfalls in einem zusätzlichen UI-Text angezeigt.

ActionMaterial

Weist dem angegebenen MeshRenderer **rendererToChange** ein anderes Material **newMaterial** zu.

ActionSkybox

Weist den allgemeinen Szenen-Einstellungen (Render/Lighting Settings) eine anderes Material für die Skybox **skybox** zu.

ActionWalkTo

Lässt den angegebenen **Character** (benötigt MovementIndirect-Component und NavMeshAgent) zum **Target** gehen. Beide sollten dafür auf dem NavMesh liegen.
Der Parameter **WaitForArrival** bewirkt, dass die weitere Abarbeitung der Sequenz unterbrochen wird, bis der Charakter näher als **ArrivalDistance** am Target-Objekt angekommen ist.

ActionLookAt

Dreht das angegebene Objekt **Character** innerhalb von **time** Sekunden so, dass es das angegebene Objekt **Target** anschaut. Die Drehung beschränkt sich dabei auf die Y-Achse.
Ist **waitForCompletion** eingeschalten, wird die Sequenz angehalten, bis die Drehung abgeschlossen ist.

ActionTeleport

Teleportiert das Objekt objToTeleport an den teleportPoint. Ist die Option copyRotation aktiv, wird das zu verschiebene Objekt an den Punkt verschoben und übernimmt die y-Rotation des Teleportpunkts.

ActionSceneLighting

Verändert diverse Lighting Parameter, die für diese Szene festgelegt werden können (Unter Window > Rendering > Lighting Settings)

Der Button „Fetch current scene light settings“ übernimmt die aktuellen Einstellungen aus den Lighting Settings. Diese können dann leichter in den betreffenden Punkten angepasst werden.

ActionSkybox

Verändert das Material für die Skybox. Sonst festgelegt unter Window > Rendering > Lighting Settings

(Enthalten in ActionSceneLighting)

ActionNavigationArea

Ändert die NavMeshAreas auf dem der zugewiesene Charakter (NavMeshAgent) **navAgent** laufen kann. Die angegebene **navArea** wird entsprechend dem Parameter **walkable** begehbar (true) oder nicht begehbar (false).

ActionMusic

Wechselt die Musik zum Clip **clip** und benötigt für das Überblenden **timeToFadeIn** Sekunden.
Volume gibt die Lautstärke an, mit der die Musik schlussendlich spielen soll (0-1).

Benötigt den MusicManager und zwei zugewiesene AudioSources. (Im SimpleGameEngine Prefab enthalten)

ActionVariable

(Anlegen von Variablen siehe **4. Variablen**)

Verändert den Wert der im DropDown Menü ausgewählt **Variable**.

Der **Typ** „Change“ sorgt für eine Veränderung der Variable um den Wert **value**.

Der **Typ** „Explicit“ setzt den Wert der angegebenen Variable direkt auf den Wert **value**.

ActionOnCondition

(Anlegen von Variablen siehe **4. Variablen**)

Vergleicht die angegebene Variable durch die Operation mit dem Vergleichswert – Resultat ist ein Wahrheitswert, also ***true***, wenn die Bedingung erfüllt ist oder ***false***, wenn sie es nicht ist.

Diese Action startet dann je nach Resultat die angegebene Sequenz, falls im entsprechenden Slot, eine Sequenz vorhanden ist.

ActionOnMultiConditions

(Anlegen von Variablen siehe **4. Variablen**)

Vergleicht die angegebenen Variablen mit den Vergleichswerten. Sind alle Vergleiche zutreffend TRUE, startet die Action die angegebene Sequenz, falls im entsprechenden Slot, eine Sequenz vorhanden ist. Falls nur eine abweicht, wird die Sequenz für FALSE ausgeführt, sofern diese vorhanden ist.

ActionTimer

Startet oder Stoppt einen Countdown an dessen Ende eine Sequenz abgespielt wird.

Start:

Startet einen Timer names **TimerName** und führt die Sequenz **Sequence** aus, wenn die Zeit **TimeToRun** (Sekunden) abgelaufen ist. Der Name **TimerName** wird benötigt, weil mehrere Timer gleichzeitig laufen können und es nur so möglich ist einen speziellen Timer zu stoppen.

StartRandom

Startet einen Timer names **TimerName** und führt die Sequenz **Sequence** aus, wenn die Zeit ist. Die Zeit wird zufällig zwischen den Grenzen „Minimum time“ und „Maximum Time“ festgelegt. (Bei jedem Aufruf neu.)

Stop:

Stoppt den in **TimerName** angegebenen Timer.

(Anzeigen von Timern im UI, siehe 11. Timer.)

ActionFollow

Ermöglicht es einem Charakter zu befehlen einem Objekt zu folgen. Dafür benötigt die Szene ein NavMesh und der Charakter sowohl einen NavMeshAgent als auch ein MovementIndirect Komponenten.

Start:

Gibt dem GameObject mit dem Komponenten MomvementIndirect (Character) den Auftrag permanent den Weg zu ObjectToFollow zu aktualisieren und zu ihm zu gehen.

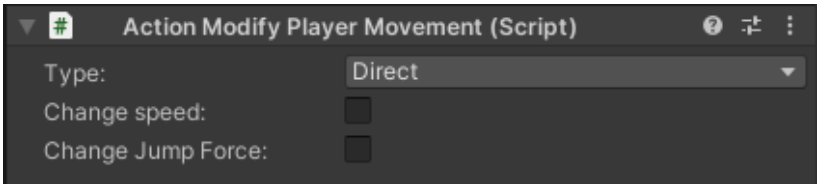
Stop:

Beendet das Verfolgen durch den angegebenen Charakter.

ActionModifyPlayerMovement

Ermöglicht das Verändern von Bewegungs-Eigenschaften des Spielers oder eines NPCs.
Unterschieden wird dabei nach Eingabe-Art (Type).
Veränderbare Eigenschaften sind der Speed und die Jump Force.

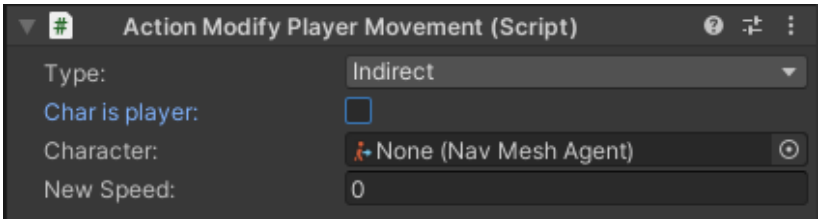
Type – Direct:



Direkte Spielerbewegung via WASD.
Eine Auswahl des Charakters entfällt, weil immer der Spielercharakter angesprochen wird.
Um eine Eigenschaft zu verändern, muss zunächst die jeweilige Option aktiviert werden. Ist die Option nicht aktiviert, bleibt der Wert unverändert.

Der Speed beschreibt die maximale Bewegungsgeschwindigkeit des Charakters.
Die Jump Force die Kraft, mit der der Spieler springt.

Type – Indirekt:



Indirekte Spielerbewegung durch Klicken auf einen Ziel-Punkt, oder ein NPC.

Ist „Character is Player“ aktivert, werden die Eigenschaften des SpielerCharakters verändert. Wenn nicht, muss ein NPC mit einem „IndirectMovement“ – Componenten festgelegt werden.

Um eine Eigenschaft zu verändern, muss zunächst die jeweilige Option aktiviert werden. Ist die Option nicht aktiviert, bleibt der Wert unverändert.

Der Speed beschreibt die maximale Bewegungsgeschwindigkeit des Charakters.

ActionTransformParent

Ermöglicht das Verändern des Parent-Objekts eines GameObjects. Das GameObject „objToChange“ wir

ActionCamFollow

Realisiert das Wechseln einer statischen Kamera zwischen „Follow“ und „Static“.

Follow bedeutet hierbei, dass die Kamera sich zu dem angegebenen Objekt dreht, nicht aber ihre Position verändert.

ActionOverlayFade

[Diese Action benötigt ein vorbereitetes Objekt mit dem „OverlayFade“ Script. Empfohlen wird das „OverlayFade“-Prefab im Prefabs-Ordner.]

Mit dieser Action kann das (in dieser Szene einzigartige) UI-Overlay ein- bzw. ausgefadet werden.

Die angegebene „time“ gibt dabei die Zeit an, bis der jeweilige Zustand erreicht ist.

Ist „Wait Until Finish“ gesetzt, wird die Sequence erst weiter ausgeführt, wenn das Fading abgeschlossen ist.

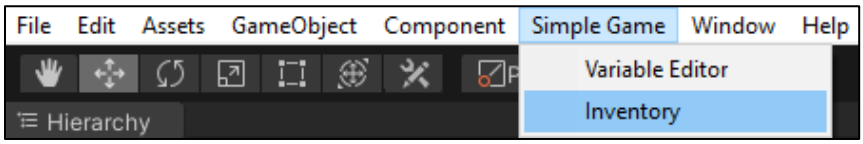
5. Variablen

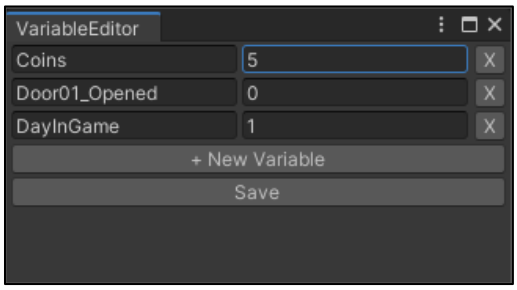
Damit das System funktioniert muss sich das Prefab **SimpleGameEngine** in der Szene befinden. Es wird automatisch so vorbereitet, dass es beim Wechsel der Szene zur Laufzeit nicht gelöscht wird.

Befindet sich in der nächsten Szene ein weiteres Prefab dieser Art, wird dieses neue Prefab beim Laden der Szene entfernt.

Um Entscheidungen anhand von Variablen treffen, bzw. die Variablen im Spielverlauf ändern zu können, müssen diese zunächst festgelegt werden. Das System basiert auf generischen Integer-Variablen, also ganzen Zahlen, denen ein eindeutiger Name zugewiesen werden muss.

Dies geschieht im **Variable Editor** unter **Game Variables** in der Menü Leiste.





Durch den Button „New Variable“ wird eine neue Variable erzeugt, der ein eindeutiger Name und ein Grundwert gegeben werden muss. Mit diesem Wert wird das Spiel starten.

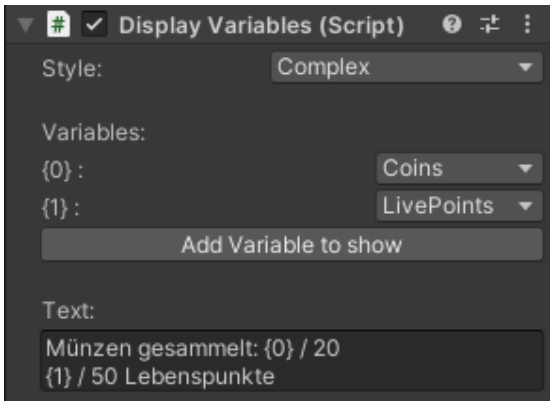
Um die Veränderung definitiv zu Speichern, sollte der „Save“-Button benutzt werden.

Anzeige von Variablen im UI

Derzeit kann eine Variable über zwei Wege dargestellt werden. Als **Text** erfolgt die Darstellung über das Script **DisplayVariables**. Hier kann zwischen einer einfachen (Simple) und komplexeren (Komplex) Methode gewählt werden.

Simple: Stellt den Inhalt der Variable nach einem Präfix **prefix** dar. Es muss nur der Variablenname ausgewählt werden.

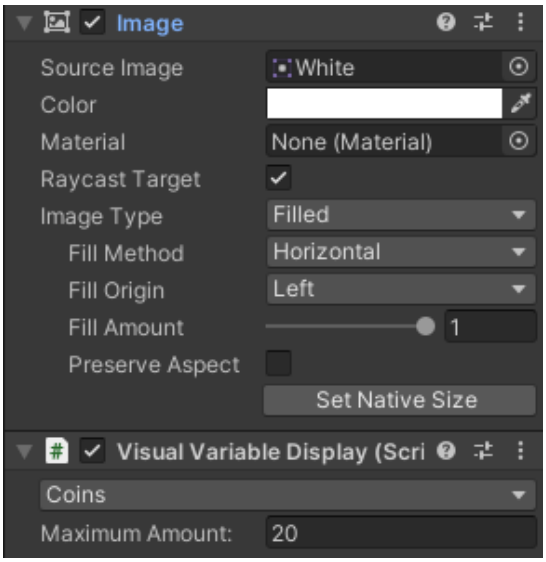
Komplex: Der Inhalt mehrerer Variablen kann innerhalb eines Textes angezeigt werden. Hierzu müssen/können mehrere Variablen einer Liste hinzugefügt werden. Ihre Nummer (beginnend bei 0) ist dabei sehr wichtig. Der Wert der Variablen wird mit dem Schlüssel **{x}** in den Text integriert. Wobei x für die Nummer der Variablen steht.



Die zweite Möglichkeit ist die Darstellung in einer **Grafik**.

Hierfür wird zunächst ein UI-**Image** Komponent benötigt, der als Type „**Filled**“ eingestellt ist. Die Variable **FillAmount** beschreibt zu wieviel Prozent die Grafik (in der angegebenen Richtung) zusehen ist (0-1)

Das Script **VisualVariableDisplay** teilt den Wert der angegebenen Variable durch den festgelegten Höchstwert **MaximumAmount**. Das Ergebnis wird im **FillAmount** des Image-Komponenten gespeichert.



Eigene Anzeigen für variablen

Der Wert einer Variablen kann in einem Script über den Befehl

```
int value = VariableManager.Instance.GetVariable(variableName);
```

abgefragt und gespeichert werden.

Die beste Methode wäre es, das neue Script vom Script **DisplayVariables** abzuleiten. Dadurch wird automatische die Funktion **AdjustUI** beim **VariablenManager** angemeldet. Ändern sich die Variablen im Manager, wird die Funktion aufgerufen und die Anzeige kann angepasst werden.

6. Trigger und MainEvents

Drei einfache, integrierte TriggerScripts ermöglichen das direkte Starten von Sequenzen beim Betreten (Enter) oder Verlassen (Exit) eines Triggers.

Der Collider muss zwingend als „Trigger“ eingestellt sein.

SequenceTriggerEvent

Reagiert auf jedes Objekt (das einen Rigidbody hat) und in diesen Trigger hinein oder heraus läuft. Startet dann die entsprechenden Sequenzen (falls vorhanden).

LayerTrigger

Reagiert auf jedes Objekt (das einen Rigidbody hat), in dem angegebenen Layer ist und in diesen Trigger hinein oder heraus läuft. Startet dann die entsprechenden Sequenzen (falls vorhanden).

Object Trigger

Reagiert ausschließlich auf das Objekt, das in **neededObject** eingetragen ist (einen Rigidbody hat) und in diesen Trigger hinein oder heraus läuft. Startet dann die entsprechenden Sequenzen (falls vorhanden).

OnSceneLoad_StartSequence

Startet die angegebene Sequenz, wenn die Szene gestartet wird.

7. Inventar

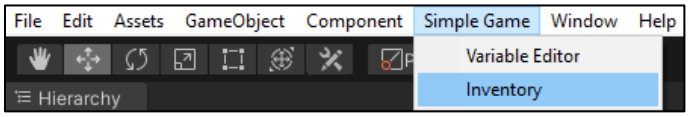
Das Inventarsystem teilt sich in 4 Bereiche:

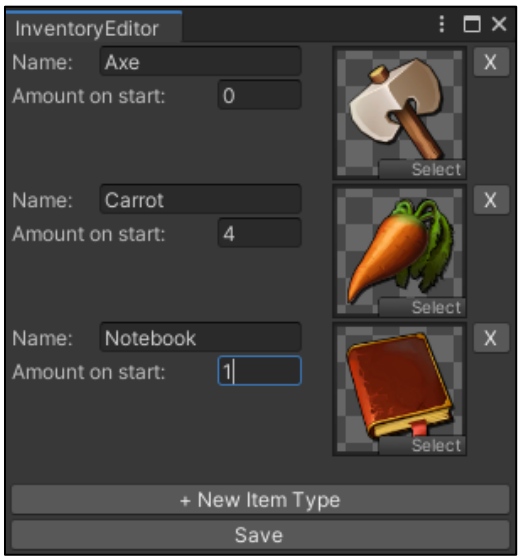
- Definition von Gegenständen
- Nutzung
- Anzeige
- Kombination

Definition

Um Inventargegenstände im Spiel erhalten zu können, müssen diese zunächst im Editor festgelegt werden. Der entsprechende Editor befindet sie in der Menü Leiste unter „SimpleGame“ > „Inventory“.

Mittels „New Item Type“-Button wird eine neue Art von Inventar Objekt angelegt. Danach müssen ein eindeutiger Name, ein Bild (Sprite) und eine Anzahl zu Spielbeginn bestimmt werden. Sicherheitshalber sollte mit dem „Save“-Button gespeichert werden.





Nutzung

Damit die Inventargegenstände aufgenommen werden können, muss sich das InventoryManager Script in der Szene befinden. Es ist automatisch am **SimpleGameEngine** - Prefab angeheftet.

Das Aufheben, Entfernen und Abfragen von Zuständen der Inventargegenstände erfolgt über zwei Actions:

ActionInventory

Der Typ **ActionType** beschreibt, welche Aktion mit dem Inventarobjekt geschehen soll, dass unter **InventoryItem** ausgewählt wurde, der Zahl-Wert dahinter die zu behandelnde Menge.

Add:	Fügt die Anzahl dem Inventar hinzu.
Sub:	Entfernt die Anzahl aus dem Inventar (Minimalwert 0)
SetExplicit:	Setzt die Anzahl direkt auf den angegebenen Wert.

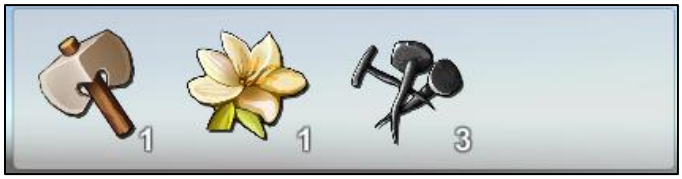
ActionCheckInventory

Fragt ab, ob sich eine bestimmte Anzahl eines Inventarobjekt-Typs im Besitz des Spielers befindet. (Gleich, Größer, Kleiner, ...) Ähnlich einer Variablen-Abfrage können zwei Sequenzen für eine Bestätigung (true) oder Ablehnung (false) dieser Abfrage eingestellt werden.

Anzeige

Um das Inventar des Spielers anzuzeigen existiert im Paket ein Prefab namens **InventoryUI**. Es stellt automatisch alle vorhandenen Objekte (Anzahl > 0) als Bild mit ihrer Anzahl in einer Liste dar und wird bei jeder Veränderung automatisch aktualisiert.

(Bisher stellt das Inventar diese Gegenstände immer in horizontaler Richtung dar. Abgesehen davon kann es beliebig angepasst werden.)



Die Standard-Anzeige des Inventars erfolgt horizontal. Die Liste der Objekte wird dabei in ihrer Breite den enthaltenen Objekten angepasst, um ein korrektes Scrollen zu gewährleisten.

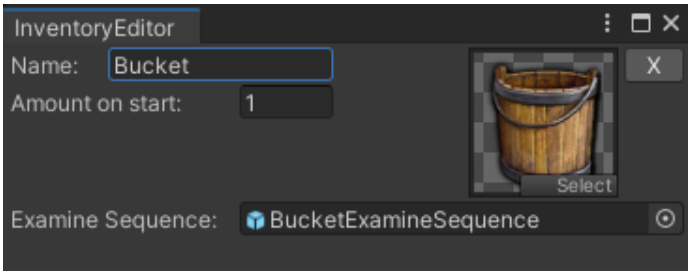
Diese Funktion kann auf Vertical umgestellt oder komplett deaktiviert werden. Dazu sollte jedoch auch das automatische LayoutScript am Objekt List geändert werden. Hier vorgeschlagene Zusammenstellungen:

	InventoryUI	List – Objekt
Horizonal	Horizontal	HorizontalLayoutGroup
Vertikal	Vertical	VerticalLayoutGroup
Gitter	None	GridLayoutGroup

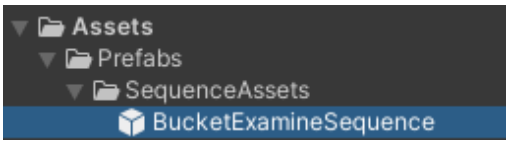
Ein Layout mit festen Slots wird derzeit nicht unterstützt.

Untersuchen von Objekten

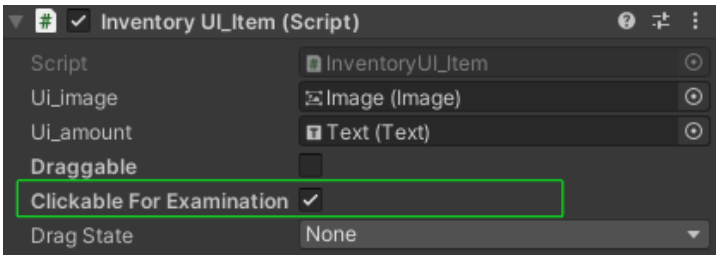
Inventar-Gegenstände können auch mit einer Sequenz versehen werden, die dann beim Klicken auf das Objekt gestartet wird.



Wichtig:
Die Sequenz eines Inventar-Objekts muss im Editor festgelegt werden und kann deswegen nicht einer speziellen Szene zugeordnet sein. Die Sequenz muss sich stattdessen an einem Prefab im Projekt-Ordner befinden. Die Zuweisungen in diesem Prefab können ebenfalls keine Verbindungen zu Objekten innerhalb einer Szene haben.



Um Die Sequenz eines Inventar-Objekts im UI auszuführen, muss die „Clickable For Examination“ – Options im InvUI_Item aktiviert sein.

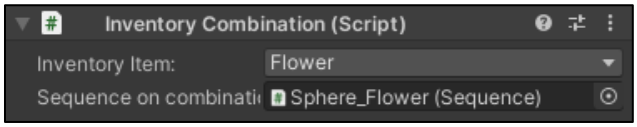


Am besten nutzen Sie die Sequenzen eines Inventar-Objekts nur um Informationen zu geben und Variablen zu ändern. Für die Veränderung von Szenen-Objekten oder starten von Cutscenes im Zusammenhang mit Inventar-Objekten nutzen Sie am besten Kombinationen.

Kombination

Für die indirekte Steuerung ist zusätzlich eine Kombination von Inventargegenständen mit Szenenobjekten möglich. Dazu kann ein Inventarobjekt mittels Drag & Drop aus dem InventoryUI gezogen werden. Die Auswahl der Szenen-Objekten mit der Maus (MouseOverSelection) erfolgt weiterhin, beendet man den Drag während das Objekt ausgewählt ist, wird auf eine Kombination geprüft. Damit eine Kombination eines Inventarobjekts mit dem Szenenobjekt erfolgreich ist, muss diese Kombination am Szenenobjekt durch den Komponenten **InventoryCombination** definiert werden.

Im **DropDown** Menü des Komponenten wird das entsprechende Inventarobjekt ausgewählt und in das Feld **Sequence** eine Sequenz zugewiesen, die bei einer Kombination mit diesem Objekt ausgeführt wird.



WICHTIG:

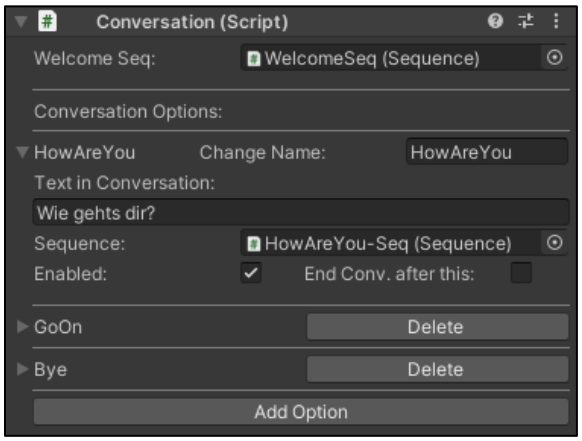
Achten sie darauf, dass für das Inventar-UI-Element die Einstellung „Raycast Target“ nur am obersten Parent-Objekt dieses Inventar-Objekts aktiviert ist. Sind andere UI-Elemente „Im Weg“, in dem sie als „Raycast Targat“ definiert sind, erreicht der Strahl der Maus nicht mehr das auswählbare Objekt in der Szene. Wird das Inventar-Objekt nun losgelassen, wird es zurück ins Inventar gelegt und nicht kombiniert.

Das Oberste UI-Element des InventarItems sollte demnach selbst ein UI-Image sein. Die Einstellung „Raycast Target“ muss aktiv sein, damit der Spieler es auswählen kann. Es wird deaktiviert, wenn sich das Objekt an der Maus befindet, um die Kombination zu ermöglichen.
Danach wird die Einstellung „Raycast-Target“ wieder aktiviert.

8. Konversationen

Konversationen dienen der dynamischen Unterhaltung mit einem Charakter oder einfach der Darstellung von Entscheidungen für den Spieler. Eine Konversation besteht aus mehreren Optionen, zwischen denen der Spieler auswählen kann. Jede dieser Optionen besteht aus 5 Parametern

- Name: Ein interner, eindeutiger Name zur besseren Auswahl.
- Text: Der Text, der auf dem Button angezeigt wird.
- Sequence: Die Sequenz, die ausgeführt wird, wenn der Spieler diese Option wählt.
- Enabled: Ein Wahrheitswert, der definiert, ob diese Option in der Konversation angezeigt wird oder nicht (im Spielverlauf änderbar).
- End Conversation after this: Ein Wahrheitswert, der definiert, ob die Konversation nach der Auswahl dieser Option beendet wird. (Sonst werden die Optionen erneut angezeigt)



Nutzung

Damit die Konversationen geführt werden können, muss sich ein **ConversationUI**-Script in der Szene befinden. Dieses ist zusammen mit dem Interface im **ConversationUI** – Prefab vorbereitet. Damit die Interaktion mit dem UI möglich ist, wird ein UI – **EventSystem** benötigt.

Der Start und die Veränderung einer Konversation erfolgt über die Action **ActionConversation**.

ActionConversation

Ist der Type „Start“ ausgewählt wird die in Conversation eingetragene Konversation gestartet. In diesem Fall wechselt das Spiel automatisch in den Cutscene-GameState, damit keine andere Eingabe mehr möglich ist. Es wird zunächst die welcomeSequence Sequenz der Konversation ausgeführt. Hier können sich die Charaktere begrüßen, oder Optionen der Konversation angepasst werden. Danach öffnet sich das Interface und zeigt alle angeschalteten (enabled) Optionen der Konversation untereinander an.

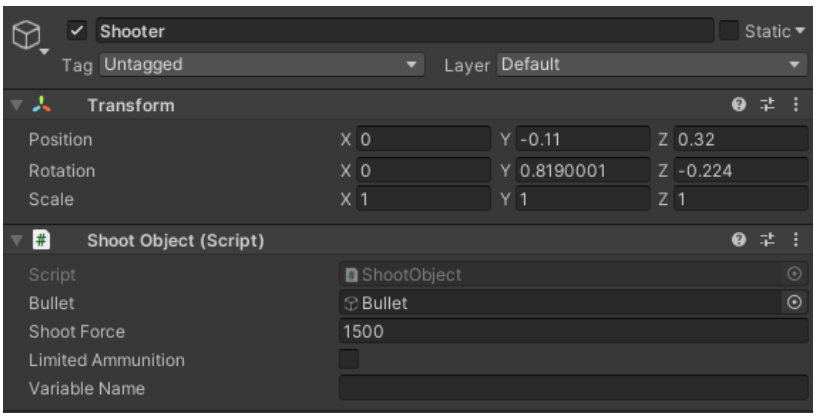
Verändern der auswählbaren Optionen

ActionConversation

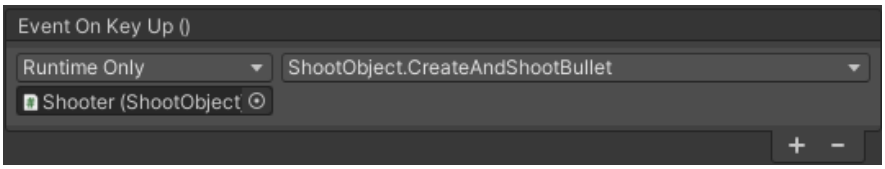
Ist der Type „SetOption“ ausgewählt, wird der Status einer Option verändert. Dafür müssen die entsprechende **Konversation** angegeben, der **Name der Option** im Drop Down Menü ausgewählt und der zu setzende Zustand **(Enable?)** definiert werden.

9. Schießen

Ein einfacher Komponent liegt dem Modul bei, der das Schießen von Objekten realisiert.



Um zu Schießen muss die Funktion „CreateAndShootBullet“ ausgeführt werden. (Beispielsweise durch das enthaltenen „Keyboard Events“-Script.)

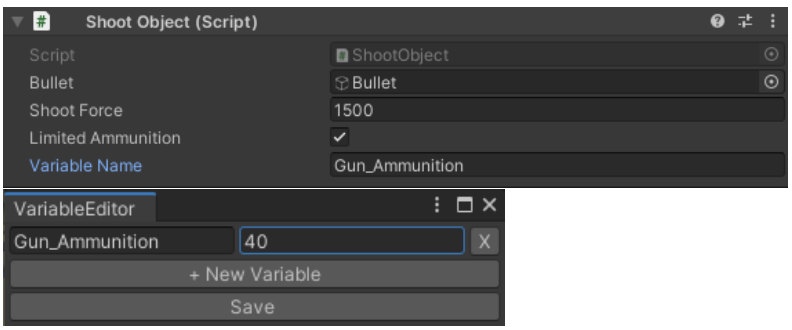


Der Komponent wird dann das Objekt **Bullet** duplizieren und an die Position des eigenen Objektes (Shooter) legen. Das **Bullet** Objekt benötigt zudem unbedingt einen **Rigidbody**, damit ihm die in **ShootForce** eingetragene Kraft übergeben werden kann. Die Bullet wird dann mit dieser Kraft in die Forward-Richtung des Grundobjekts (Shooter) geschleudert. Um die Richtung zu ändern, muss also dieses GameObject gedreht werden.

Munitionsbegrenzung

Um nur eine begrenzte Anzahl an Munition beim Schießen zu realisieren, wird das Variablen System benutzt. Schalten Sie zunächst die Variable **LimitedAmmunition** an und tragen Sie dann in **VariableName** den Namen der Variable ein, die die zur Verfügung stehende Munition angibt.

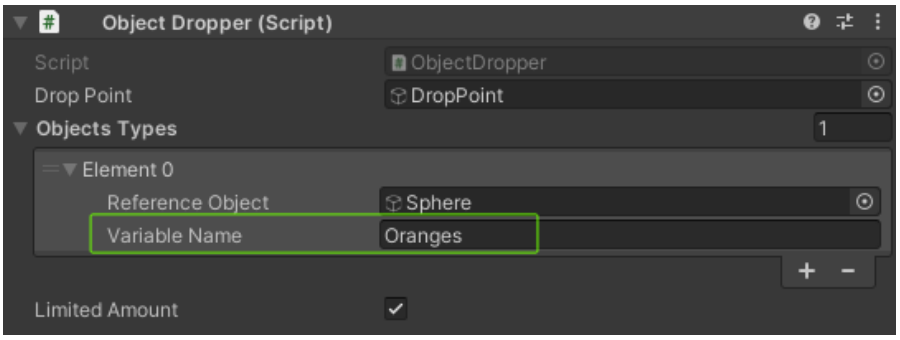
In diesem Fall wird diese Variable nach jedem Schuss um 1 verringert. Fällt der Wert der Variable auf 0, kann nicht mehr geschossen werden. Die Munition kann dann mit Hilfe von normalem Scripting und der Action **ActionVariable** wieder erhöht werden.



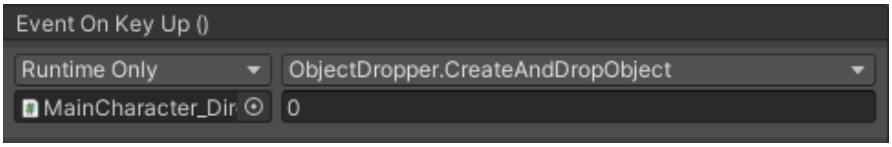
10. Fallenlassen

Ein einfacher Komponent liegt dem Modul bei, der das Fallenlassen von Objekten realisiert (ObjectDropper)
Im Feld **DropPoint** muss ein GameObject angegeben werden, an dessen Position die neuen Objekte erstellt werden sollen. (Meist ein Child-GameObject des Spieler-Charakters)

In der Liste **Object Types** können verschiedene Objekte definiert werden. Das in **Reference Object** angegebene Game Object wird beim Fallen lassen einfach kopiert und angeschalten.

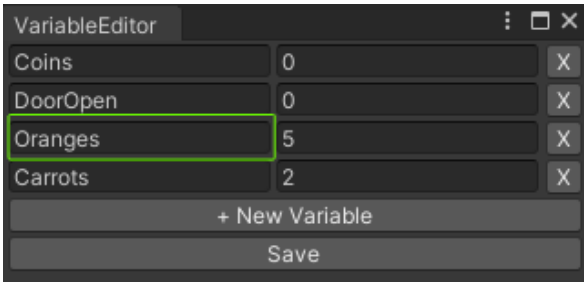


Um ein Objekt fallen zu lassen, muss die Funktion „CreateAndDropObject“ ausgeführt und ein entsprechender Index übergeben werden. Der Index bezieht sich auf die Objekt-Types in der Liste **Object Types** (beginnend bei 0!).



Begrenzung der Anzahl

Wird die Option **Limited Amount** aktiviert, muss jedem ObjectType eine Variable des Variablen-Manager zugewiesen werden – Durch Eintragen des **Variablennamen**. Diese Variable funktioniert dann als „Verfügbare Anzahl“ dieses Objekt-Types. Nach dem Fallen lassen eines Objekts, wird die entsprechende Variable um 1 verringert. Hat die Variable einen Wert von 0, wird kein weiteres Objekt erstellt. Die Anzahl kann dann mit Hilfe von normalem Scripting und der Action **ActionVariable** wieder erhöht werden.



11. Timer

Um zeitkritische Spielmechaniken zu ermöglichen, sind einfache Timer im Modul enthalten. Diese werden mit der Action ActionTimer gestartet. (Siehe Actions)

Läuft der Timer ab, wird eine angegebene Sequenz gestartet und der Timer entfernt. Für den Fall, dass der Timer frühzeitig gestoppt werden soll, kann die ActionTimer dafür verwendet werden. Zu diesem Zweck muss jedem Timer ein eindeutiger Name gegeben werden.

Anzeigen von Timern

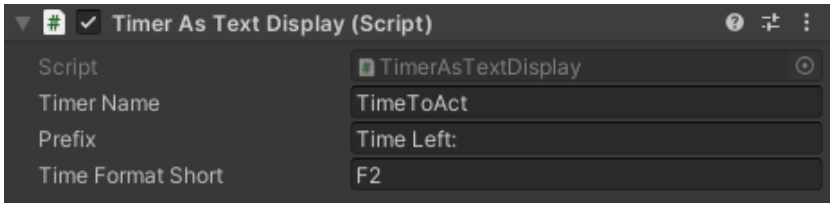
Ähnlich zu den Variablen, können auch die Werte von Timern im UI als Text oder „Füllung“ eines Bildes angezeigt werden.

Für die Text-Darstellung existiert derzeit nur die „Simple“ Methode. Es wird der eingetragene Prefix in den Text geschrieben, gefolgt vom Wert des Timers in Sekunden als Dezimalzahl.

Um die Darstellungsform zu verändern kann derzeit nur ein Text-Kürzel „TimeFormatShort“ angegeben werden.

Die Möglichkeiten und Verwendungen dieses Kürzels finden Sie bspw. hier:
https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings

Beispiel:
„0“ Ganze Zahlen
„N2“ Zwei Nachkommastellen



Die zweite Möglichkeit ist die Darstellung in einer **Grafik**.

Hierfür wird zunächst ein UI-**Image** Komponent benötigt, der als Type „**Filled**“ eingestellt ist. Die Variable **FillAmount** beschreibt zu wieviel Prozent die Grafik (in der angegebenen Richtung) zusehen ist (0-1)

Das Script **TimerAsImageDisplay** teilt den aktuellen Wert des angegebenen Timers durch dessen Startwert. Das Ergebnis wird im **FillAmount** des Image-Komponenten gespeichert.