Spieleprogrammierung und 3D-Animation Praktikum 1

Allgemeines

Im Praktikum zur Veranstaltung Spieleprogrammierung und 3D-Animation erlernen Sie den praktischen Umgang mit der Spiele-Engine Unity. Es dient als Ergänzung zu den theoretischen Inhalten aus der Vorlesung. Die grundlegenden Konzepte der Spieleentwicklung sind i. d. R. unabhängig von der genutzten Engine, so dass das Wissen, welches Sie bei der Entwicklung mit Unity erlangt haben, leicht auf andere Systeme übertragen werden kann. So sollte Ihnen neben dem Spezialwissen in Unity die Einarbeitung in andere Engines deutlich leichter fallen, wenn Sie das Praktikum erfolgreich absolviert haben.

Zum Bestehen des Praktikums benötigen Sie für das komplette Praktikum mindestens 80% der zu vergebenden regulären Punkte und dürfen kein Aufgabenblatt mit weniger als 50% der Punkte abschließen. Die Punkte pro Aufgabe erhalten Sie nur, wenn Sie beim Testat Ihre Lösungen schlüssig erläutern können.

Bei diesem Aufgabenblatt können Sie regulär 12 und mit Zusatzaufgaben 15 Punkte erreichen.

Das **Testat** zum Aufgabenblatt erfolgt am: **19.04. (Mittwoch) & 21.04. (Freitag)**

Zur Bearbeitung der Aufgaben laden Sie bitte das Package „P1_Package.unitypackage“ aus dem OSCA-Lernraum herunter, legen Sie ein neues Unity Projekt an und importieren Sie das Package (Menüpunkt Assets->Import Package->Custom Package..).

Arbeiten Sie in der TipToeScene Szene.

Thema von Praktikum 1

Das Ziel des ersten Aufgabenblatts ist es, dass Sie sich mit der Entwicklungsumgebung von Unity vertraut machen und Ihre ersten Skripte verfassen. Hierfür sollen Sie den Spielmodus *TipToe* aus dem Spiel *Fall Guys* logisch nachempfinden.

In TipToe müssen die Spieler versuchen, den richtigen Pfad über ein Raster aus Plattformen zu finden. Einige Plattformen sind dabei gut (stabil) und die Spieler können darauf stehen, andere sind

jedoch Fake-Plattformen und verschwinden, wenn der Spieler drauftritt (siehe https://www.youtube.com/watch?v=kjWhnk8qha4).

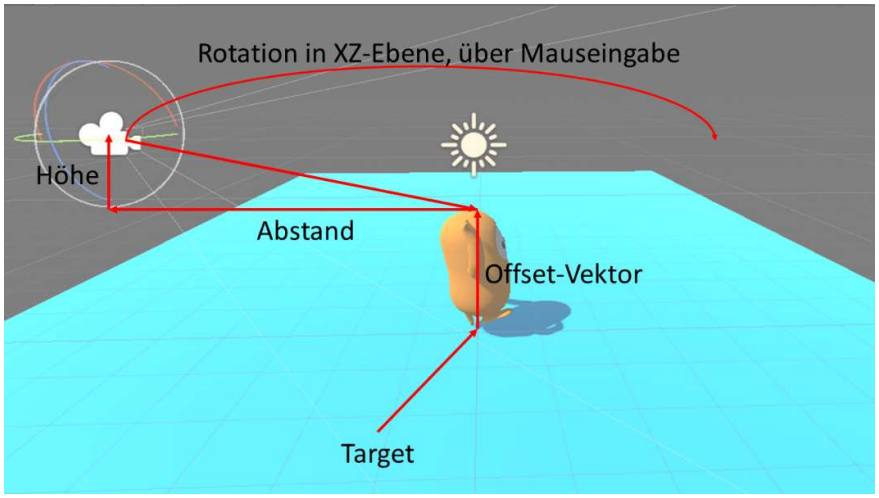
Nach diesem ersten Praktikum werden Sie einen Spieler-Charakter mit Maus und Tastatur durch das Level steuern können (Aufgaben 1-3), seine Animationen steuern (Aufgabe 4) und die Logik des Spielmodus entwickelt haben (Aufgaben 5-6).

Didaktische Themen:

- Bedienung des Unity Editors
- Unity C#-Scripting
- Arbeiten mit GameObjects & Components
- Kommunikation zwischen Components
- Eingabeabhängige Bewegungssteuerung über Transforms
- Einfache Spiellogiken
- Update-Loop
- Debugging

Aufgabe 1 (2 Punkte)

Ziel der ersten Aufgabe ist es, ein eigenes Kamera-Skript zu erstellen. Die Kamera soll den Spielcharakter im Blickzentrum haben, sich mit ihm bewegen und über Mausbewegungen um ihn rotieren können. Die Kamera wird über die Seitwärtsbewegung der Maus bei gedrückter linker Maustaste (links & rechts) um die Y-Achse der Welt rotiert. Der initiale Abstand und die Höhe zum Charakter (Target) soll über den Editor einstellbar sein. Ferner soll ein Offset-Vektor zum Spielcharakter und somit zum Blickzentrum im Editor einstellbar sein (initial x=0, y=0, z=0).



Für die Implementierung ziehen Sie das Fallguy Prefab in die Szene (Assets->Prefabs->Fallguy) und erstellen ein Skript für das Main-Camera-GameObject in Ihrer Szene. Nutzen Sie die Position des Charakters, um Ihre Kamera zu positionieren und die Mauseingaben, um die Kamera um den Charakter herum zu rotieren. Entwickeln Sie die Kamera so, dass sie ohne Änderungen im Code auch für andere Spielobjekte (Targets) genutzt werden kann.

Tipp: Schauen Sie sich mal die LateUpdate-Methode im Vergleich zur Update-Methode an.

Tipp: Um die Interaktion von Kamera und Charakter zu testen, können Sie sich die Scene-View und Game-View gleichzeitig anzeigen lassen. Wenn Sie den Charakter zur Laufzeit über die Scene-View verschieben, sehen Sie zum Beispiel, ob sich die Kamera mitbewegt.

Extra Achievement (+1 Bonuspunkt):

Implementieren Sie eine „Zoom"-Funktion über das Mouse-Scrollrad mit minimaler und maximaler Entfernung, die im Editor eingestellt werden können.

Aufgabe 2 (3 Punkte)

In der zweiten Aufgabe wird eine einfache Steuerung des Charakters entwickelt. Vervollständigen Sie dafür das *SimpleCharacterControl* Skript.

Ihr Charakter soll am Ende in X- und Z-Richtung laufen können und über die w-, a-, s- & d-Tasten gesteuert werden. Ihr Charakter soll nicht rückwärtslaufen, sondern sich immer in die Richtung drehen, in die er läuft. Die Links-/Rechts- und Vor-/Zurück-Bewegung soll im Raum der Kamera implementiert werden. Das bedeutet, dass die Bewegung des Charakters nach vorne entlang und die Bewegung nach hinten entgegen der Blickrichtung der Kamera verläuft. Läuft der Charakter nach Links, so ist es das Links aus Sicht der Kamera.

Sie haben verschiedene Optionen, Ihren Charakter zu bewegen. Die wohl einfachste Möglichkeit ist seine *transform*-Komponente zu manipulieren. Alternativ könnten Sie ihn auch über einen kinematischen *Rigidbody* und die *MovePosition* Funktion bewegen. **Wichtig:** Nutzen Sie weder einen *CharacterController*, noch einen *RigidBody* der nicht kinematisch ist.

Aufgabe 3 (2 Punkte)

Aufgabe 3 erweitert die Steuerung des Charakters. Ziel ist es den Charakter fallen zu lassen, wenn unter ihm nichts ist.

Da Sie ihren Charakter entweder mit einem kinematischen *RigidBody* oder einfach über seine *transform*-Komponente bewegen, übernimmt Unity nicht die Ausübung von Schwerkraft oder das Handling von Kollisionen. Damit Ihr Charakter aber fällt, wenn er von einer Plattform läuft, müssen Sie überprüfen, ob er gerade Boden unter den Füßen hat oder nicht. Falls der Charakter nicht auf einer Oberfläche steht, müssen Sie ihn nach unten bewegen.

Um die Kollision mit dem Boden zu überprüfen, gibt es verschiedene Möglichkeiten. Eine Option wäre zum Beispiel, einen *SphereCast* unter die Knie des Charakters durchzuführen (senkrecht nach unten).

Passen Sie ihr Skript so an, dass der Charakter fällt, wenn unter ihm kein Boden ist.

Aufgabe 4 (1 Punkt)

In Aufgabe 4 wird Ihrem Charakter nun durch Animationen Leben eingehaucht.

Dafür sollten Sie dem Charakter zunächst den passenden Animator-Controller geben. Wählen Sie dafür den Fallguy im Editor aus und werfen einen Blick in den *Inspector* rechts. Einer der Komponenten ist der *Animator*, der als Feld einen Controller hat, in dem zur Zeit keiner ausgewählt ist. Fügen Sie dort den *Fallguy_Controller* ein (Assets-> Animators->Fallguy_Controller) und öffnen ihn per Doppelklick.

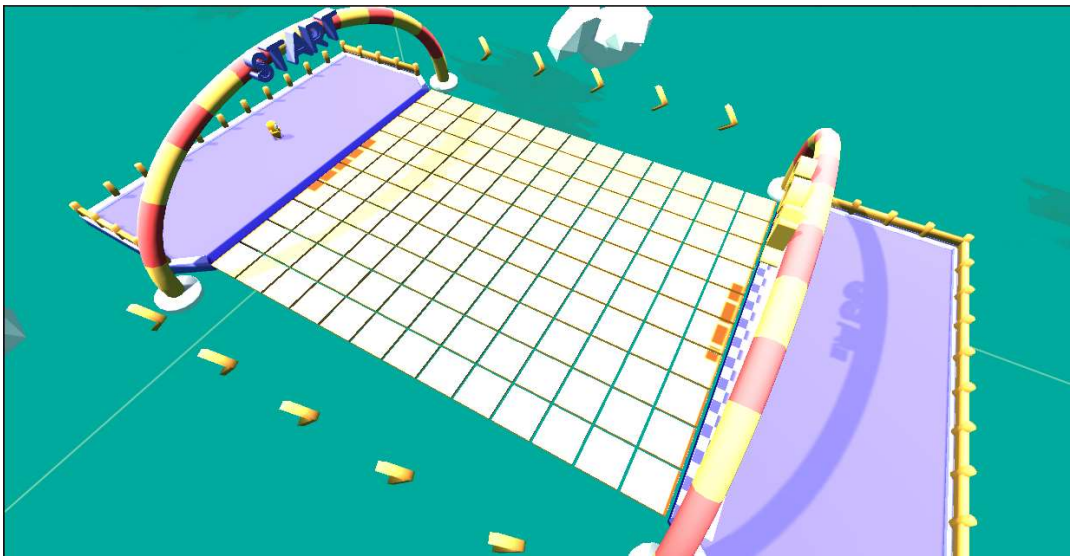
Dort sehen Sie nun die verschiedenen Animationen als Zustände und die Transitionen in Form von Pfeilen. Wenn Sie auf die Transition von *Airborne* zu *Landing* klicken, sehen Sie zum Beispiel, dass es eine *Condition* gibt, damit diese Transition ausgelöst wird: *Grounded==true*. Tatsächlich gibt es in dem *Animator* nur zwei für Sie interessante Parameter: *bool Grounded*, der entscheidet, ob ein Charakter vom Boden in seine *Jump*- oder *Falling* Animation übergehen soll und *float Speed*, welcher es ermöglicht, zwischen der *Idle*- und *Walk* Animation zu wechseln. Wollen Sie den Charakter also korrekt animieren, müssen Sie in Ihrem Skript auf diese Parameter zugreifen und sie passend setzen.

Aufgabe 5 (2 Punkte)

Nun haben Sie einen beweglichen Charakter, also bleibt nur noch die Spiellogik für dieses Level zu implementieren. Die *TipToeScene* enthält bereits zwei simple Plattformen für den Start und das Ziel des Levels. In dieser Aufgabe soll das *TipToeLogic* Skript so erweitert werden, dass bei Start des Spiels kleine Plattformen zwischen dem Start und Ziel Objekt erscheinen. Die Plattformen sollen ein Raster bilden, das aus 10 Plattformen in der Breite und 13 in der Tiefe besteht, insgesamt also 130 Stück. Bei so vielen Objekten ist es sinnvoll instancing zu benutzen. Vor Start des Spiels darf also maximal eine dieser Plattformen in der Szene auftauchen.

Erstellen Sie sich zuerst ein passendes Prefab für Ihre Plattform, und geben Sie ihm das *TipToePlatform* Skript. Füllen Sie die Variablen des Skripts sinnvoll. Stellen Sie sicher, dass Ihr Charakter Ihre Plattformen als ein Boden-Hindernis erkennt und nicht hindurch fällt (abhängig von Ihrer Implementation von Aufgabe 3).

In der Szene befindet sich bereits ein *TipToeLogic* GameObject, auf dem das *TipToeLogic* Skript liegt. Dort können Sie das Prefab für Ihre Plattform übergeben. Nun können Sie das *TipToeLogic* Skript bearbeiten und Ihre Plattformen an den passenden Stellen erscheinen lassen. Achten Sie darauf, dass die Plattformen auf der gleichen Höhe wie Start und Ziel sind, da der Spieler nicht springen kann. Der Charakter soll zudem nicht durch die Lücken zwischen den Plattformen fallen.
Tipp: Unity bietet die *Instantiate()* Methode zur Instanziierung von GameObjects während der Laufzeit. Diese kann vorhandene GameObjects kopieren oder Prefabs der Szene hinzufügen.



Aufgabe 6 (2 Punkte)

Zu guter Letzt fehlt nun noch, dass ein Teil Ihrer Plattformen verschwindet, wenn der Spieler auf sie tritt. Dafür müssen Sie bei der Instanziierung Ihrer Plattformen die *isPath* Variable des

TipToePlatform Skripts auf der jeweiligen Plattform so anpassen, dass es einen richtigen Pfad vom Start zum Ziel gibt. Alle anderen Plattformen bekommen *isPath* = *false*.

Im *TipToePlatform* Skript gibt es eine *CharacterTouches()* Funktion, die den Zustand der Plattform anpasst, sodass sie entweder aufleuchtet (wenn sie Teil des Pfads ist) oder verschwindet. Diese ist *public* und kann von einem anderen Skript aus ausgelöst werden. Solange Ihr Charakter also auf einer Plattform steht, muss er auf dieser Plattform die *CharacterTouches()* Funktion auslösen.

Extra Achievement (+2 Bonuspunkte)

Erweitern Sie das Spiel so, dass der Spieler gewinnen kann, wenn er die Zielplattform erreicht oder zurückgesetzt wird, wenn er runterfällt.