Hochschule Luzern
Gummiband-Physik in Games

Lehrperson:
Dragica Kahlina
dragica.kahlina@hslu.ch
Sebastian Hollstein
sebastian@studio-fizbin.de

Student:
Florian Etgeton
Kalcheggweg 23
3006 Bern
florian.etgeton@stud.hslu.ch
Digital Ideation, 5. Semester

Anzahl Zeichen: 6’825, Anzahl Wörter: 1’069, 03. Oktober 2019

Inhaltsverzeichnis

Einleitung

Da ich in meiner kommenden Bachelorarbeit voraussichtlich eine Art Gummiband-Mechanik zur Fortbewegung einbauen möchte, habe ich mich im Laufe dieses Moduls mit Federn und Gummibändern auseinandergesetzt. Da sich Gummibänder ähnlich wie Federn verhalten, habe ich mich zuerst mit den Physikalischen Gesetzmässigkeiten von Federn beschäftigt. Anschliessend habe ich das Ganze in Unity implementiert und mit verschiedenen Werten getestet und eine spielbare Version erstellt.

Theorie[1]

Eine Feder ist ein Objekt, das durch eine Kraft verformt werden kann und nach Wegnehmen der Kraft in seine ursprüngliche Form zurückkehrt.

Bei der Untersuchung von Federn und Elastizität bemerkte der Physiker Robert Hooke, dass die Spannungs-Dehnungs-Kurve für viele Materialien einen linearen Bereich aufweisen. Innerhalb bestimmter Grenzen ist die Kraft, die erforderlich ist um einen elastischen Gegenstand wie eine Metallfeder zu dehnen, direkt proportional zur Ausdehnung der Feder. Dies ist als Hookesches Gesetz bekannt und wird allgemein geschrieben:

$$F = k \cdot \Delta x$$

Wobei *F* die Kraft ist, *Δx* die Länge der Ausdehnung/Kompression ist und *k* eine Konstante ist, die als Federkonstante bekannt ist und üblicherweise in *N/m* angegeben wird.

Da in Unity bereits eine Physik-Engine eingebaut ist, ist dies alles was wir zur Berechnung einer Federkraft benötigen. Da sich Gummibänder nicht komplett linear verhalten (Abbildung 1: OPQ: Ausdehnung; QRO: Kontraktion), braucht es noch eine Kurve welche die Federkonstante je nach ausdehnungslänge modifiziert.

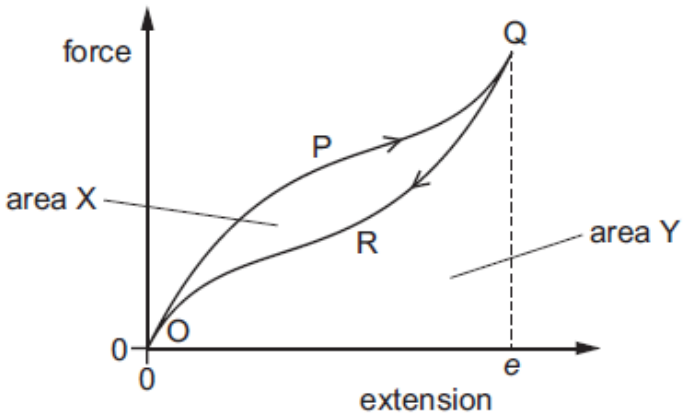

Abbildung 1: OPQ: Ausdehnung; QRO: Kontraktion[2]

1 https://www.khanacademy.org/science/physics/work-and-energy/hookes-law/a/what-is-hookes-law
2 http://physics-ref.blogspot.com/2019/05/a-rubber-band-is-stretched-and-then.html

Umsetzung

Einfache Federmechanik

Als ersten Schritt habe ich versucht eine einfach Federmechanik in Unity umzusetzen und habe diese anschliessend mit einer Kurve erweitert. Ein UI-Element für Kurven ist bereits in Unity implementiert und heisst *AnimationCurve*. Standartmässig ist diese eine Gerade mit Wert 1. Damit die Spielfigur von der Wand abprallt ist ausserdem ein *PysicsMaterial* notwendig. Diesem kann man eine gewünschte Bounciness angeben. Damit die beschleunigte Spielfigur über die Zeit wieder abgebremst wird, kann auf dem *Rigidbody2D* einen *Liner Drag* gesetzt werden.

Abbildung 2: Properties des Gumeli Handler Skript

Character: Spielfigur, die der Spieler*in steuert.
Pushpin: Startpunkt, von wo das Gummiband hin zum Spieler*in hin aufgespannt wird.
Spring Constante: Federkonstante, welche die Stärke der Feder angibt
Rubber Band Length: Länge des Gummibandes im ungedehnten Zustand
Rubber Band Max Stretch Length: Maximale ausdehnungslänge des Gummibandes
Stretch Curve: Verhalten des Zusammenziehens. Wert unter 1 bedeutet langsamer, alles darüber schneller.

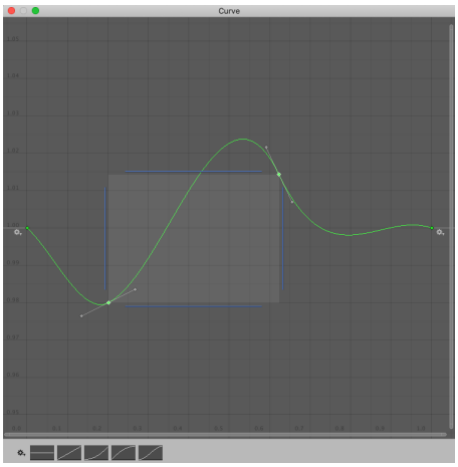

Abbildung 3: Kurve bearbeiten

Code

```
float distance = Vector2.Distance(character.transform.position,
pushpin.transform.position);
float curveValue = stretchCurve.Evaluate(Mathf.InverseLerp(rubberBandMaxStretchLenght,
rubberBandLenght, distance));
float force = springConstante * (distance - rubberBandLenght) * curveValue;
Vector2 dir = new Vector2(pushpin.transform.position.x - character.transform.position.x,
pushpin.transform.position.y - character.transform.position.y);
rb.AddForce(dir.normalized * force);
```

Findings

Ich war überrascht wie einfach das Ganze in einer Gameengine wie Unity implementiert werden kann, da diese bereits viele Teile der Berechnung übernimmt.

Weiter habe ich gemerkt, dass die *Stretch Curve* in meinem Fall praktisch keinen Einfluss hat und diese bei einer möglichen Umsetzung weggelassen werden könnte. Grund dafür ist, da sich mein Gumiband sehr schnell zusammenziehen soll. Somit bemerkt man die Kurve kaum. Nur bei einer kleinen Federkonstante, das heisst wenn sich das Gummiband langsam zusammenzieht, ist die Kurve bemerkbar.

Viel wichtiger sind Werte wie *Linear Drag, Federkonstante* oder *Bounciness*, welche ich im nächsten Kapitel weiter untersuchen werde.

Steuerung der Spielfigur

Um besser nachvollziehen zu können wie sich das Gummiband in einer Spiel Situation anfühlt, habe ich das Ganze mit einer Charaktersteuerung und der Möglichkeit ein Gummiband manuell aufziehen zu können erweitert. Als nächsten Schritt implementierte ich noch die Collider für das Gummiband, damit der Ball beim Aufspannen abgewehrt werden kann. Dies war etwas zeitaufwendig, da ich die Struktur des Charakters anpassen musste, damit die beiden Collider von Gummiband und Spielfigur auf dem gleichen Gameobject sind. Ansonsten würden sie miteinander kollidieren und sich gegenseitig wegstossen. Weiter musste ich manuell die Start- und Endpositionen des Gummibandes berechnen, da sich diese zu jeder Zeit ändern können.

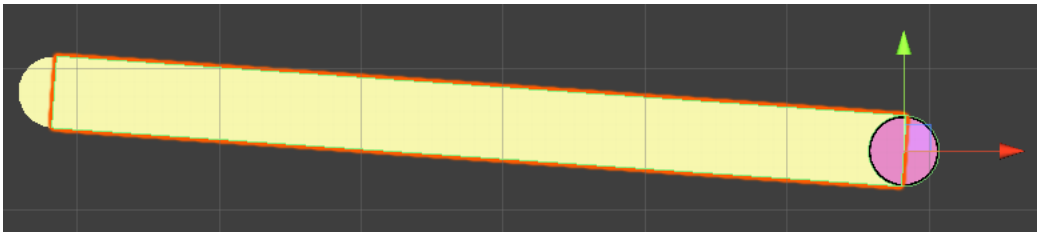

Abbildung 4: Orange: Collider des Gummibandes; Pink: Spielfigur

Um das ganze nun besser testen zu können und lassen habe ich noch ein kleines UI mit den wichtigsten Werten hinzugefügt, welche über den Controller verändert werden können.

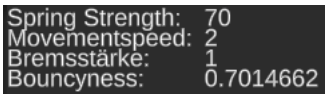

Abbildung 5: UI mit wichtigen Werten zum verstellen

Findings

Collisions und Rigidbodies sind ein sehr grosses und umfangreiches Thema. Da die Spielfigur mit sehr hoher Geschwindigkeit nach vorne katapultiert werden kann, ist es besser eine konturierende[3] Kollisionsberechnungsmethode zu verwenden. Diese sorgt dafür, dass bei hohen Geschwindigkeiten keine getroffenen Collider «übersprungen» werden. Dies hat jedoch einen negativen Einfluss auf die Performance.

Dabei hat sich ein weiteres Problem herauskristallisiert. Da sich die Collider langsamer updaten als die Spielfigur schnell ist, stösst der Collider des Gummibandes über die Pushpin Position hinaus und stösst den Ball ungewollt an. Dieses Phänomen kann minimiert werden, indem man den *Fixed Timestep*[4] verkleinert. Dieser definiert das

3 https://docs.unity3d.com/ScriptReference/CollisionDetectionMode.Continuous.html
4 https://docs.unity3d.com/Manual/class-TimeManager.html

Intervall der Physik Kalkulation. Dies sollte aber auch nur mit Bedacht angepasst werden, da auch hier mit Performanceeinbussen zu rechnen ist.

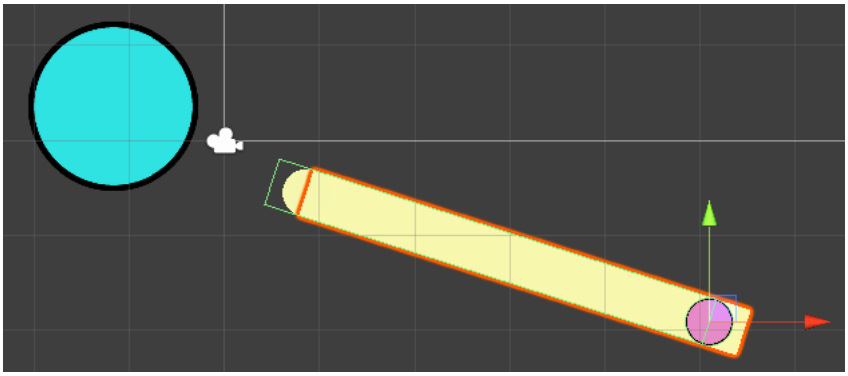

Abbildung 6: "Ghosting" des Gummiband Colliders

2 Spieler Modus

Da es immer noch relative schwierig ist, die ganze Implementation zu testen, habe ich den momentanen Prototyp mit einem zweiten Spieler, Goals und einer Score Anzeige erweitert. Somit fällt es etwas einfacher die finale Idee zu testen.

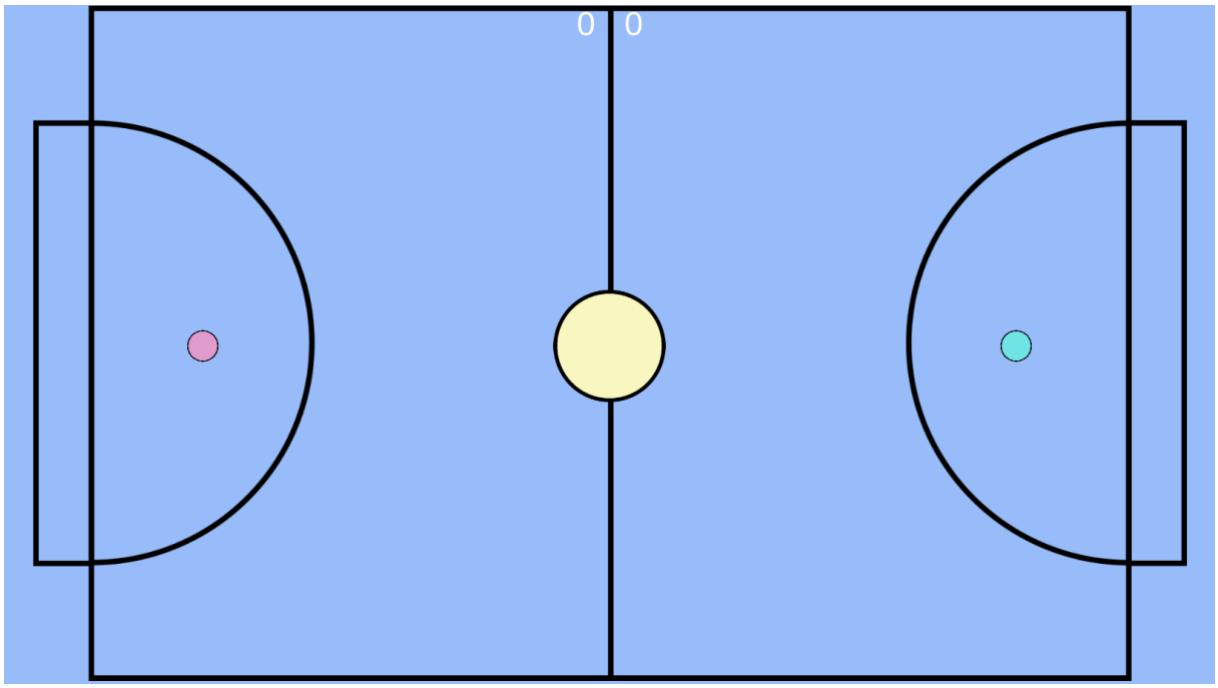

Abbildung 7: 2-Spieler Modus

Findings

Bis auf wenige Bugs, hat es eigentlich relativ gut funktioniert und die Testpersonen hatte Spass beim Spielen, was ja eigentlich das Wichtigste ist.Neben dem Tweaking der Mechaniken muss ich sicherlich noch an besserem UX arbeiten, da oft nicht klar ist in welchem State sich der Spieler befindet. Dies könnte man beispielsweise durch verschiedene Farben, Particles oder Shader verbessern.

Conclusion & Future

Ich habe bereits bei den ersten Umsetzungen relativ rasch bemerkt, dass mir das ewige Ausprobieren mit verschiedenen Werten nicht so liegt. Ich bin da wohl viel zu

ungeduldig und möchte lieber gleich das nächste Feature implementieren, um den Prototypen weiterzubringen.

Wie bereits erwähnt, müsste ich sicherlich noch viel an den Mechaniken schrauben und einzelne Bugs beheben und mit Hilfe von besserem UX die States der Spielfigur veranschaulichen.

Bezüglich meiner Bachelorarbeit bin ich wieder etwas zuversichtlicher, obwohl ich mir immer noch nicht 100% sicher bin, ob ich als Abschlussarbeit wirklich so ein Physik-lastiges Spiel machen möchte.

Abbildungsverzeichnis