

BLENDER GAME B

Tutorium von Raphael Menges

ZIEL

- Charakter kann laufen und springen
- Charakter bewegt passend die Arme und Beine
- Charakter dreht sich in Laufrichtung
- Charakter kann Hirn (-stücke) aufsammeln
- Aufgesammeltes Hirn wird in HUD angezeigt



DISCLAIMER

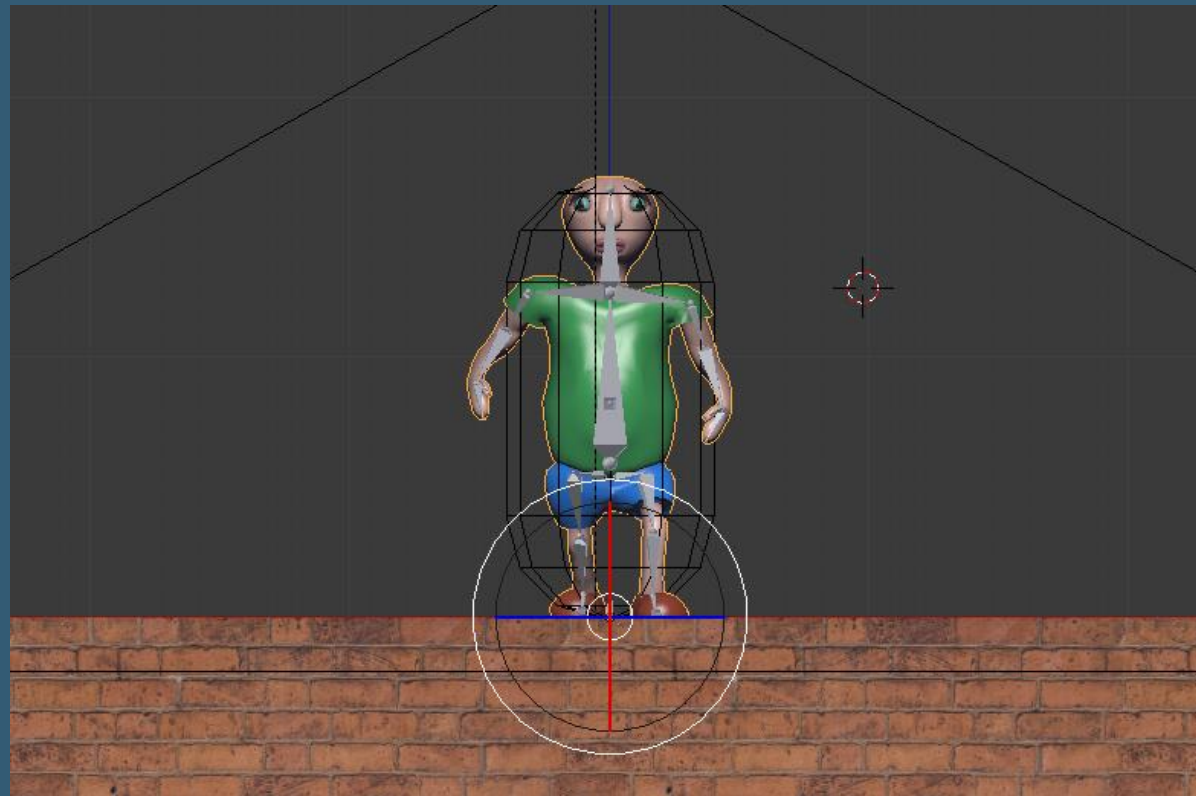
- Folgende Umsetzung muss nicht optimal sein, ich habe mich selbst erst für diese Veranstaltung in das Thema vertieft

THEMEN

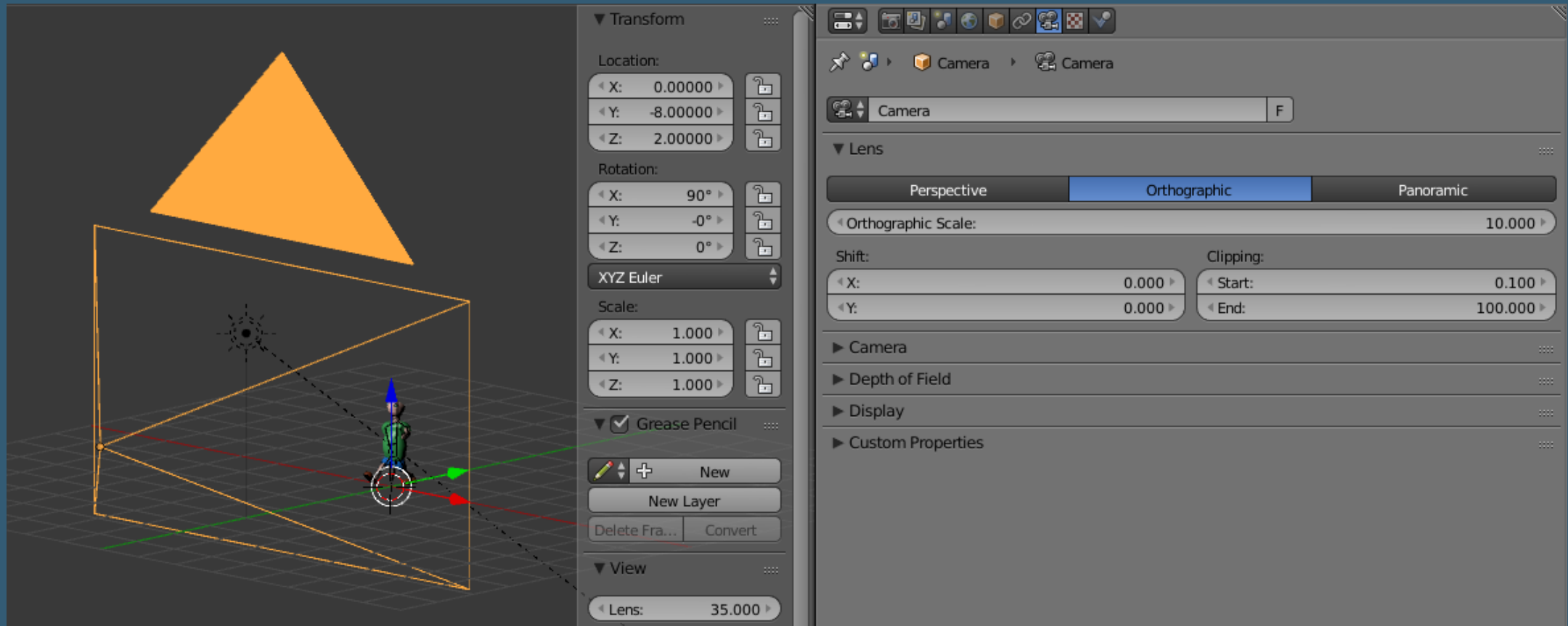
- Vorbereitung
- Bewegung
- Animation
- Aufsammeln
- HUD

VORBEREITUNG

- Template von meiner userpage (<http://userpages.uni-koblenz.de/~raphaelmenges/BGE1415/05>) herunterladen



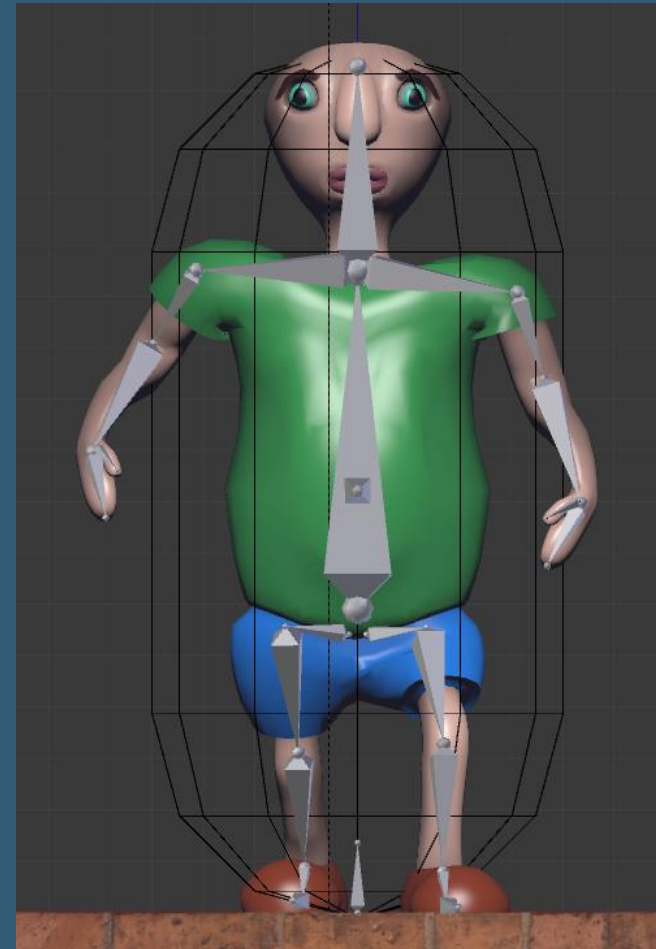
ORTHOGRAPHISCHE KAMERA



Schon vorbereitet im Template

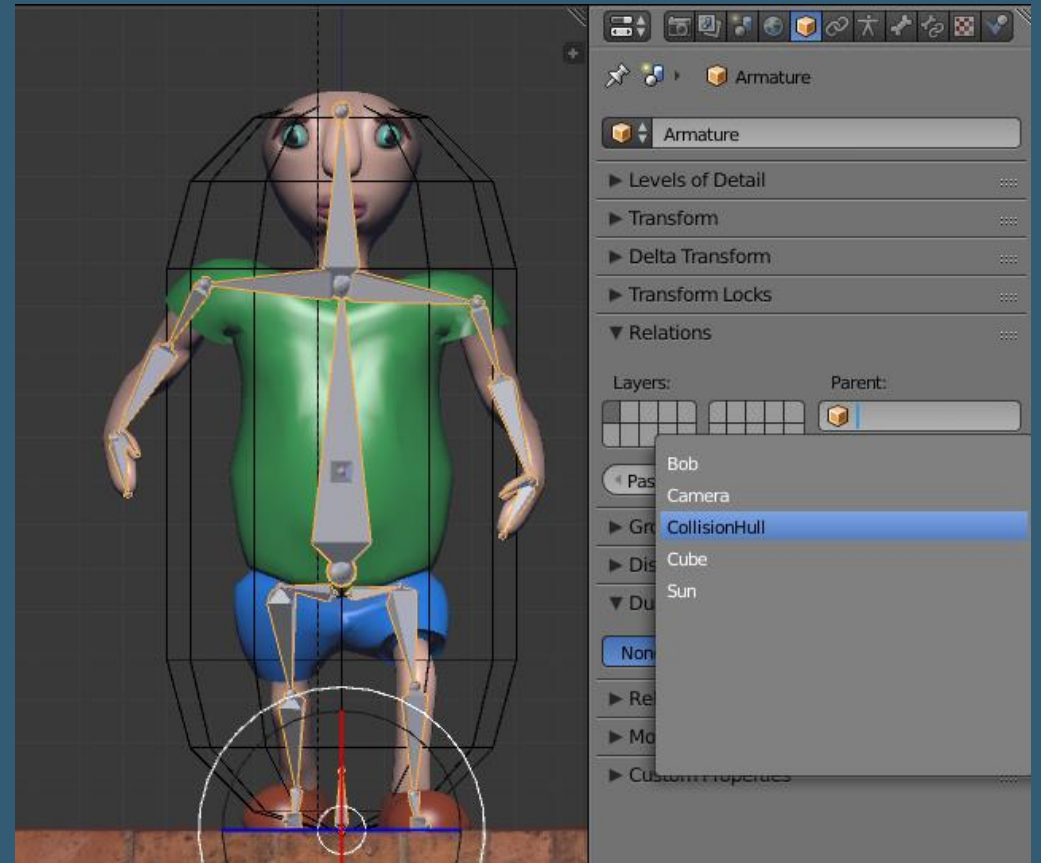
CHARAKTER

- Vorgehen ist wegen Physik etwas kompliziert
 - Man benötigt das Charaktermodell
 - Die Armature, welche das Parent vom Modell ist
 - Eine einfache Kollisionshülle
- Da das Modell an der Armature / dem Skelett hängt, bewegt sich das Modell nicht, noch kollidiert es selbst mit der Umwelt
- Die Armature selbst sollte auch nicht kollidieren
- Daher benutzt man eine Hülle, für welche man Physik und Logik erstellt. Das Skelett wird dann an diese Hülle gehängt



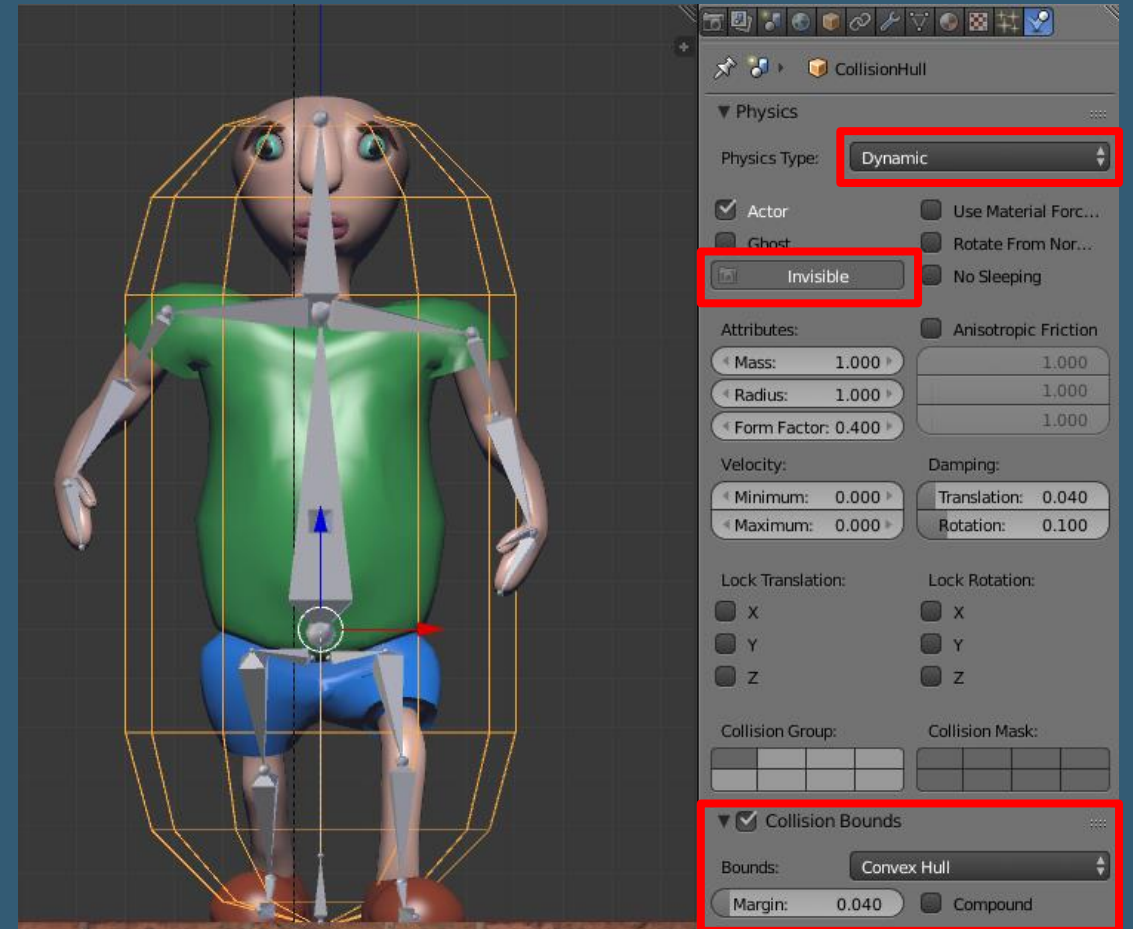
CHARAKTER

- Bis auf letzten Punkt soweit im Template vorbereitet...
- Einfach beim Skelett die Kollisionshülle („CollisionHull“) als Parent eintragen



CHARAKTER

- Jetzt noch die Physik der Kollisionshülle justieren
- Wenn man das Spiel startet, merkt man: Hülle ist zu lang. Daher noch etwas kürzer skalieren, und zwar im Edit Mode der Hülle, nicht im Object Mode



BEWEGTER CHARAKTER

- Nun wird die Kollisionshülle via Keyboard steuerbar gemacht

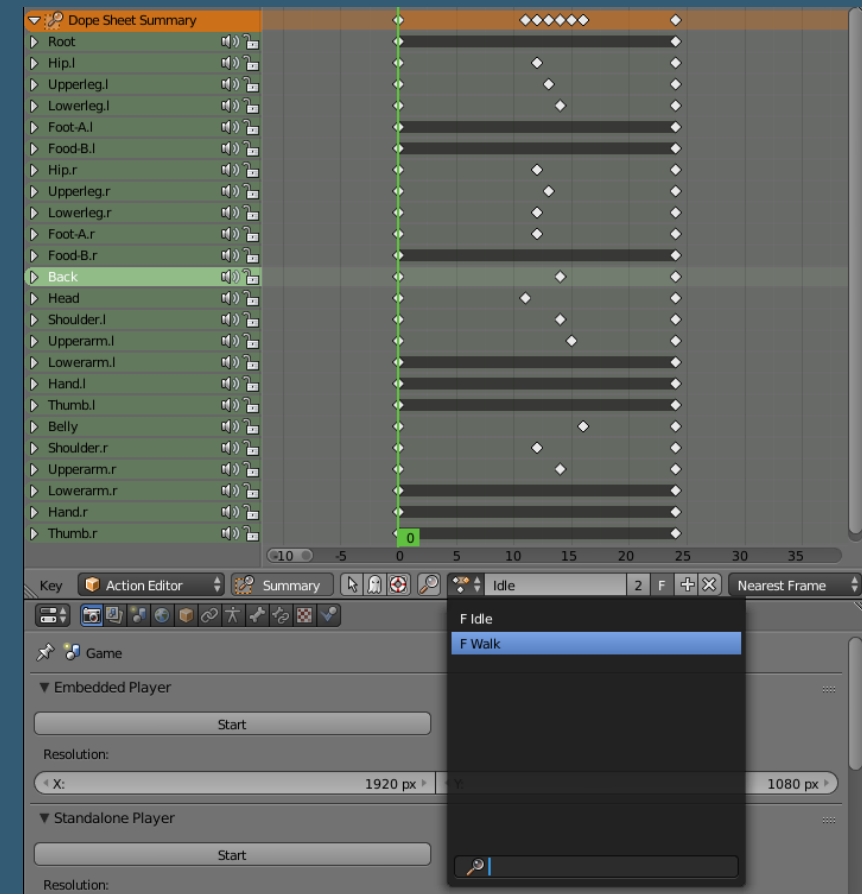
The screenshot displays the logic editor for a 'CollisionHull' object. It is organized into three main sections: Sensors, Controllers, and Actuators. In the Sensors section, three 'Keyboard' sensors are listed, each with a 'Tap' button highlighted in a red box. The Controllers section contains three 'And' controllers, each with a '1' button highlighted in a red box. The Actuators section contains three 'Motion' actuators, each with a 'Tap' button highlighted in a red box. Arrows point from the 'Tap' buttons in the Sensors and Actuators sections to the '1' buttons in the Controllers section, indicating that the sensors trigger the controllers, which then trigger the actuators. The actuators are configured with 'Simple Motion' and various parameters like Location, Rotation, Force, and Torque.

Logik des Objektes „CollisionHull“

Tap: Wird nur einmal ausgelöst (auch wenn z.B. Taste gedrückt bleibt)

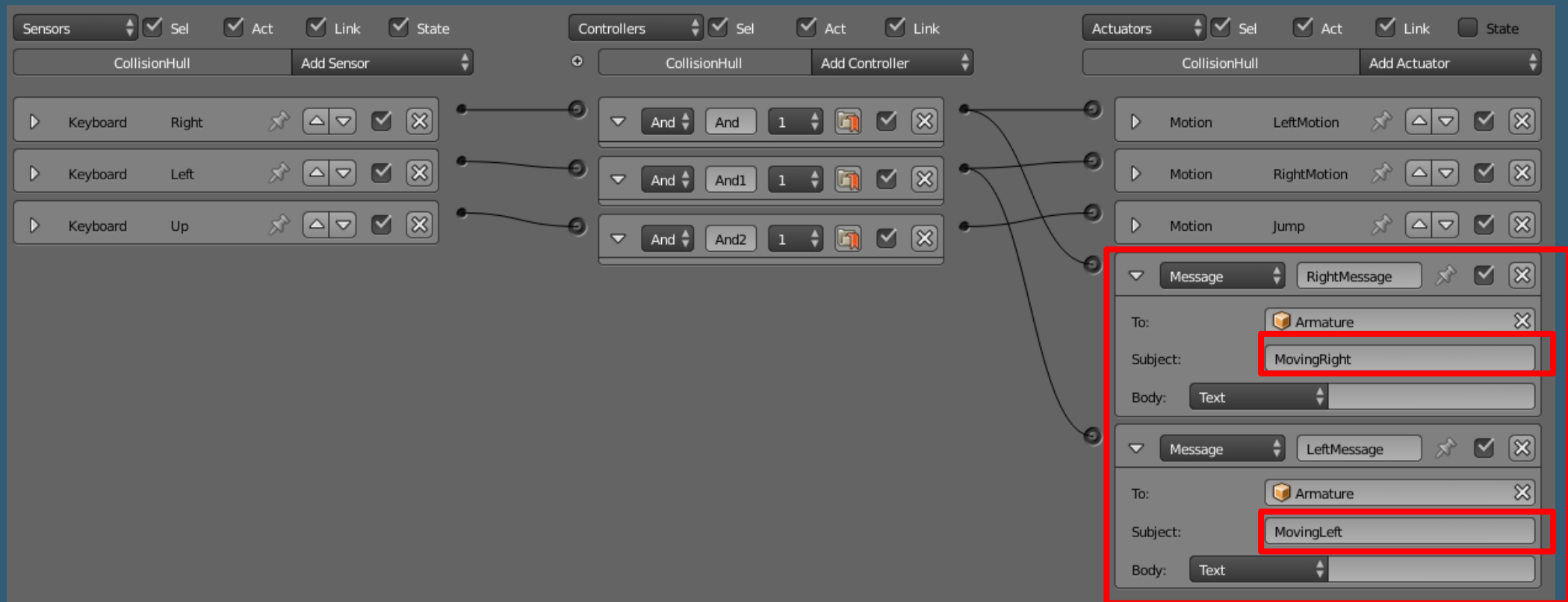
ANIMIERTER CHARAKTER

- Nun kann man den Charakter zwar bewegen, aber die Figur selbst verbleibt starr
- Das Armature wird von der Kollisionshülle nur „mitgezogen“ und macht selbst noch nichts
- Damit das Armature mitbekommt, welche Animation abgespielt werden soll, kann die Kollisionshülle das Armature zum Beispiel mit dem Message Actuator informieren
- Dabei wird zum einen der Empfänger und zum anderen das Subject der Nachricht angegeben (nächste Folie)



Vorhandene Actions für das Armature

ANIMIERTER CHARAKTER



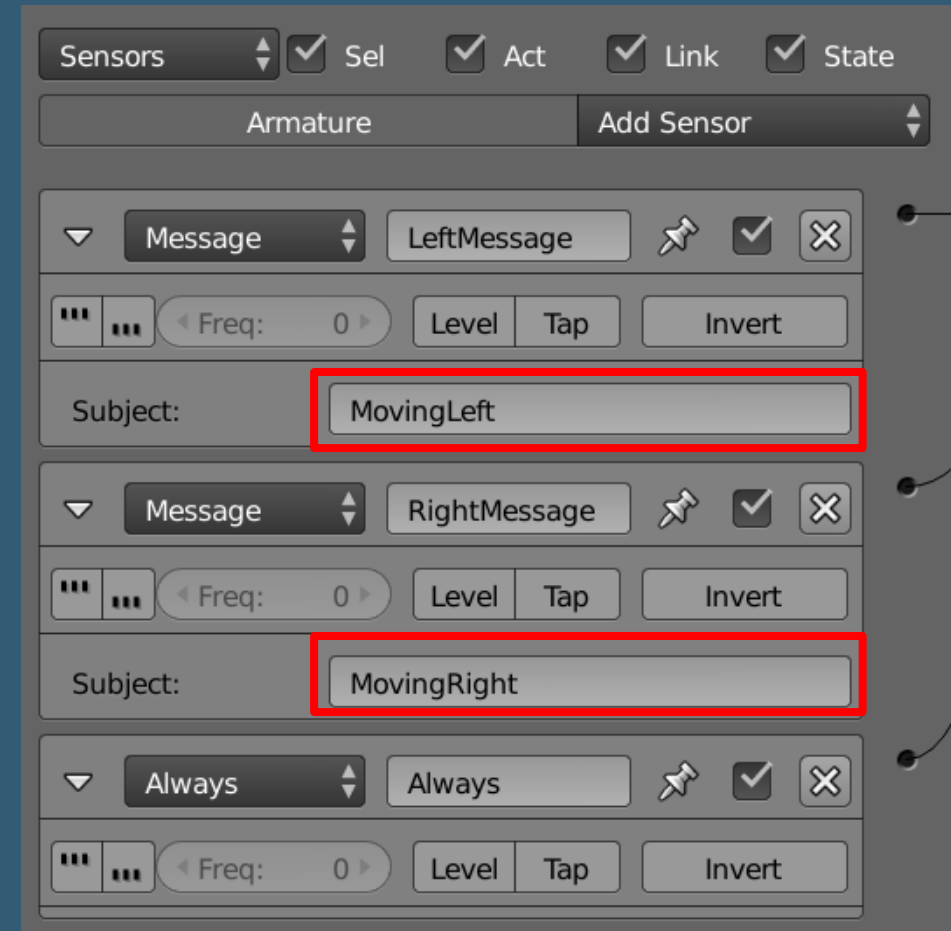
Erweiterte Logik des Objektes „CollisionHull“

ANIMIERTER CHARAKTER

- Es folgt die Logik innerhalb des Armatures
- Nun müssen die Nachrichten des Objekts „CollisionHull“ (aka Kollisionshülle) vom Objekt „Armature“ mit Hilfe eines Sensors entgegengenommen werden
- Nach der Auswertung durch einen Controller soll dann die passende Action (in denen die Animationen gespeichert sind) ausgeführt werden
- Zuletzt soll das Armature noch zur Laufrichtung gedreht werden
- Im Folgenden sind die Titel der Logikblöcke wichtig, da sie mit einem Script angesprochen werden sollen!

ANIMIERTER CHARAKTER

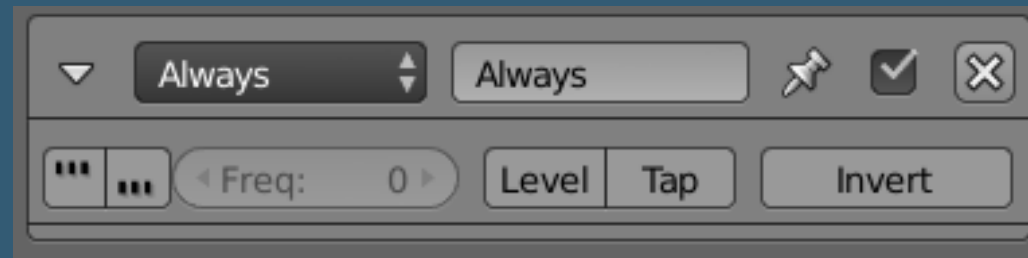
- Message Sensoren in Armature nehmen die Nachrichten von der Kollisionshülle entgegen
- Es muss kein Sender angegeben werden, aber das gleiche Subject wie beim sendenden Actuator
- Always wir benötigt, damit am Anfang schon die Idle-Animation abgespielt wird



Sensoren des Objektes „Armature“

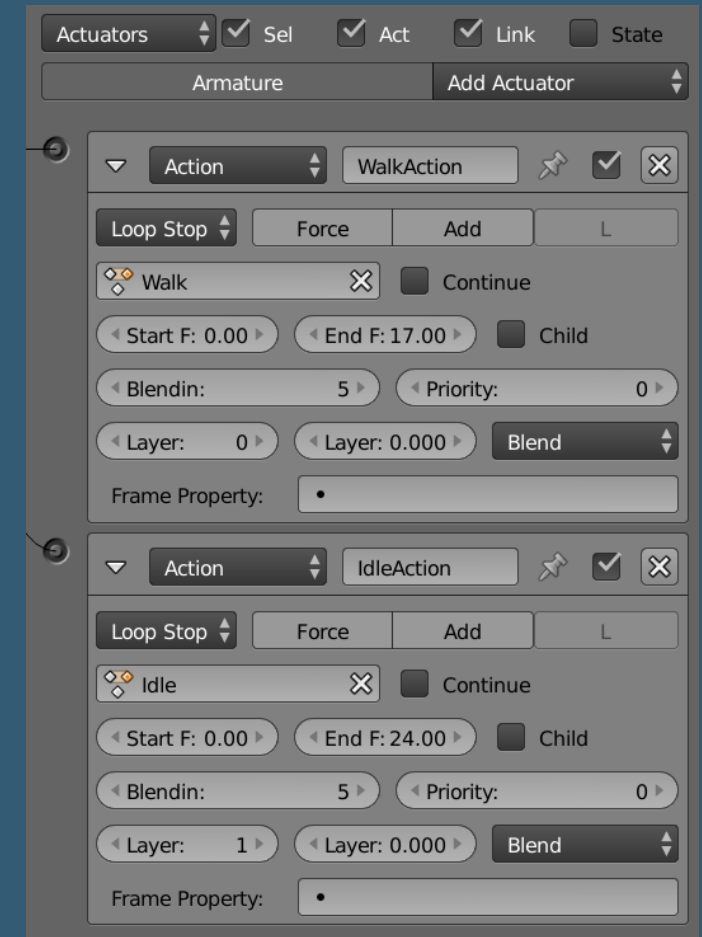
ALWAYS SENSOR

- Der Always Sensor sendet standardmäßig einen einzelnen Impuls zu Spielbeginn
- Linke „...“: Löst pro Intervall einen TRUE/positiven Impuls aus
- Rechte „...“: Löst pro Intervall einen FALSE/negative Impuls aus
- Freq: Einstellung des Intervalls



ACTION ACTUATOR

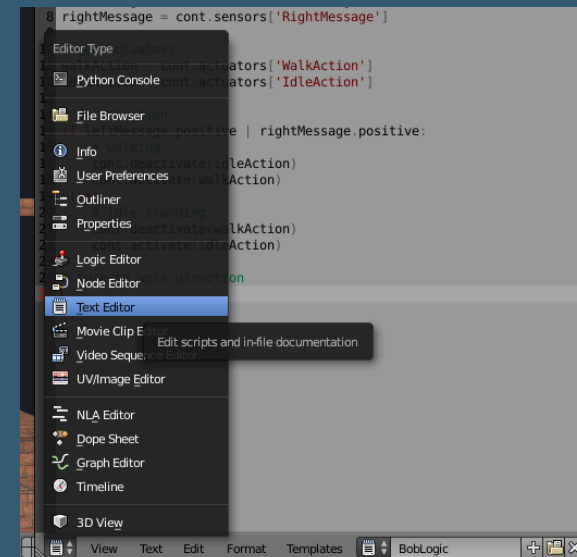
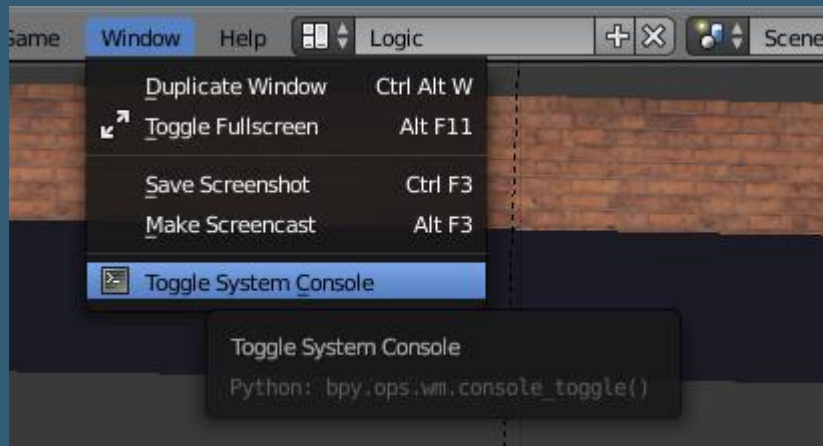
- Vor der Logik im Controller müssen noch Actuators zum Ansprechen erstellt werden. Pro Action wird jeweils einer in der Armature benötigt
- Wichtige Einstellungen im Action Actuator
 - Action Playback Type
 - Play: Spielt die Action einmal ab
 - Loop Stop: Spielt die Action, bis abgebrochen wird
 - Loop End: Spielt die Action, bis abgebrochen wird. Spielt sie da aber noch bis zum letzten Keyframe
 - Value: Name der Action
 - End Frame: Letzter Frame der Action bzw. die Länge
 - Blendin: Anzahl der Frames für Überblendung der Actions
 - Layer: Jede Action sollte in einen eignen Layer



Actuators des Objektes „Armature“

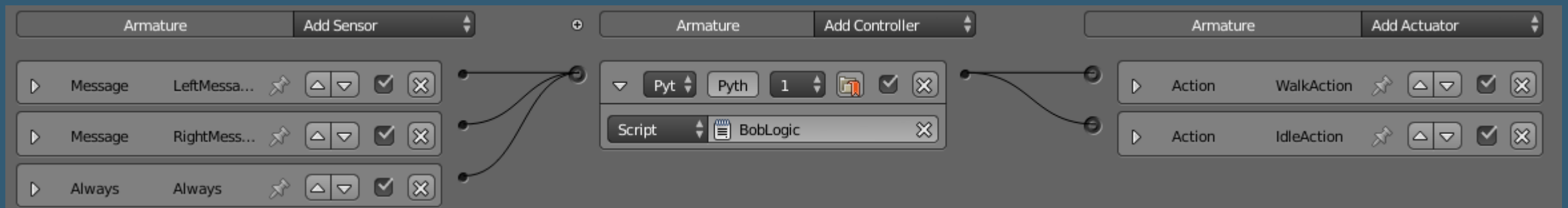
ANIMIERTER CHARAKTER

- Nun sollen die erstellten Sensoren und Actuators miteinander durch einen Controller verbunden werden
- Dazu wird ein Controller mit Python Script benötigt
- Erstmal sollte man die System Console öffnen und ein Viewport mit Text Editor



ANIMIERTER CHARAKTER

- Im Text Editor wird mit dem Plus in der Fußleiste einer neuer Text erstellt, hier „BobLogic“ genannt
- Im Armature wird in der Logik ein Controller erstellt und das neue Python Script zugewiesen
- Alle benötigten Sensoren und Actuators werden dann mit dem neuen Controller verknüpft



ANIMIERTER CHARAKTER

- Das Script selbst ist einfach
- Es gibt in Blender aber keine Code-Completion oder ähnliches
- Außerdem wird der Code nicht kompiliert sondern Zeile für Zeile während dem Spiel ausgeführt
- Daher immer die System Console im Blick haben, da kommen Fehlerausgaben während dem Spiel raus
- Jetzt sollten im Spiel die passenden Animationen bzw. Actions abgespielt werden

```
1 import bge
2
3 # get controller
4 cont = bge.logic.getCurrentController()
5
6 # get sensors
7 leftMessage = cont.sensors['LeftMessage']
8 rightMessage = cont.sensors['RightMessage']
9
10 # get actuators
11 walkAction = cont.actuators['WalkAction']
12 idleAction = cont.actuators['IdleAction']
13
14 # do animation
15 if leftMessage.positive | rightMessage.positive:
16     # walking
17     cont.deactivate(idleAction)
18     cont.activate(walkAction)
19 else:
20     # idle standing
21     cont.deactivate(walkAction)
22     cont.activate(idleAction)
```

BobLogic

ANIMIERTER CHARAKTER

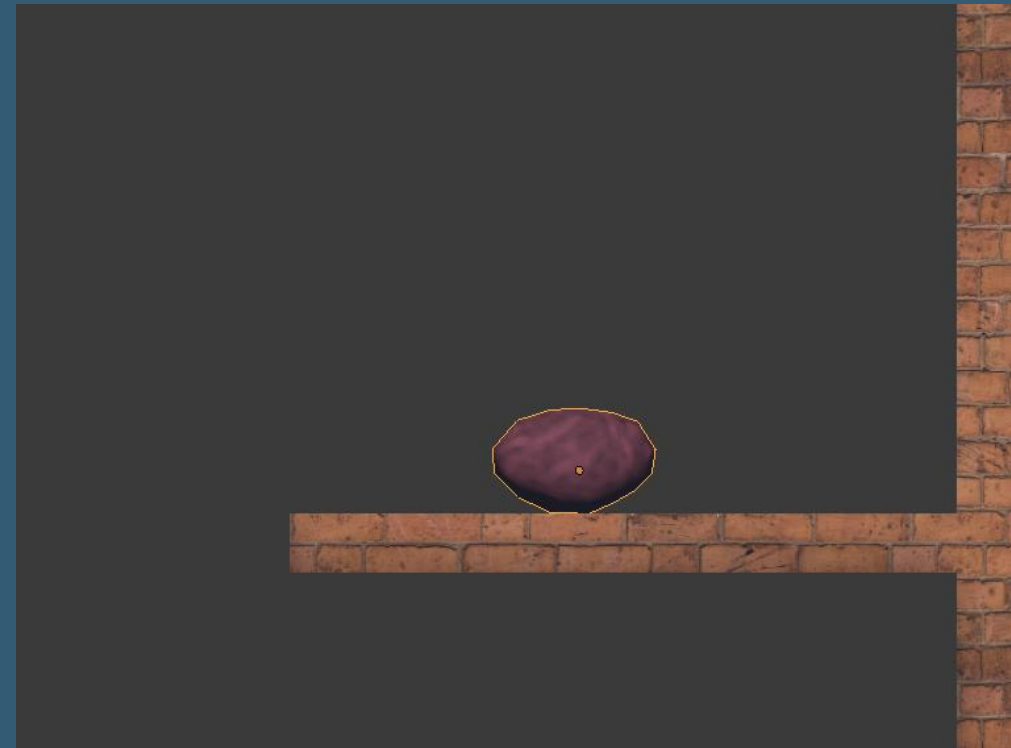
- Optional kann man die Armature während dem Laufen noch in die passende Richtung drehen. Dazu muss nur das Python Script der Armature erweitert werden

```
19 cont.deactivate(idleAction)
20 cont.activate(walkAction)
21 else:
22     # idle standing
23     cont.deactivate(walkAction)
24     cont.activate(idleAction)
25
26 # TURNING
27 import math
28
29 # get owner of controller (Armature)
30 own = cont.owner
31
32 # get rotation angles
33 xyz = own.localOrientation.to_euler()
34
35 # decide new angle
36 if leftMessage.positive:
37     xyz[2] = math.radians(-90)
38 elif rightMessage.positive:
39     xyz[2] = math.radians(90)
40 else:
41     xyz[2] = math.radians(0)
42
43 # set new angle
44 own.localOrientation = xyz.to_matrix()
```

Erweiterte BobLogic

AUSAMMELN VON HIRN

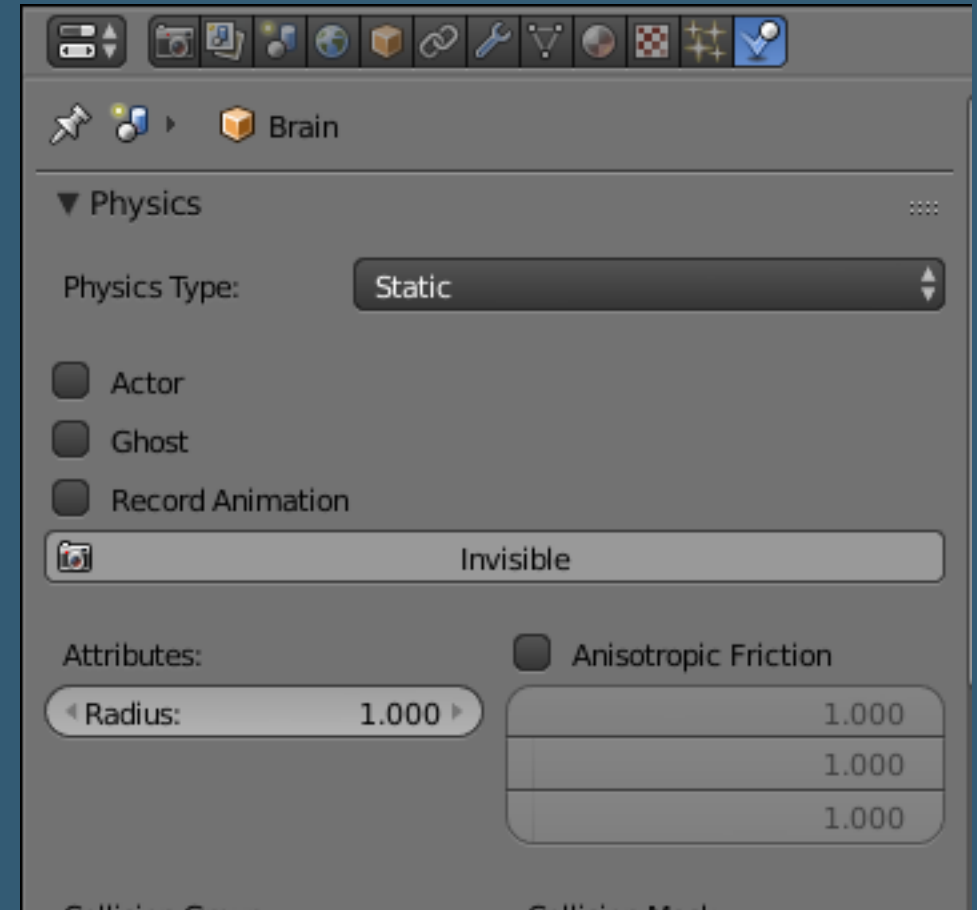
- Oben rechts liegt im Template ein Klumpen Hirn
- Dieser soll beim Darüberlaufen verschwinden (aufgesammelt werden)
- Das erfolgreiche Aufsammeln wird dem Spieler dann noch in einem HUD angezeigt



Objekt „Brain“

AUSAMMELN VON HIRN

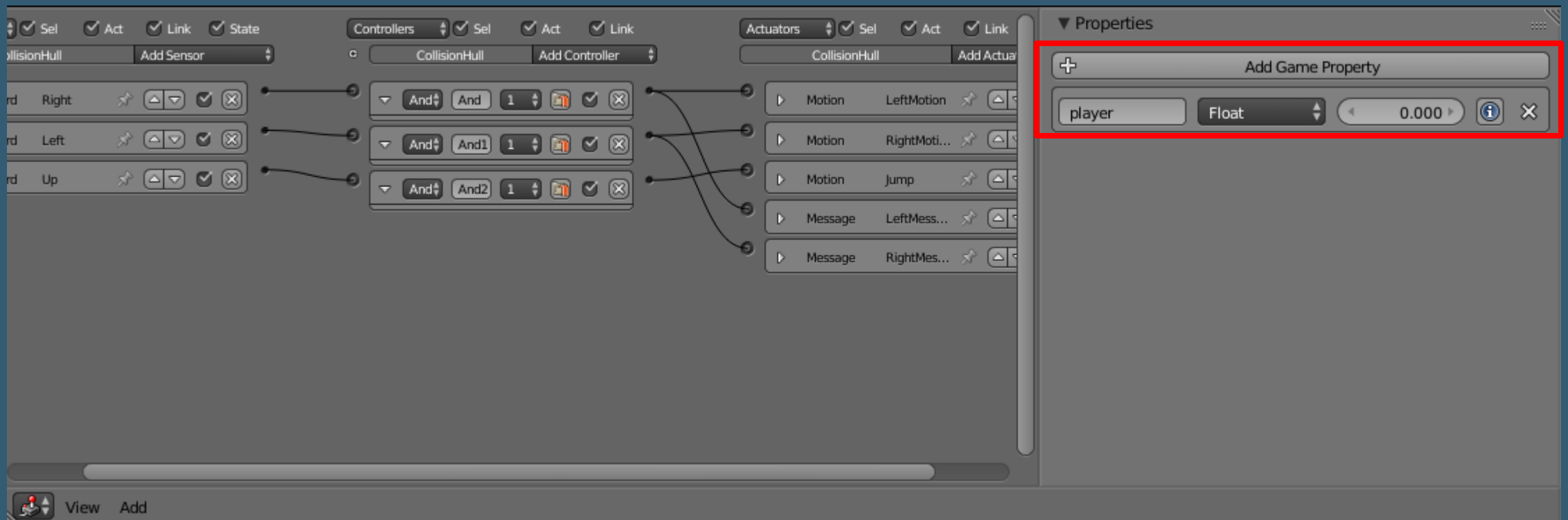
- Als erstes soll das Aufsammeln durch Darüberlaufen umgesetzt werden
- Dabei meldet das Hirn eine Kollision mit dem Spieler (bzw. dessen Kollisionshülle) und blendet sich dann selbst aus
- Das Hirn muss als Physik Typ auf Static oder ähnliches gestellt werden, damit eine Kollision erkannt werden kann



Physics Settings von „Brain“

AUSAMMELN VON HIRN

- Dann braucht das Objekt, mit dem man kollidiert, *irgendeine* Game Property
- In unserem Fall bekommt „Collision Hull“ die Game Property „player“



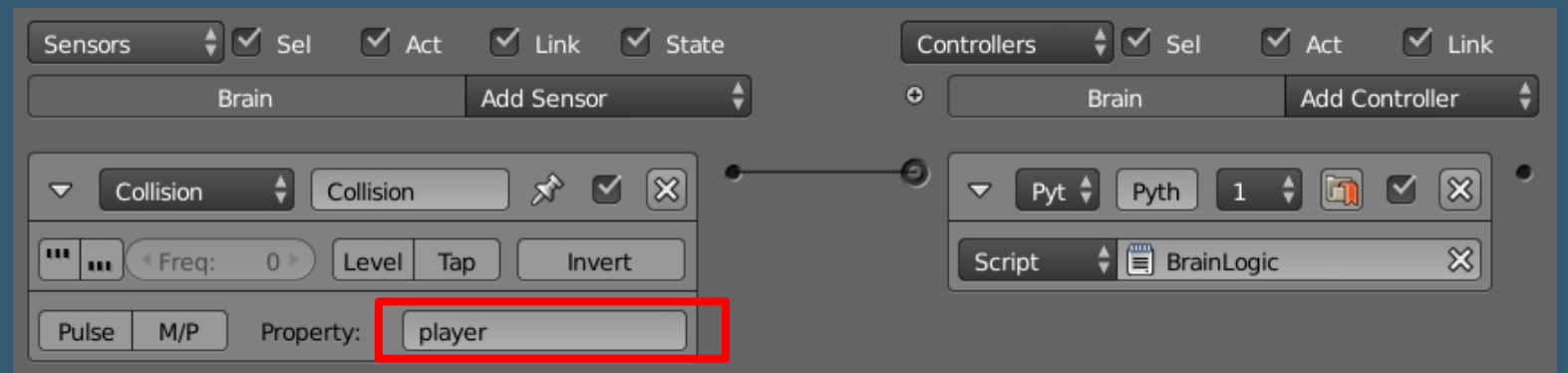
Logic von „CollisionHull“

AUSAMMELN VON HIRN

- Daraufhin im Objekt „Brain“ einen Sensor für Kollisionen erstellen und als Property „player“ angeben
- Nur Objekte mit dieser Property lösen den Sensor aus
- Der Sensor löst dann einen Controller aus, welcher ein Python Script benutzt (siehe rechts)

```
1 import bge
+2
3 # get controller
4 cont = bge.logic.getCurrentController()
5
6 # get owner of controller
7 own = cont.owner
8
9 # end object
10 own.endObject()
```

BrainLogic



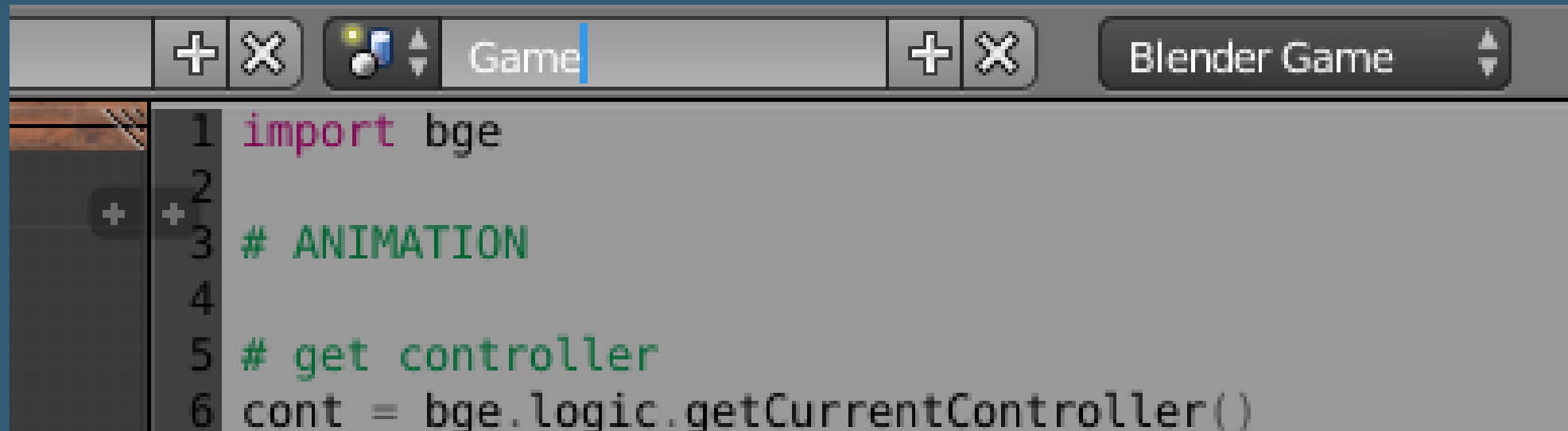
Logik von „Brain“

ANZEIGEN VON HIRN

- Nun soll nach dem Aufsammeln das Hirn im HUD sichtbar werden
- Es gibt aber kein explizites HUD oder GUI in der Blender Game Engine
- Man kann aber mehrere Szenen übereinander anzeigen
- Das wird jetzt etwas komplizierter, folgendes Vorgehen:
 - Die aktuelle Szene sinnvoll benennen („Game“)
 - Eine neue Szene erstellen („HUD“) und eine orthographische Kamera anbringen
 - Ein Stück Hirn platzieren, benennen („HUD_Brain“) und auf invisible stellen
 - Mithilfe eines Controller-Scripts der Kamera der Szene „Game“ die Szene „HUD“ als Overlay benutzen
 - Beim Aufsammeln von „Brain“ in „Game“ das „HUD_Brain“ in „HUD“ auf visible setzen

ANZEIGEN VON HIRN

- Die aktuelle Szene sinnvoll benennen („Game“)



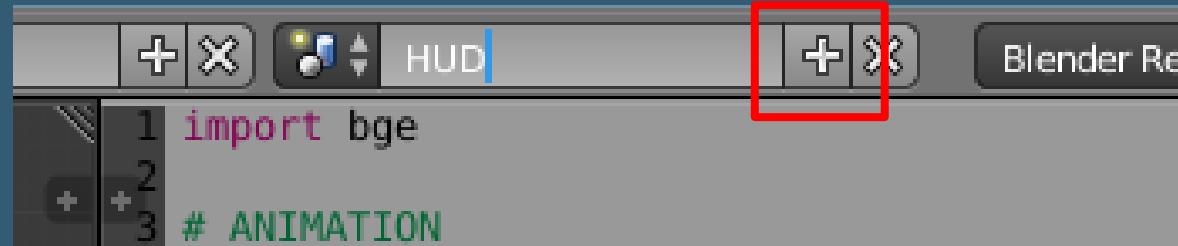
The image shows a screenshot of the Blender 2.79 interface. The top title bar displays the scene name 'Game' and the window title 'Blender Game'. Below the title bar, a code editor window is open, showing Python code for Blender Game Engine (BGE) logic. The code is as follows:

```
1 import bge
2
3 # ANIMATION
4
5 # get controller
6 cont = bge.logic.getCurrentController()
```

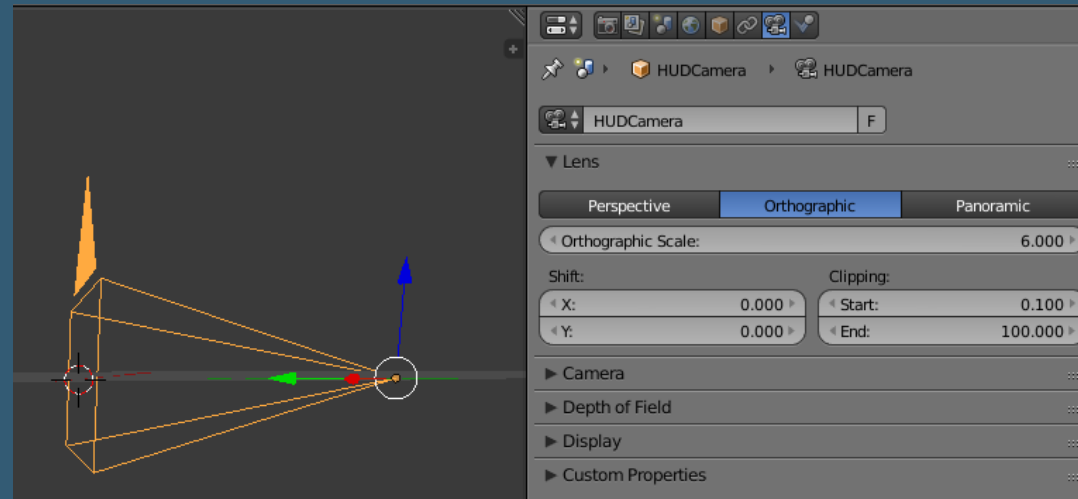
Titelleiste von Blender

ANZEIGEN VON HIRN

- Eine neue Szene erstellen („HUD“) und eine orthographische Kamera anbringen



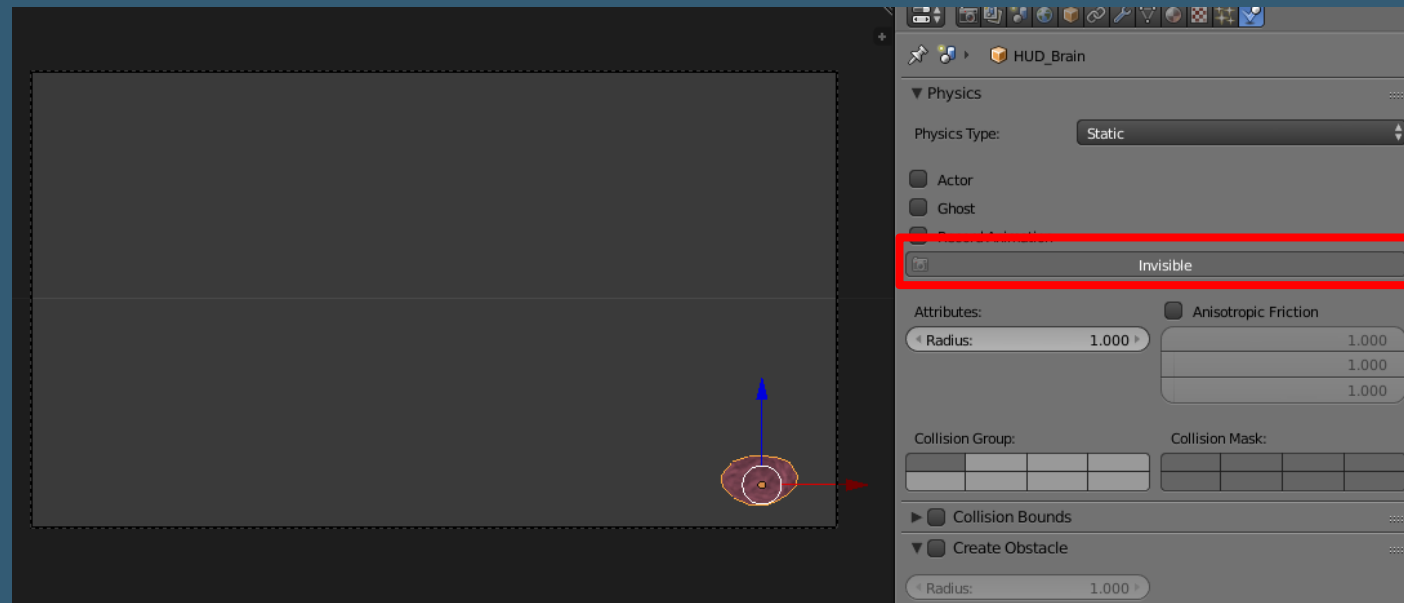
Titelleiste von Blender



Szene „HUD“

ANZEIGEN VON HIRN

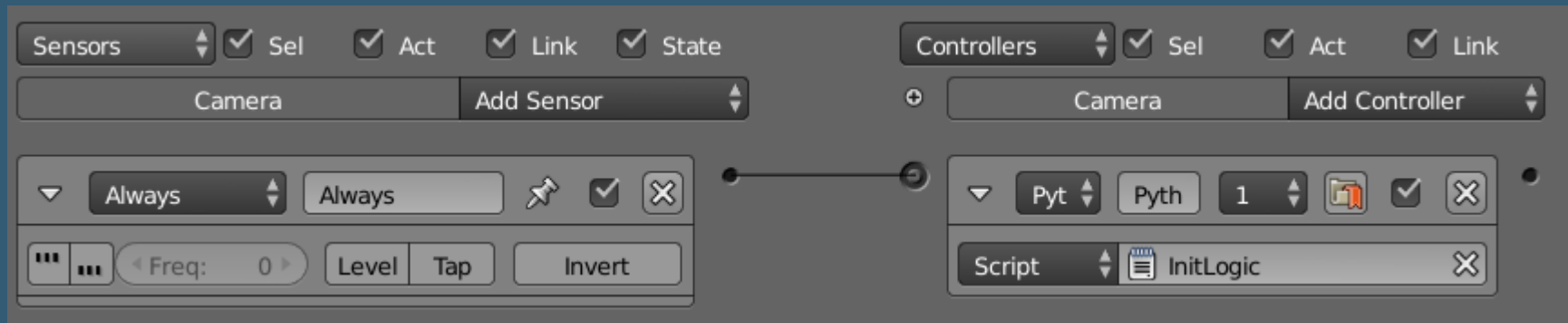
- Ein Stück Hirn platzieren, benennen („HUD_Brain“) und auf invisible stellen
 - „Brain“ in „Game“ duplizieren und via Outliner und Drag'n'Drop in „HUD“ verschieben
 - Logik aus kopierten „Brain“ löschen und umbenennen
- Nicht die Beleuchtung in der neuen Szene vergessen..



Szene „HUD“

ANZEIGEN VON HIRN

- Mithilfe eines Controller-Scripts in der Kamera der Szene „Game“ die Szene „HUD“ als Overlay benutzen



Szene „Game“

```
1 import bge
2
3 # use scene "HUD" as overlay
4 bge.logic.addScene("HUD", True)
```

InitLogic

ANZEIGEN VON HIRN

- Beim Aufsammeln von „Brain“ in „Game“ das „HUD_Brain“ in „HUD“ auf visible setzen

```
1 import bge
2
3 # get controller
4 cont = bge.logic.getCurrentController()
5
6 # get owner of controller
7 own = cont.owner
8
9 # end object
10 own.endObject()
11
12 # get hud scene
13 hud = bge.logic.getSceneList()[1]
14
15 # get object from hud
16 hud.objects["HUD_Brain"].setVisible(True)
```

Erweiterte BrainLogic

ZIEL

- Charakter kann laufen und springen
- Charakter bewegt passend die Arme und Beine
- Charakter dreht sich in Laufrichtung
- Charakter kann Hirn (-stücke) aufsammeln
- Aufgesammeltes Hirn wird in HUD angezeigt



ANREGUNGEN

- Man könnte einen Mehrfachsprung mit einem Ray Sensor verhindern
 - Nur wenn Boden unter den Füßen ist, kann man hochspringen
- Aufgesammeltes Stück Hirn kann im Kopf des Charakters auftauchen
 - Man könnte es von Anfang an unsichtbar an den Kopf-Bone hängen und beim Aufsammeln des Stückes sichtbar machen
- Bemerkung: Action Actuator funktioniert nicht, wenn es neben Armature noch andere Modifier im Objekt gibt