



**Hochschule  
Bonn-Rhein-Sieg**  
University of Applied Sciences

# Grundlagen für Visual Computing

-

## Murmelfuchs

von  
Kai Friese  
und  
Jasmin Knott

22.01.2024

# 1 Konzept

In dem Spiel „Murmelfuchs“ schlüpft der Spieler in die Rolle eines Fuchses, welcher aus dem Zoo auszubrechen versucht. Dabei kann der Spieler sich auf allen Vieren durch die Gehege navigieren. Möchte er ins nächste Gehege gelangen, rollt sich der Fuchs zusammen und rollt wie eine Kugel, eine Murmelbahn herunter. Wenn der Spieler jedoch fünfmal herunterfällt, hat er verloren und muss es nochmal probieren.

Bei „Murmelfuchs“ handelt es sich um ein Highscore Spiel. Der Spieler kann also mehrfach versuchen, über verschiedene Abzweigungen den schnellsten Weg aus dem Zoo zu finden.

## 2 Visuelles

### 2.1 Charaktermodell und Animationen

Das Modell und die Animationen für den Fuchs als Spieler-Charakter wurden aus dem Unity Asset Store verwendet. Dazu wurde ein eigener Animator erstellt, um sämtliche Animationen für das Laufen, Gehen, Rollen, sowie die Idle Animation miteinander zu verknüpfen. Dadurch kann aus jeder Animation in eine beliebige andere übergeleitet werden. Die Animationen werden über Parameter gesteuert, welche im Steuerungsscript des Charakters gesetzt werden.

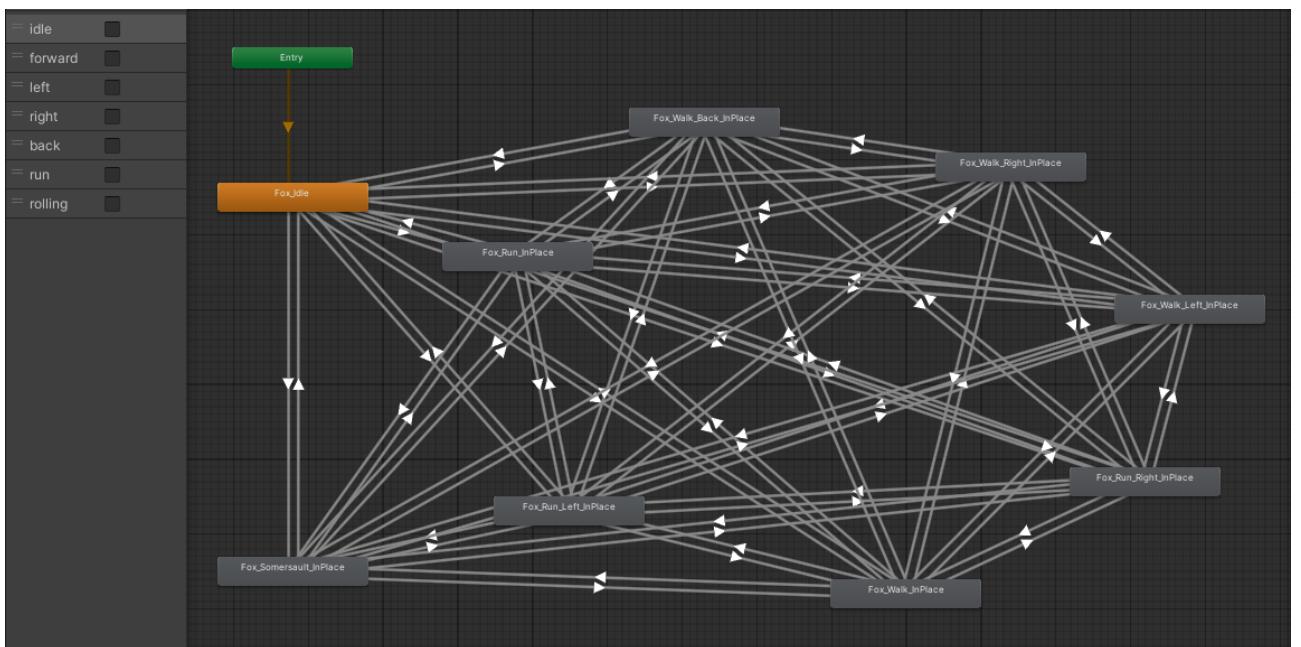


Abbildung 1 Animator

Auch die Modelle der restlichen Tiere wurden aus dem Unity Asset Store importiert. Jedoch wurden für diese, eigene Animationen angefertigt.

### 2.2 Level Design

Das Level des Spiels stellt einen Zoo dar, aus welchem der Fuchs fliehen möchte. Es ist so aufgebaut, dass es mehrere Gehege gibt, in welchen der Spieler herumlaufen und diese erkunden kann. Zwischen den Gehegen gibt es Murmelbahnen, welche verschiedene Abzweigungen und Untergründe haben. Um zum Ziel zu gelangen, muss man aus dem Anfangsgehege heraus, über die Murmelbahnen durch ein weiteres Gehege hindurch um so zur zweiten Murmelbahn zu gelangen, die einen dann in die Freiheit entlässt. Das Anfangsgehege ist ein relativ großes Gehege, in dem Tiger auf einem etwas bergigem Terrain hausen. Auf der anderen Seite befinden sich Katzen und

weitere Füchse in einem Gras bewachsenem Terrain. Dazwischen und davor befinden sich einige Hühner, die friedlich im Gras picken. Das zweite Gehege hat ein schneeiges Terrain, indem sich Pinguine und Hirsche befinden. Die Fläche zwischen den beiden, auf der die erste Murmelbahn endet und die Zweite anfängt, besteht aus rutschigem Eismaterial. Das Ziel befindet sich wieder in einer Grasumgebung. Dort wird der Spieler von zwei umliegenden Wäldern begrüßt, die nicht durch einen Zaun eingegrenzt sind. Etwas weiter hinten gibt es dann noch eine Pferdekoppel und daneben ein paar Picknicktische und einen Hund. Vor diesen liegt eine Schatztruhe, zu der der Spieler laufen muss, um das Spiel zu beenden. Gehege eins und zwei sind von einem Zaun umzäumt um zu symbolisieren, dass der Spieler gefangen ist und es nur einen möglichen Fluchtweg über die Murmelbahn gibt. Das Ziel ist bewusst nicht umzäunt um zu symbolisieren, dass der Spieler es geschafft hat und nun machen kann was er möchte. Am Anfang und Ende jeder Murmelbahn gibt es einen Checkpoint, bei welchem automatisch gespeichert wird.



Abbildung 2 Scene

## 2.3 Post Processing

Zur visuellen Nachbearbeitung des Spiels wurden fünf verschiedene Post Processing Effekte eingesetzt. Durch „Vignette“ und „Depth Of Field“ wird der Fokus des Spielers auf das Zentrum des Bildschirms geleitet. Mit „Ambient Occlusion“ erhält die Szene etwas realistischere Schatten im Bereich der Ecken und Kanten. Ein „Color Grading“ gibt dem Spiel eine etwas wärmere Farbe und macht das Bild angenehmer fürs Auge.

Der „Chromatic Aberration“ Effekt wurde variabel eingesetzt. Die Intensität dieses Effektes wird an die Geschwindigkeit des Spielers angepasst, wenn dieser auf der Murmelbahn rollt. Dadurch soll das Gefühl für die Bewegung intensiviert werden.

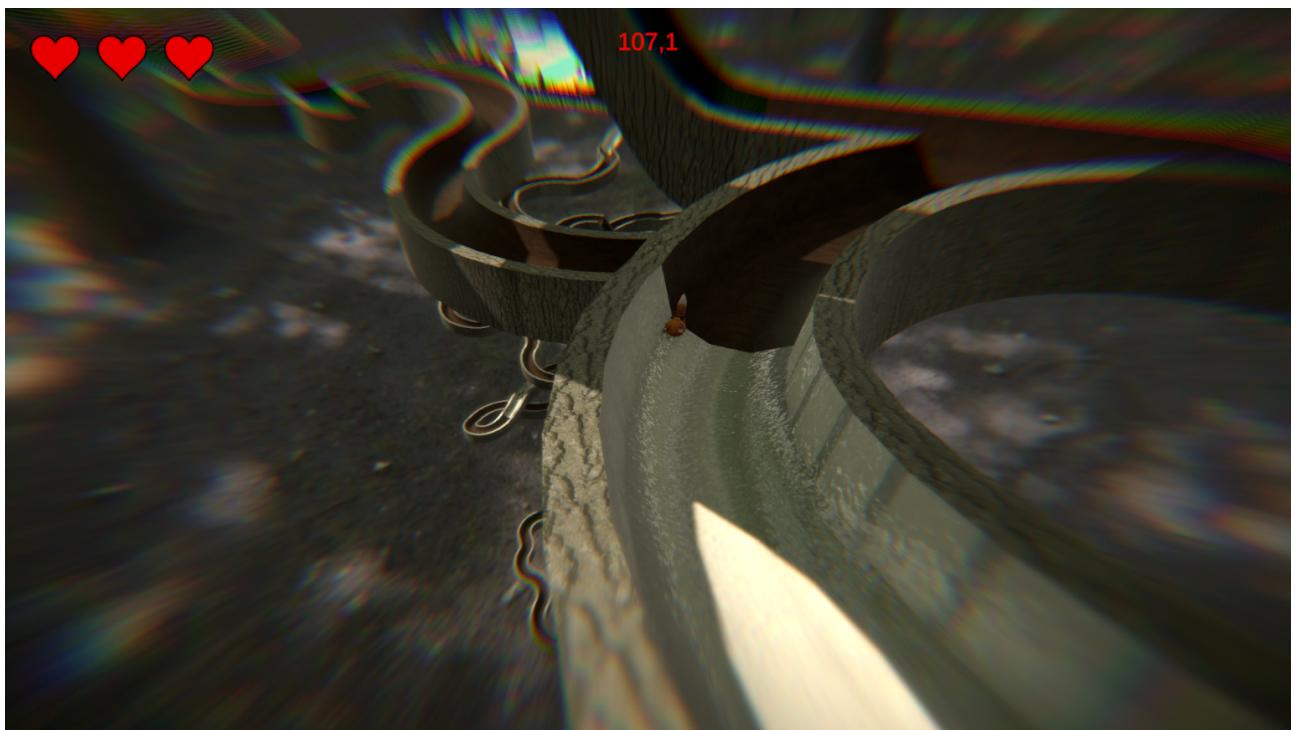


Abbildung 3 Chromatic Aberration

## 3 Funktionen / Technische Implementierung

### 3.1 Fuchs und Murmel

Eine zentrale Komponente in diesem Spiel ist es, dass sich der Fuchs in eine Murmel und zurück verwandeln kann. Dazu gibt es an den Ein- und Ausgängen der Murmelbahn unsichtbare Schranken, welche durch „Box Collider“ mit der „Is Trigger“ Eigenschaft implementiert wurden. Beim Auslösen der „OnTriggerEnter“ Methode, wird mit einem Tag geprüft, ob der Spieler die Methode ausgelöst hat und wenn dies der Fall ist, werden verschiedene Signale über einen Signalbus versendet.

Durch das Empfangen der Signale wird der Collider des Fuchses deaktiviert und der Collider der Murmel aktiviert (oder andersherum). Außerdem ändert sich der Animationsstatus des Fuchses, die Musik wird gewechselt und ein Checkpoint wird gespeichert.



Abbildung 4 Fuchs stehend



Abbildung 5 Fuchs rollend

### 3.2 Menü

Es gibt ein fast identisches Menü beim Spielstart, als auch im Spiel. In beiden Fällen kann die Highscore-Liste angezeigt, Einstellungen getroffen, oder das Spiel beendet werden. Im Hauptmenü kann zusätzlich ein neues Spiel gestartet, oder der letzte Spielstand fortgesetzt werden. Im Spiel kann lediglich das aktuelle Spiel fortgesetzt werden.

In den Einstellungen kann der Spieler Bildschirmauflösung und Bildmodus ändern, sowie die Musik- und Soundlautstärke ändern. Beim Schließen der Einstellungen wird automatisch gespeichert.

Im Spiel kann das Menü mit „Escape“ geöffnet und geschlossen werden. Wenn das Menü geöffnet ist, steht das Spiel still und wird etwas abgedunkelt.

### 3.3 Physics

Auf der Murmelbahn wurden verschiedene „Physic Materials“ verwendet. Das Standardmaterial der Rampe hat eine geringe „Dynamic Friction“ und sehr geringe „Static Friction“. Dadurch rollt die Murmel schnell über die Bahn und es ist leichter sie wieder zu beschleunigen, falls der Spieler einmal ausgebremst wurde. Außerdem ist die „Bounciness“ ebenfalls sehr gering, dadurch titscht der Charaktere nicht zu sehr auf der Strecke, da er sonst leicht aus der Bahn fliegen kann.

Das Grasmaterial hat dieselben Eigenschaften für „Dynamic Friction“ und „Static Friction“, wie das Standardmaterial, doch die „Bounciness“ ist auf 0. Das Material haben wir für einige Kurven und Stellen verwendete, an denen wir nicht wollen, dass der Spieler von der Bahn titscht.

Dazu haben wir ein Eismaterial, welches eine „Static Friction“ und „Bounciness“ von 0 hat, dazu ist die „Dynamic Friction“ geringer, als beim Standardmaterial. Auf diesem Material rollt der Spieler schneller und einfacher die Bahn entlang. Wir haben dieses Material für Hügel und Sprungschanzen verwendetet, wo der Spieler viel Schwung benötigt.

Zuletzt haben wir noch ein Erdmaterial, welches eine hohe „Dynamic Friction“ und „Static Friction“ besitzt. Auch hier ist die „Bounciness“ auf 0, dieses Material verwenden wir, um den Spieler an bestimmten Stellen auszubremsen.

Damit der Spieler die physikalischen Änderungen sehen und sich darauf einstellen kann, hat die Murmelbahn zu jedem „Physic Material“ an den Stellen auch eine anders farbige Textur.

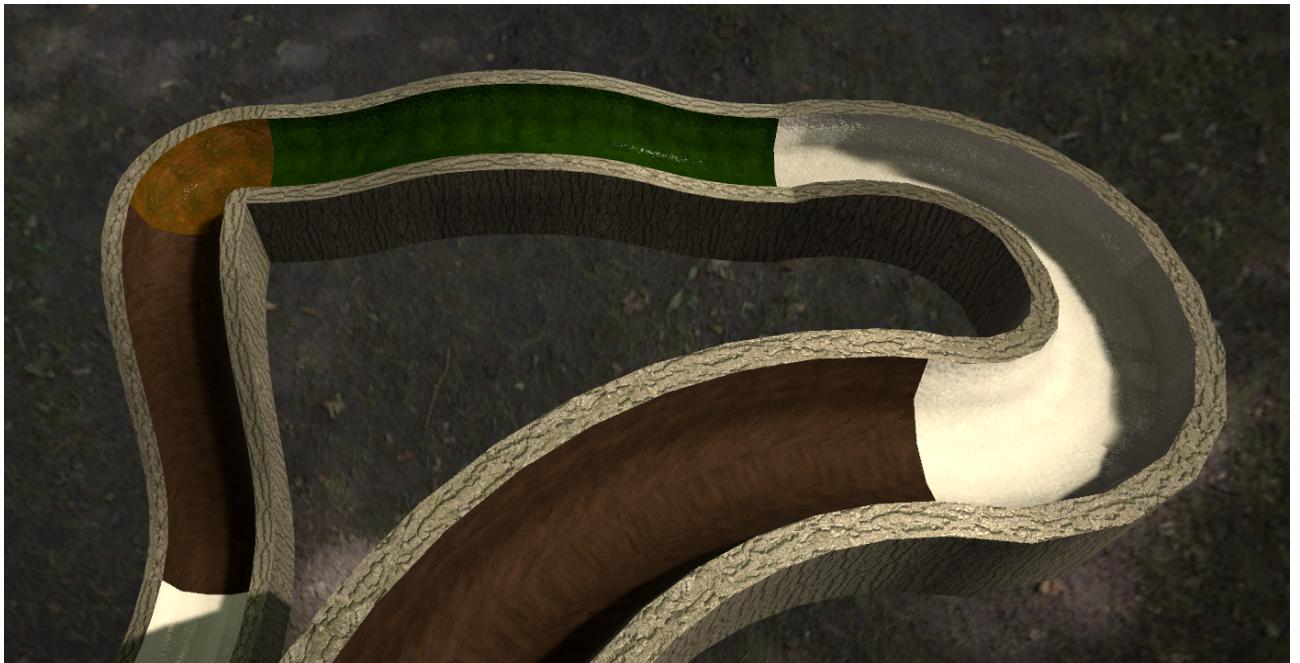


Abbildung 6 Visualisierung von verschiedenen Physics durch Materialien

### 3.4 Scoreboard

Das Spiel startet mit einem drei Sekunden Countdown, an dessen Ende ein Timer beginnt hoch zu zählen. Wenn der Spieler das Ziel erreicht, wird dieser Timer gestoppt und die Zeit im Highscore eingetragen. Für die Implementierung des Highscores haben wir eine statische Klasse verwendet, welche eine float Liste verwaltet. Wenn eine neue Zeit eingetragen wurde, wird überprüft, ob es mehr als 10 Highscores gibt, in dem Fall wird der schlechteste Highscore gelöscht. Die Zeit wird dabei immer in Sekunden angegeben.

### 3.5 Speichern und Laden

Das Spiel speichert immer automatisch, wenn die Einstellungen geschlossen werden, der Spieler einen Checkpoint, oder das Ziel erreicht und wenn er das Spiel beendet. Es werden immer alle Einstellungen, sowie die Highscores gespeichert und ob es einen Spielstand gibt, oder nicht. Falls der Spieler sich im Spiel befindet, werden auch Leben, aktuelle Zeit, Spawn Punkt und Spawn Rotation, sowie ob der Fuchs eine Murmel ist oder nicht gespeichert. Falls der Spieler sich beim Speichern im Hauptmenü befindet, wird überprüft, ob es einen alten Spielstand gibt, von welchem die eben genannten Variablen übernommen werden können. Wenn der Spieler das Ziel erreicht wird gespeichert, dass es keinen alten Spielstand mehr gibt.

Beim Öffnen des Spiels wird der Speicherstand geladen. Dabei werden alle Einstellungen und die Highscores übernommen, falls es einen Spielstand gibt werden die entsprechenden Daten in einem GameState Object abgespeichert. Dieses GameState Object implementiert das Singleton Pattern und nutzt die „DontDestroyOnLoad“ Funktion, sodass es nur einen GameState gibt, welcher beim Szenenwechsel bestehen bleibt. Beim Fortführen des Spielstandes werden die Variablen über Signale an entsprechende Scripte versendet.

## **3.6 Verwendete Patterns**

### **3.6.1 Signalbus**

Wir haben das Signalbus Pattern aus der Vorlesung in zahlreichen Fällen verwendet. Mit 11 verschiedenen Signalen verschicken wir Daten und lösen verschiedene Methoden aus. Wir fanden das Pattern äußerst nützlich, um unzählige Verknüpfungen zu vermeiden, welche schnell zu „Spaghetti-Code“ führen können. Von den bereits zuvor erwähnten Anwendungen abgesehen, nutzen wir das Pattern in unserem Spiel beispielsweise für das Wechseln von Musik und Abspielen von Soundeffekten, sowie den Tod des Spielers.

### **3.6.2 Singleton**

Das Singleton Pattern haben wir verwendet, um beispielsweise den MusicPlayer einmalig zu machen. Der MusicPlayer wird beim Öffnen des Spiels im Hauptmenü erzeugt und spielt sofort Musik ab. Damit die Musik beim Szenenwechsel nicht abbricht, oder mehrfach gespielt wird, wenn das Hauptmenü erneut geladen wird, haben wir ein Singleton MusicPlayer mit „DontDestroyOnLoad“ Funktion.

## **4 Audio**

Das Spiel wird durch fünf verschiedene Musikstücke untermauert, welche alle über eine „Audio Source“ abgespielt werden. Das Hauptmenü ist mit einem eher idyllischen Lied untermauert. Wenn der Spieler als Fuchs unterwegs ist, wird ein Musikstück abgespielt, welches abenteuerlich ist und zum Erkunden anregt. Auf der Murmelbahn wird ein schnelles und aufregendes Lied gespielt, um das Geschehen zu betonen. Außerdem wird ein passendes kurzes Lied abgespielt, wenn der Spieler gewonnen, oder verloren hat.

Die vier verschiedenen Soundeffekte werden über eine weitere „Audio Source“ abgespielt. Es gibt jeweils einen Soundeffekt für das Klicken eines UI-Buttons, das Erreichen eines Checkpoints und wenn der Spieler herunterfällt und damit stirbt. Des Weiteren wird ein Kollisionssound abgespielt, wenn der Spieler als Murmel auf einen Collider stößt.

Das Wechseln und abspielen der Lieder und Soundeffekt wird durch ein Signal ausgelöst.

# **Quellen**

## **3D Assets für Level Design:**

<https://www.kenney.nl/assets/fantasy-town-kit>

<https://www.kenney.nl/assets/marble-kit>

<https://www.kenney.nl/assets/minigolf-kit>

<https://www.kenney.nl/assets/platformer-kit>

<https://www.kenney.nl/assets/survival-kit>

## **Materials:**

<https://ambientcg.com/view?id=Snow006>

<https://ambientcg.com/view?id=Wood062>

<https://ambientcg.com/view?id=Moss003>

<https://ambientcg.com/view?id=Ground068>

<https://ambientcg.com/view?id=Bark001>

## **Fuchs:**

<https://assetstore.unity.com/packages/3d/characters/animals/toon-fox-183005>

## **Tiere:**

<https://assetstore.unity.com/packages/3d/characters/animals/animals-free-260727>

## **Audio:**

<https://pixabay.com/sound-effects/gewonnen2-7000/>

<https://pixabay.com/sound-effects/violin-lose-5-185126/>

<https://pixabay.com/sound-effects/violin-win-5-185128/>

<https://www.kenney.nl/assets/impact-sounds>

<https://www.kenney.nl/assets/interface-sounds>

<https://freepbr.com/materials/stylized-grass1/>