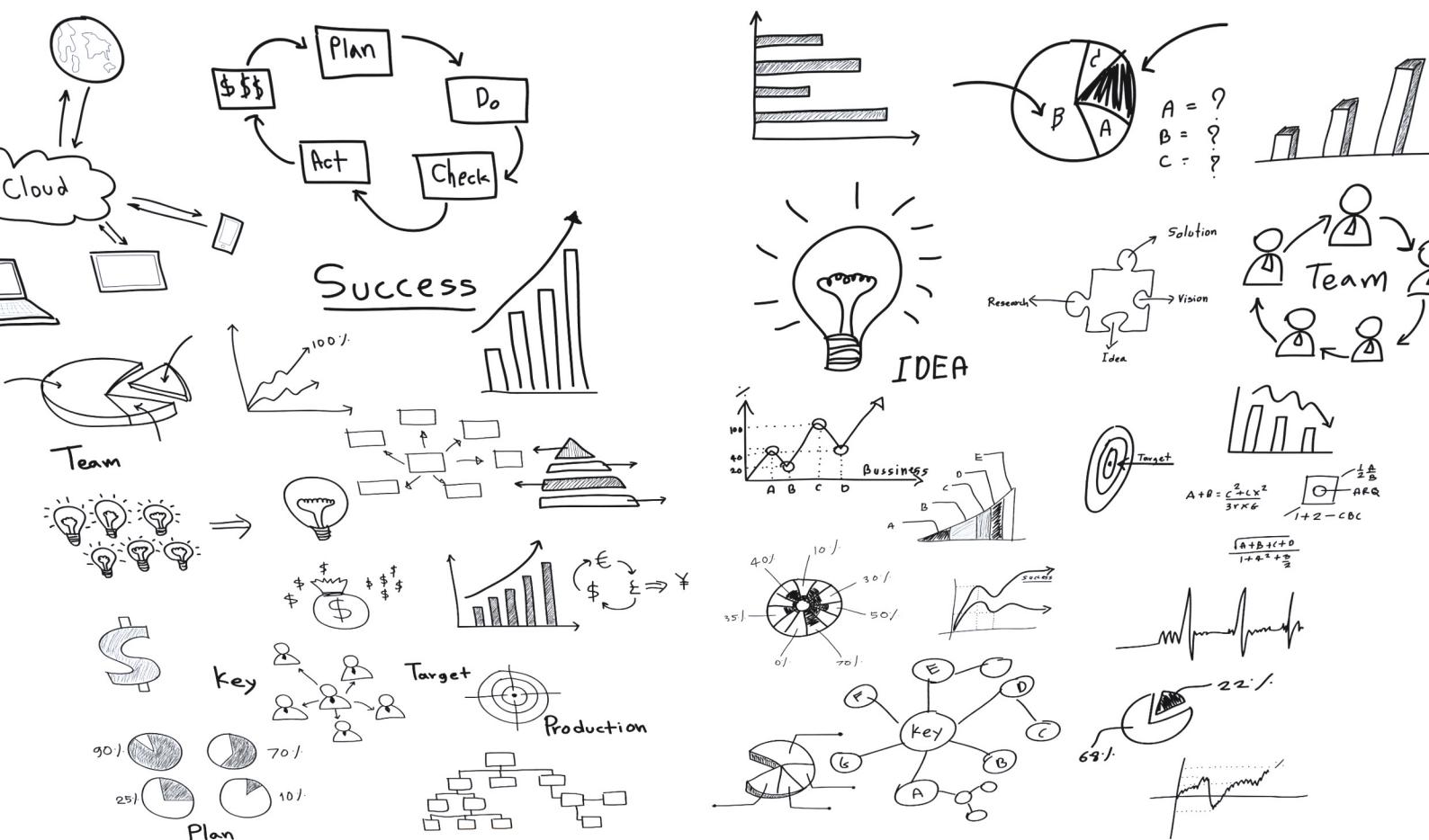


Mathematik & Physik Modul

Lernjournal

Fabian Grossenbacher



Vorwort

Mathematik Kenntnisse

Seit meinem letzten richtigen Mathematik Unterricht ist es ca. 6 Jahre her. In der Schule hat mir Mathematik gar kein Spass gemacht, da ich kein Anwendungsbedarf dafür hatte.

Mittlerweile, durch das Interesse der Spielentwicklung und allgemein wie das Universum funktioniert, macht mir Mathematik um einiges mehr Spass und ist ein interessantes Thema für mich. Am meisten begeistert mich das praktisch alles im Universum Mathematik ist.

Trotzdem fehlt mir leider viel Schulstoff den ich Nachholen musste und mir ab und zu auch ein wenig Probleme bereiten wird in diesem Modul.

Unterricht Mathematik und Physik

Wir haben das Glück einen sehr tollen Lehrer erhalten zu haben. Andy erklärt die Themen der Mathematik zwar ziemlich schnell, dafür meistens ziemlich verständlich.

Mir persönlich hat es ziemlich geholfen wenn ich bereits am Vortag zu den aufkommenden Themen im Unterricht, auf eigene Faust recherchiert habe. So konnte ich wichtige Fragen und Unklarheiten direkt ansprechen und lief weniger die Gefahr während dem Unterricht den Faden zu verlieren und nicht mehr hinterher zu kommen.

Schwierigkeiten

Am meisten Mühe bereitet mir der Zeitdruck. Die Mathematik hinter Systemen zu verstehen fällt mir um einiges leichter als ich erwartet hatte. Beim implementieren habe ich manchmal noch etwas mühe, jedoch bessert sich das auch.

Cellular Automata

Game of Life

Conway's Game of Life ist eigentlich ziemlich einfach zu verstehen. Wir können mit Hilfe eines 1D, 2D oder sogar 3D Grid ein Lebensraum für sogenannte Zellen erschaffen. Jede Zelle hat eine eigene Koordinate auf diesem Grid.

Wir können jetzt durch das Array iterieren und bei jeder einzelnen Zelle gewisse Bedingungen abfragen. Diese können ganz simpel sein, aber auch ziemlich komplex werden.

Wenn wir 1x durch das ganze Array iteriert sind, sprechen wir von der ersten Generation. Bei einem weiteren mal, von der zweiten Generation etc. Wir können die Anzahl der Generation selber bestimmen und so unterschiedliche Endresultate bei der Zellenanordnung erreichen.

Beispiel:

```
If(cell.neighbourHood.Count > 2
```

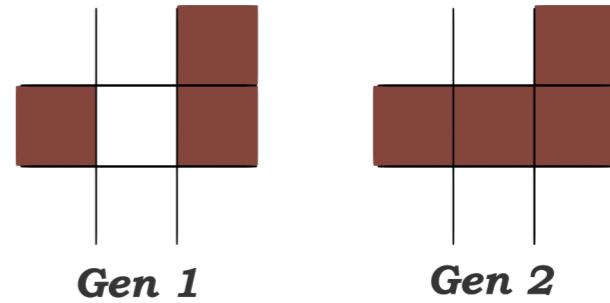
```
{
```

```
    Cell.SetActive(true)
```

```
}
```

Die Zelle wird also in Generation 1 überprüft & geändert und ist bei der 2 Generation nicht leer.

Zelle mit 3 Nachbarn



Wenn wir 1x durch das ganze Array iteriert sind, sprechen wir von der ersten Generation. Bei einem weiteren mal, von der zweiten Generation etc. Wir können die Anzahl der Generation selber bestimmen und so unterschiedliche Endresultate bei der Zellenanordnung erreichen

Regeln

Für Conways Game of Life gibt es eine ganze Liste von Regeln. Die Regel 30 ist dabei eine besondere.

Implementation

Obwohl das Prinzip der Zellulären Automaten sehr simpel ist, habe ich zu Beginn bei der Implementation etwas Mühe gehabt. Das hat denke ich damit zu tun, dass ich bis jetzt for Schleifen viel zu wenig verwendet habe & ich wusste nicht genau wie ich die Bedingungen aufzustellen soll, dass sich etwas komplexeres wie z.B Feuer, oder Wasser aus einem solch simplen System ergibt.

Am meisten Probleme habe ich jedoch beim Herausfinden der Nachbarschaft. Die Benachbarten Zellen links, rechts, oben & unten, sind nicht gross ein Problem, aber die diagonal Benachbarten Zellen haben mir Mühe bereitet.

Ich habe mich deshalb mit meinen Klassenkameraden ausgetauscht und habe bemerkt, dass es noch andere Möglichkeiten gibt die Benachbarten Zellen herauszufinden. Wie z.B mit Layer-Masks.

Fragen, Theorien & Unklarheiten.

Ich denke dass das Feuer System von Zelda (so wie es auf dem kurzen Clip gezeigt wurde), mit einem 2D Array gemacht wurde, und dann Prefabs von Feuer VFX aktiviert oder deaktiviert werden. So könnte ziemlich simpel eine Ausbreitung des Feuers in einem kleineren lokalen Bereich simuliert werden.

Welches ist die performanteste Methode um die Nachbarschaft herauszufinden?

Gibt's es performantere oder übersichtlichere Wege um die Regeln zu definieren und implementieren?

Physik Anpassung

Idee

Ich wusste lange nicht was ich machen sollte für diese Aufgabe. Schlussendlich habe ich mich für eine Wasserauftrieb Mechanik entschieden, welche den GameObjekten Auftrieb an spezifischen Punkten verleiht.

Meer und Wetter

Um mir enorm viel Zeit zu sparen habe ich mich an 2 Assets bedient & um die maximale Power herauszuholen habe ich das Projekt auf die Universal Render Pipeline umgestellt. Die Assets lassen sich gut miteinander verknüpfen und bieten maximale Kontrolle über das Wetter und Meer.

Auftriebskraftsformel

Alle Objekte auf der Welt werden von der Gravitationskraft angezogen. Die Gravitationskraft beträgt 9,81 und bringt somit Objekte zu Fall. Wieso also nicht auch im Wasser.

Die Auftriebskraftsformel besagt, dass wenn die Auftriebskraft grösser als die Gewichtskraft ist, dann steigt der Körper nach oben. Anders rum sinkt der Körper auf den Boden. Wenn beide im Gleichgewicht sind, dann schwimmt der Körper in der Flüssigkeit.

$$\mathbf{F}_A = \mathbf{F}_G = \rho \cdot V \cdot g \quad < \quad \mathbf{F}_{G,K} = \rho_K \cdot V \cdot g$$

Verbesserungspotenzial

Bis jetzt habe ich die Auftriebskraftsformel so genutzt, dass der Meeresspiegel konstant 0 ist. Das mag auf einem See ziemlich gut funktionieren, ich möchte dass mein Wasser jedoch ein Meer ist.

Ich denke es würde viel mehr Leben in die sonst schon tolle Mechanik bringen, wenn man die Auftriebskraft noch an die Wellen anpassen könnte.



Vektorfelder

Segel System Intro

Bei den Vektorfeldern wollte ich eigentlich etwas einfacheres machen. Geplant war ein einfaches Windsystem welches eine AddForce() auf den Rigidbody gibt.

Bei der Implementierung habe ich etwas mit dem Cloth Component herumgespielt und ehe ich mich versah, habe ich mich mitreissen lassen und möchte jetzt ein komplexeres Physikbasiertes Segelsystem aufsetzen.

Ich war bereits 2 Wochen auf einem Segelschiff in Griechenland und habe danach 1 Woche die Windsurf Ausbildung absolviert..

Research

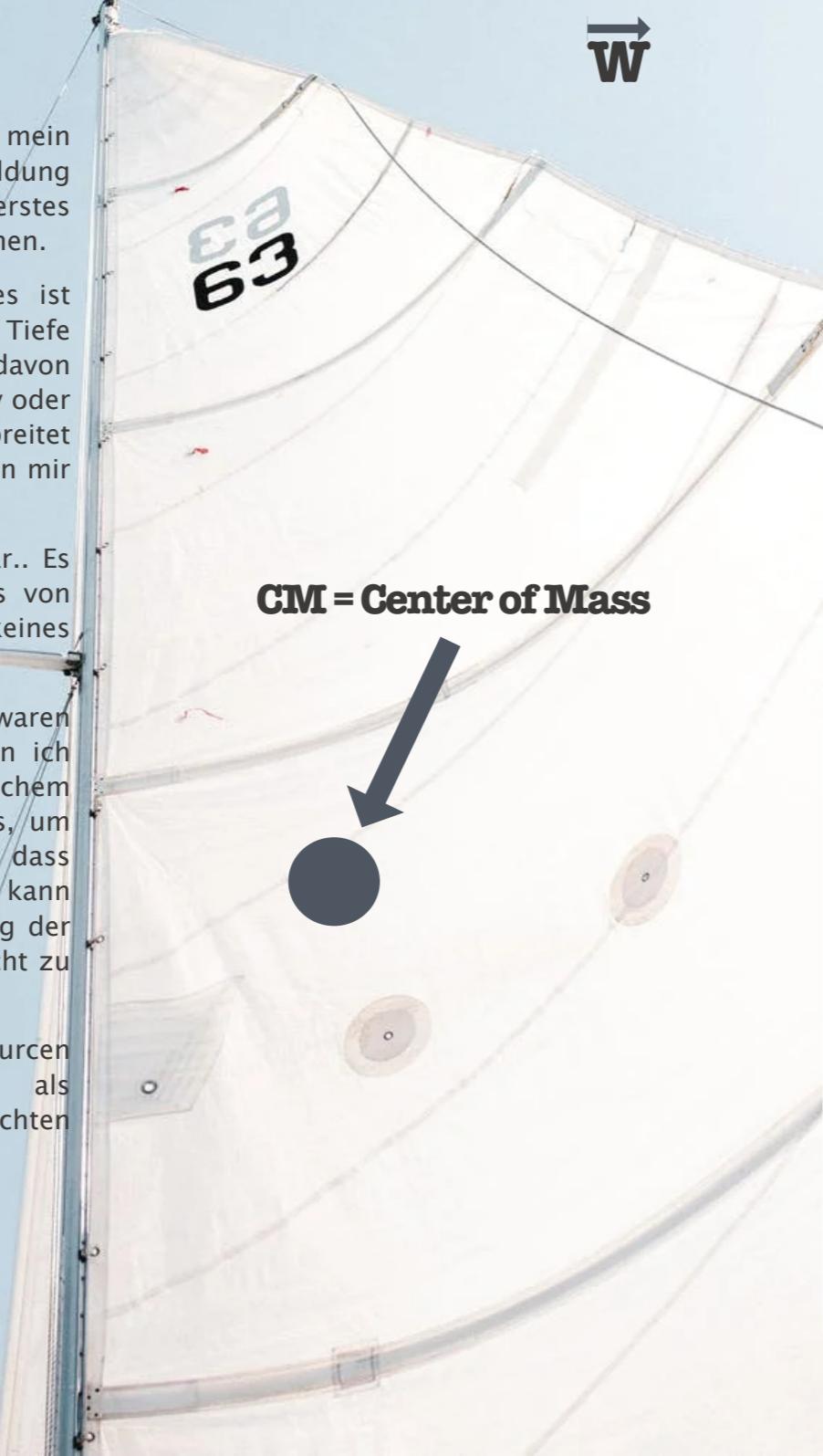
Es ist bereits 5 Jahre her seit ich mein Zertifikat für die Windsurf Ausbildung erhalten habe. Ich musste also als erstes mein Wissen wieder etwas auffrischen.

Ich habe bemerkt wie wichtig es ist solche Systeme in der Tiefe anzuschauen und wie viel man davon lernen kann. Begriffe wie Lee & Luv oder Backbord die im Segeln weit verbreitet und ziemlich essenziell sind, helfen mir enorm bei der Implementierung.

Mein Hauptsächliches Problem war.. Es gibt ein paar vereinzelte Devlogs von Segel Systemen in Unity, aber keines davon ausführlich genug.

Die meisten Videos auf YouTube waren von einfachen Seglern, von denen ich zwar super gelernt habe in welchem Winkel ich mein Segel halten muss, um gegen den Wind zu segeln oder dass man schneller als der Wind segeln kann etc. Aber eine komplette Erklärung der Segelphysik war nicht ganz so leicht zu finden.

Ich habe hier die wichtigsten Ressourcen zusammengefasst, welche ich als Guideline bei der Entwicklung beachten kann.



Die wichtigsten Kräfte im Überblick

Scheinbarer Wind

Windauftriebs- und Widerstandskräfte

Auftriebskraft des Wassers

Rumpfviskoser Druckwiderstand

Reibungswiderstand des Rumpfes

Kielhub- und Widerstandskräfte

Wellenwiderstand

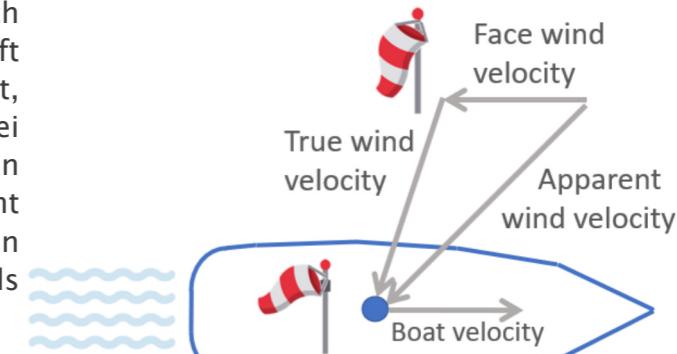
Steuerruderauftrieb und Widerstandskräfte

Luftwiderstand des Rumpfes

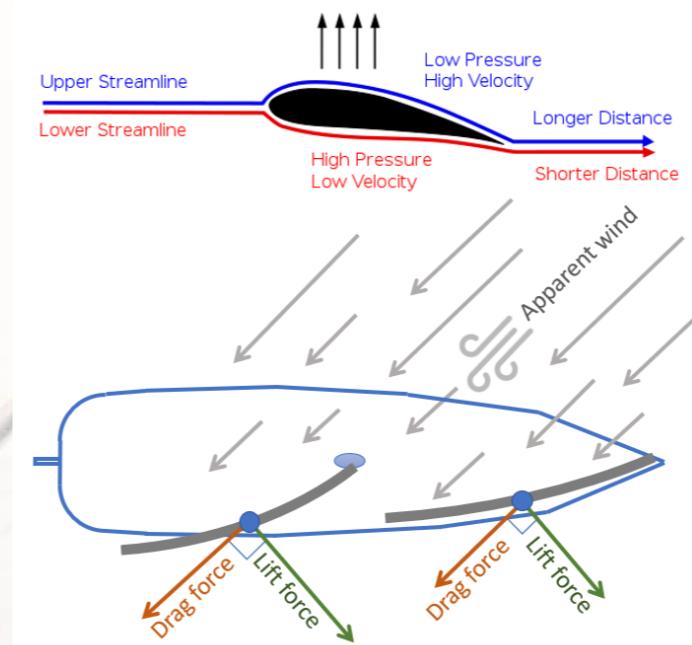
Seitlicher Widerstand des Rumpfes

Scheinbarer Wind

Wenn du ein Cabrio fährst, spürst du, wie dir der Wind ins Gesicht weht. Dieser Wind wird durch die Autobewegung gegen stehende Luft verursacht. Wenn Seitenwind von links kommt, fühlst du es wie Front-Links-Wind. Denn zwei Winde vereinen sich und bilden einen neuen Windvektor. Dasselbe passiert, wenn die Yacht schnell fährt. Zwei Winde bilden einen neuen Windgeschwindigkeitsvektor, der als scheinbarer Wind bezeichnet wird.



Es ist sehr einfach, den scheinbaren Windvektor in Unity zu berechnen, da die Vector3-Klasse, die Sie zur Darstellung von Wind- und Bootsgeschwindigkeiten verwenden können, Vektoroperationen unterstützt.



Windauftriebs & Widerstandskräfte

Die Flüssigkeit oder Luft, die um eine Oberfläche strömt, erzeugt zwei Kräfte, Auftrieb und Widerstand. Die Segel-Auftriebskraft ist ähnlich wie die Flügel-Auftriebskraft und steht immer senkrecht zur Windrichtung. Der Auftrieb wird durch unterschiedliche Drücke auf gegenüberliegenden Seiten einer Oberfläche hervorgerufen. Der Auftrieb wird immer von einer Widerstandskraft begleitet, welche die Komponente der Oberflächenwiderstandskraft parallel zur Wind-/Strömungsrichtung ist.

Bernoulli Gleichung

Auftriebs- und Widerstandskräfte können mit der berühmten Bernoulli-Gleichung berechnet werden, die in der Strömungsmechanik weit verbreitet ist.

$$F_{Lift} = C_L \rho A \frac{v^2}{2}$$

$$F_{Drag} = C_D \rho A \frac{v^2}{2}$$

C_L = lift coefficient
 C_D = drag coefficient
 A = sail area in M^2
 v = velocity
 ρ = air density

Auftriebs- und Widerstandskoeffizienten

Der Luftwiderstandsbeiwert wird verwendet, um den Widerstand eines Objekts in einer flüssigen Umgebung wie Luft oder Wasser zu quantifizieren. Der Auftrieb ist eine Kraft, die durch Körper und Flüssigkeit oder Luft erzeugt wird (verursacht durch unterschiedliche Luftdichte an den oberen und unteren Flügelseiten). Das drückt den Körper senkrecht nach oben zur Windrichtung. Beide Koeffizienten werden experimentell für verschiedene Objektformen berechnet. Bei Segeln sind diese Koeffizienten nicht konstant und hängen vom Windwinkel und der Segelform ab.

Die Segelform spielt eine sehr wichtige Rolle bei der Manövrierfähigkeit von Schiffen. Seit der Antike hatten Schiffe Rahsegel. Es war sehr schwierig, gegen den Wind zu segeln. Während der Renaissance im 15. und 16. Jahrhundert wurden Lateinersegel erfunden, die das Segeln gegen den Wind ermöglichen. Dreieckslateinsegel haben eine viel bessere Auftriebwirkung. Im 17. Jahrhundert wurde das Bermuda-Rigg erfunden, die Basis für alle modernen Segelyachten.



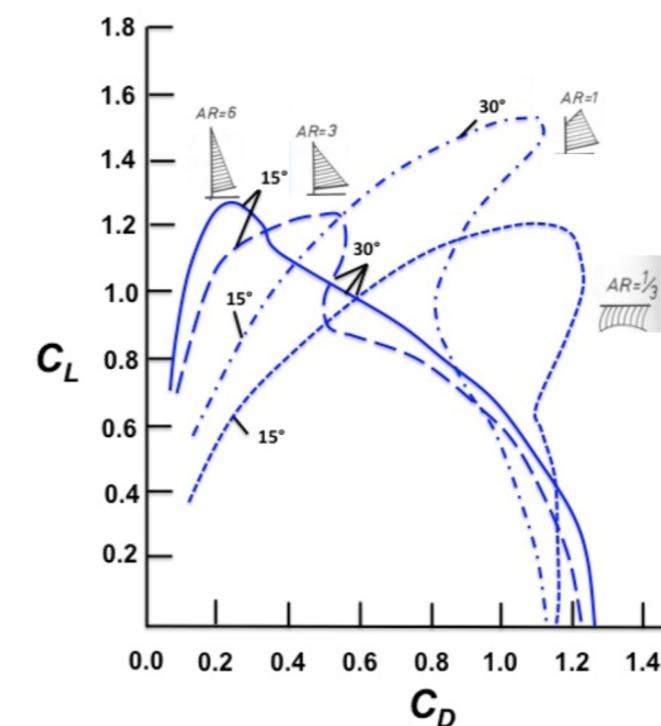
Square sail



Lateen sails



Bermuda sails



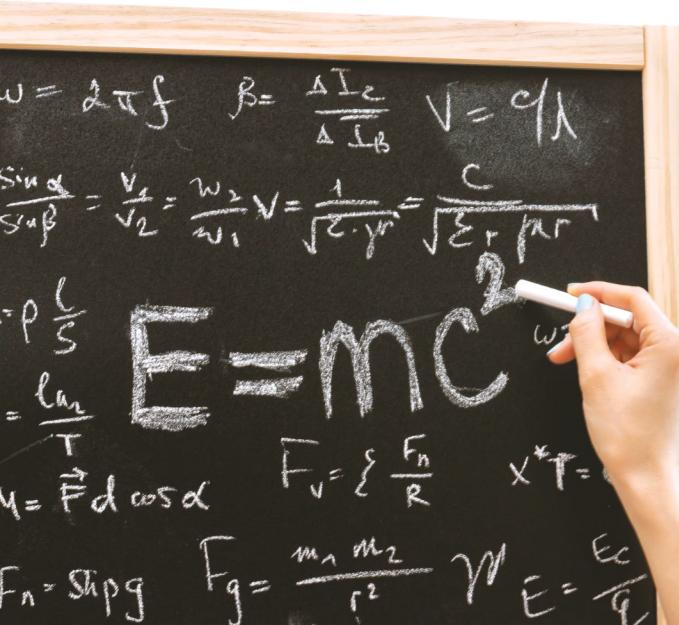
Dieses Diagramm zeigt die Beziehungen zwischen den Auftriebs- und Widerstandskoeffizienten für verschiedene Segelseitenverhältnisse und scheinbare Windwinkel. Wie du siehst, verschiebt sich der maximale Auftrieb bei niedrigen Seitenverhältnissen weiter in Richtung erhöhten Widerstand (im Diagramm nach rechts). Außerdem erzeugt ein höheres Seitenverhältnis bei kleineren Anstellwinkeln mehr Auftrieb und weniger Luftwiderstand als bei niedrigeren Seitenverhältnissen.

Alle modernen Yachtsegel haben hohe Streckungsverhältnisse $AR \sim 6$, was die beste Auftriebskraft und sehr effektives Segeln gegen den Wind erzeugt. Für das Vorwindsegeln werden jedoch die speziellen Segel mit niedrigem Seitenverhältnis ($AR \sim 1$ bis 2) verwendet (z. B. Spinnaker). Weil sie einen viel besseren Widerstandskoeffizienten haben, der eine dominierende Kraft beim Vorwindsegeln ist.

Kiel

Der Kiel ist der entscheidende, Mittschiffs im Boden angebrachten Längsverband eines Schiffes oder Bootes. Der Kiel ist somit das „Rückgrat“ des Schiffes. An ihm sind die querstabilisierenden Spanen, die „Rippen“, angebracht. An seinen Enden geht der Kiel in die Steven über. Neben der Stabilisierung des Rumpfes dient er auch der Erhöhung der Kursstabilität und vor allem bei Segelfahrzeugen – der Verringerung des seitlichen abdriftens.

Im Unterschied zu einem aufholbaren Schwert ist ein Kiel in der Regel fest montiert und hat ein unvermeidliches Eigengewicht. Je nach Art des Schiffes gibt es allerdings sehr unterschiedliche Kielformen, die sich teilweise nicht ganz klar vom Schwert trennen lassen.



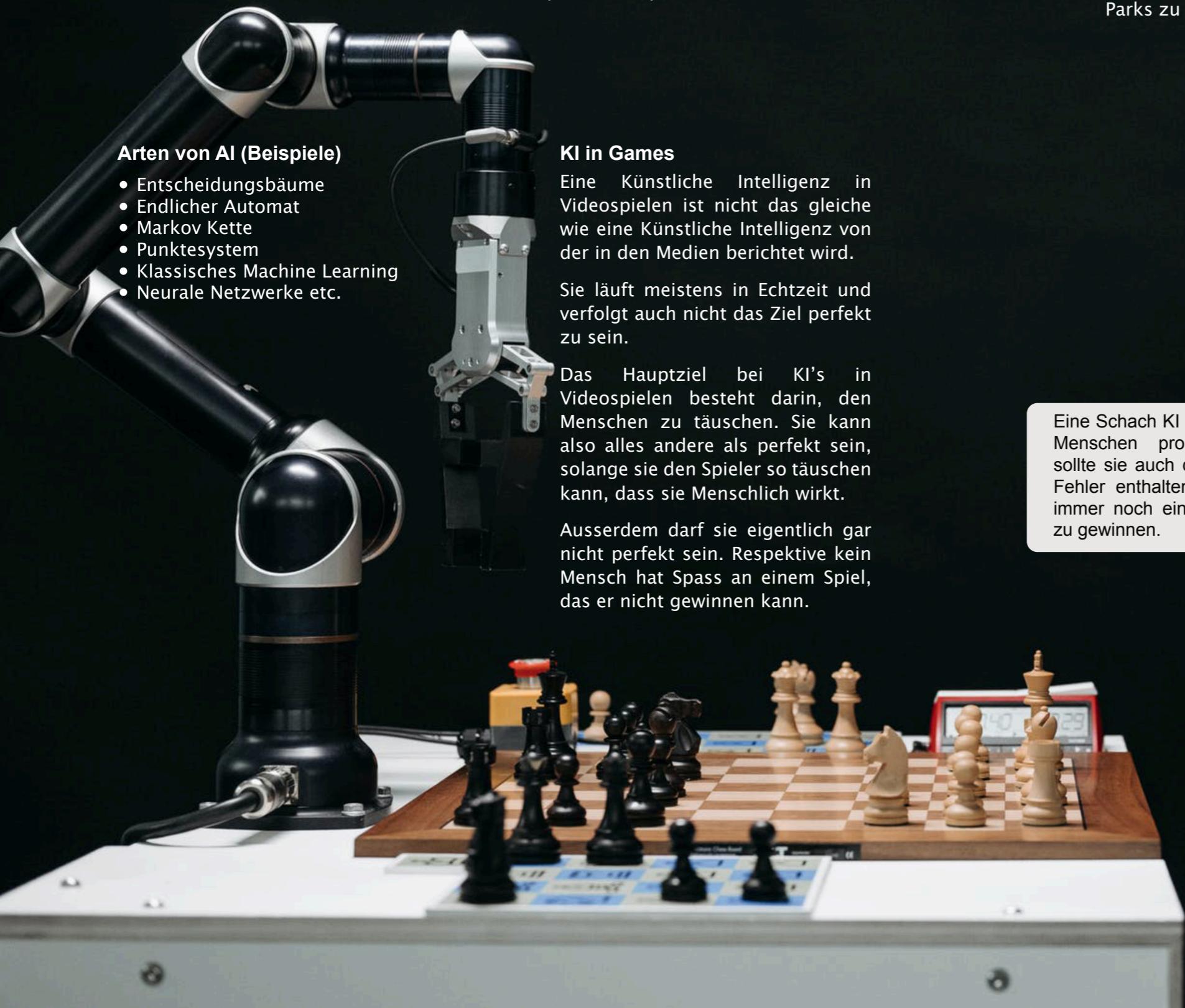
Der dominierende Faktor, der die Segeleffizienz beeinflusst, ist das Segelquerschnittsverhältnis. Da die Auftriebskraft effektiver als der Luftwiderstand zum Vorantreiben des Schiffes beiträgt, versuchen Segelmacher, den Auftrieb zu erhöhen. Die Streckung ist ein Verhältnis zwischen Segelbreite und -höhe. Eine hohe Streckung weist auf ein langes, schmales Segel hin, während eine niedrige Streckung auf ein kurzes, breites Segel hinweist.

Künstliche Intelligenz

Einführung

Wir haben zuerst einmal angeschaut was eine künstliche Intelligenz überhaupt ist, dabei war ich selbst etwas überrascht, wie schnell etwas als künstliche Intelligenz betrachtet werden kann & wie vielseitig sie angewendet werden kann.

Es gibt ausserdem verschiedene Arten von AI's, welche verschiedene Anwendungsbereiche haben. Wir haben einige von Ihnen unter die Lupe genommen und versucht herauszufinden in verschiedenen Spielen eine passende AI zu finden.



Arten von AI (Beispiele)

- Entscheidungsbäume
- Endlicher Automat
- Markov Kette
- Punktesystem
- Klassisches Machine Learning
- Neurale Netzwerke etc.

KI in Games

Eine Künstliche Intelligenz in Videospielen ist nicht das gleiche wie eine Künstliche Intelligenz von der in den Medien berichtet wird.

Sie läuft meistens in Echtzeit und verfolgt auch nicht das Ziel perfekt zu sein.

Das Hauptziel bei KI's in Videospielen besteht darin, den Menschen zu täuschen. Sie kann also alles andere als perfekt sein, solange sie den Spieler so täuschen kann, dass sie Menschlich wirkt.

Ausserdem darf sie eigentlich gar nicht perfekt sein. Respektive kein Mensch hat Spass an einem Spiel, das er nicht gewinnen kann.

Projektidee KI Verhalten

Da ich im 2 Modul schon auf eigene Faust sehr viel mit AI zu tun hatte, fällt mir die Herangehensweise jetzt etwas leichter. Ich möchte jedoch das dazu gelernte Wissen dieses Moduls verwenden wie z.B Statistik oder Nav Meshs und mit dem Fokus auf viele Parameter eine möglichst transparente und benutzerfreundliche AI programmieren. Für die Modulabgabe möchte ich gerne eine Simulation von Kunden, eines Unternehmens machen. Die Simulation sollte zeigen wie sich Kunden in einem Stadtviertel bewegen & anhand der Statistik (Wirtschaft des Unternehmens), den Laden besuchen oder nicht. Ich denke die beste Angehensweise für diese Aufgabe ist eine State Machine für den Kunden zu schreiben und die NPC's mithilfe eines Navigation Meshs und Wegpunkten durch die verschiedenen Bereiche der Stadt & Parks zu führen.



Eine Schach KI wurde von einem Menschen programmiert, also sollte sie auch die Menschlichen Fehler enthalten. Ich habe also immer noch eine kleine Chance zu gewinnen.

Fun Fact: Warum Computer an Schach scheitern...

Bei einer Schachpartie gibt es ungefähr 10^{120} verschiedene Kombinationen bei einem Spiel die gespielt werden können. Eine perfekte Schach KI müsste jeden dieser Züge berechnen um jeden Zug vorhersehen zu können. Ein Moderner Supercomputer bräuchte 10^{90} Jahre um alles auszurechnen. Das ist länger als vom Urknall bis jetzt. In unserem beobachtbaren Universum gibt es außerdem nur ungefähr 10^{80} Atome, also nicht ausreichend genug um alle möglichen Spielmöglichkeiten zwischenzuspeichern. Seen on 100 Sekunden Physik: <https://www.youtube.com/watch?v=WxFL8DUTslw>

Navigation Mesh

Mit einem Navigation Mesh kannst du mit Hilfe von Meshs, verschiedene GameObjects z.B. begehbar oder unbegehbar machen. Du kannst verschiedene Einstellungen vornehmen und mit Hilfe des Nav Mesh Agenten Components, deine Enemy's oder verbündete NPC's auf diesem Nav Mesh per Code herum navigieren.



A* Algorithmus / Pfadfindung

Du kannst auf deinem Nav Mesh verschiedene Wegpunkte platzieren. Dein Nav Mesh Agent orientiert sich dann an diesen Wegpunkten, um zum Beispiel einem bebauten Weg/Strasse zu folgen, anstatt durch eine Wiese zu gehen.

Damit der Agent den schnellst möglichen Weg findet, kannst den Wegpunkten sogenannte „Gewichte“ geben. Das Gewicht, das ein Wegpunkt in der Nähe hat, ist einerseits abhängig von der Distanz, aber auch von der Steigung des Terrains oder die Art des Bodens, Wasser oder Land etc.

Navigation Mesh Research

Bei der Recherche nach komplexeren Navmesh Konzepten bin ich auf diese Tutorial Reihe über Navmeshs, Navmesh Agenten usw. gestossen. Seine Videos haben mir sehr geholfen mein Navmesh Projekt erfolgreich umzusetzen. Sehr Empfehlenswert!

https://www.youtube.com/playlist?list=PLIINmP7eq6TSkwDN8OO0E8S6CWybSE_xC



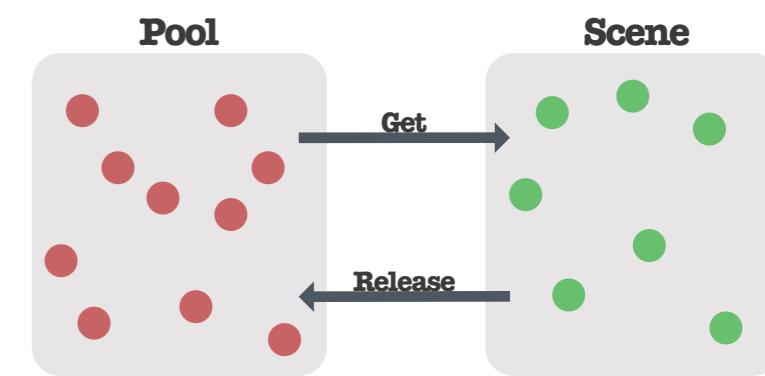
Attack on Titan KI

Beim durcharbeiten der Navmesh Tutorials bin ich auf eine Idee für ein kleines Attack on Titan Minigame gekommen.

Die Titanen tauchen in der Stadt auf und verfolgen dich mit Hilfe des Nav Meshs. Es gibt Titanen unterschiedlicher Meter Klassen. Kleine Titanen können dich an mehreren Orten erreichen. Sind jedoch etwas langsamer. Grössere Titanen kommen nicht überall durch sind jedoch schneller und gefährlicher.

Spawning - Object Pooling

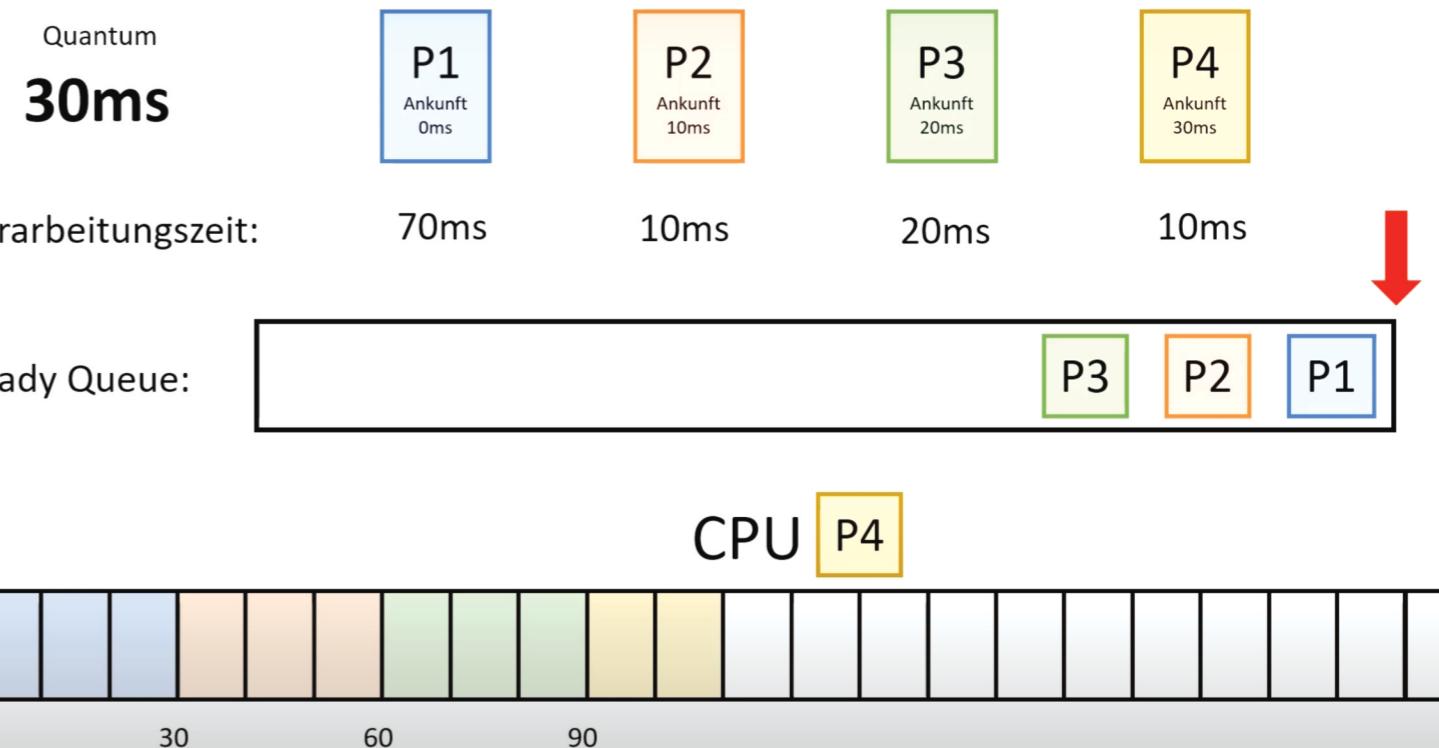
Zum Spawning der Titanen eignet sich das benutzen von Object Pooling perfekt. Object Pooling ist eine richtig coole Optimierungs Technik um „Garbage von GameObjects“ zu vermeiden. Wir haben das bereits in der Schule angeschaut aber ich konnte mein Wissen in dem Modul noch vertiefen.



Round Robin Schedule Algorithmus

Zum spawning der Titanen verwende ich den Round Robin Algorithmus. Er ist einer der ältesten und einfachsten Algorithmen der in interaktiven Systemen verwendet wird.

<https://www.youtube.com/watch?v=1IQQGZ9ZxfM>



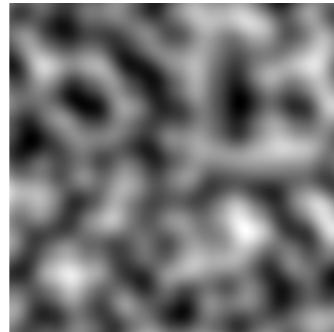
Terrain Generation

Wir haben im Unterricht mit Marcel auch angeschaut wie ein Perlin Noise eigentlich aufgebaut ist. Ich habe hier mal noch ein paar Grafiken herausgesucht, welche Visuell besser erklären wie ein Perlin Noise entsteht.

Perlin Noise

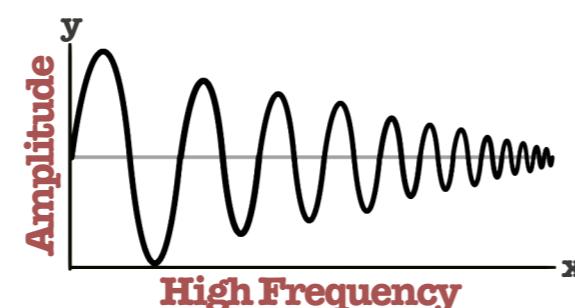
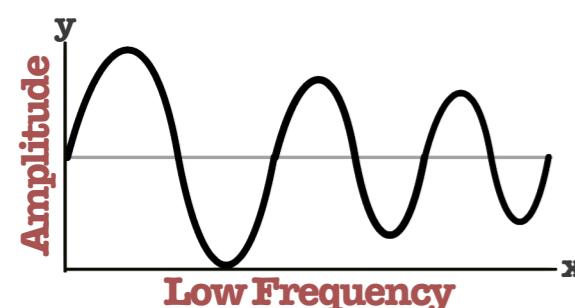
Bei der ersten Feedback Session hat mich Michael darauf aufmerksam gemacht evt. Perlin Noise für den Wind zu benutzen um das ganze etwas dynamischer zu machen. Auch um ein procedural generiertes Terrain zu erstellen brauche ich ein Perlin Noise. Es gibt bei Unity selbst schon eine Funktion dafür.

`Mathf.PerlinNoise();`



Aber was ist Perlin Noise überhaupt? Perlin Noise ist ein Pseudo random Muster von float Werten, die über eine 2D-Ebene generiert werden. Das „noise“ enthält nicht an jedem Punkt einen vollständigen Zufallswert, sondern besteht aus Wellen, deren Werte über das Muster hinweg allmählich zunehmen oder abnehmen.

Bei der Amplitude handelt es sich um die Y-Achse und die Frequenz ist die X-Achse.

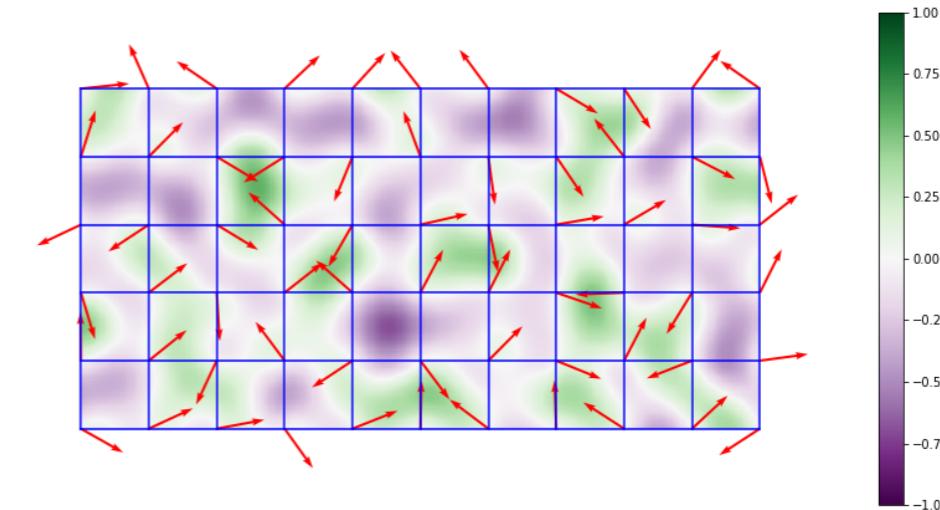


Octaves

Um eine Landschaft oder so zu formen können wir mehrere „Noise“ zusammen tun. Es entsteht dadurch eine neue NoiseMap. Du kannst die verschiedenen Noise Maps als einzelne Teile eines ganzen Betrachten. Zum Beispiel der Grundriss eines Berges, der zweite als Felsblöcke und der dritte als kleine Steine.

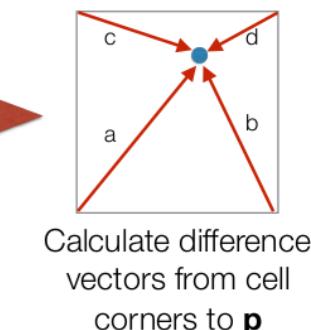
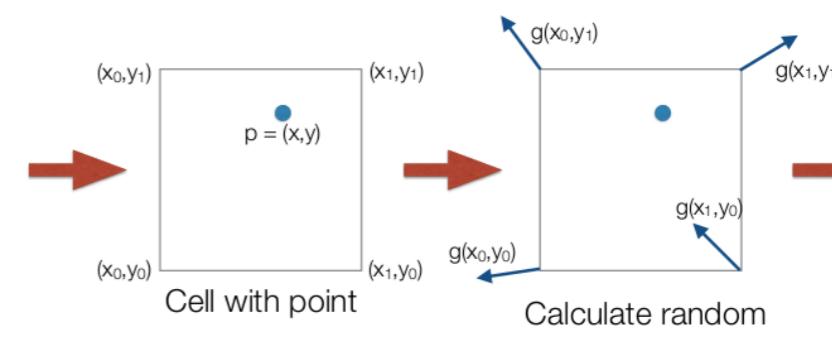
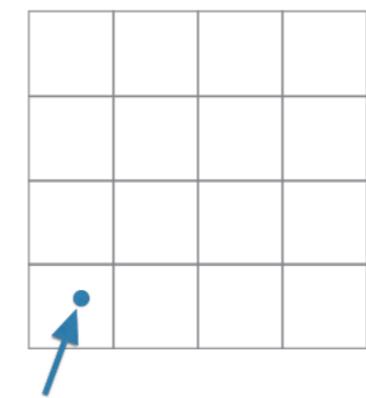
Oktave 1 Grundriss Frequenz = Lacunarity ⁰	Oktave 2 Felsblöcke Frequenz = Lacunarity ¹	Oktave 3 Kleine Steine Frequenz = Lacunarity ²
---	--	---

Die Lacunarity kontrolliert das erhöhen der Frequenz von den verschiedenen Oktaven.
Persistance kontrolliert die Verringerung der Amplitude von den Oktaven.



2D Perlin Noise

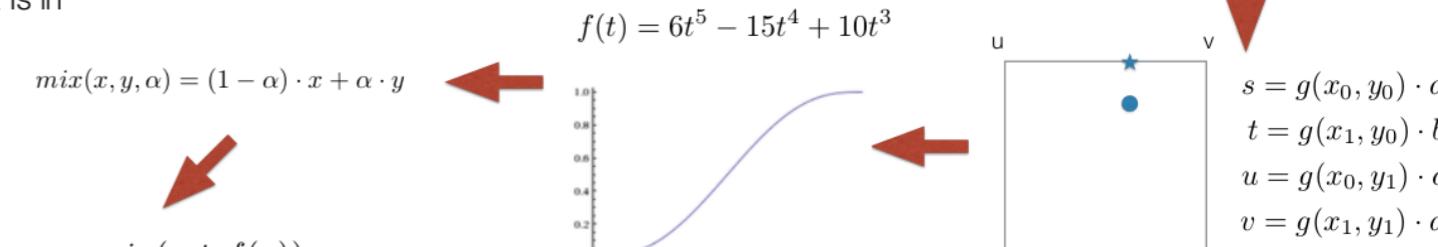
Subdivide domain into grid with unit cells



Find cell that our point is in

$$mix(x, y, \alpha) = (1 - \alpha) \cdot x + \alpha \cdot y$$

$$\begin{aligned} st &= mix(s, t, f(x)) \\ uv &= mix(u, v, f(x)) \\ noise &= mix(st, uv, f(y)) \end{aligned}$$



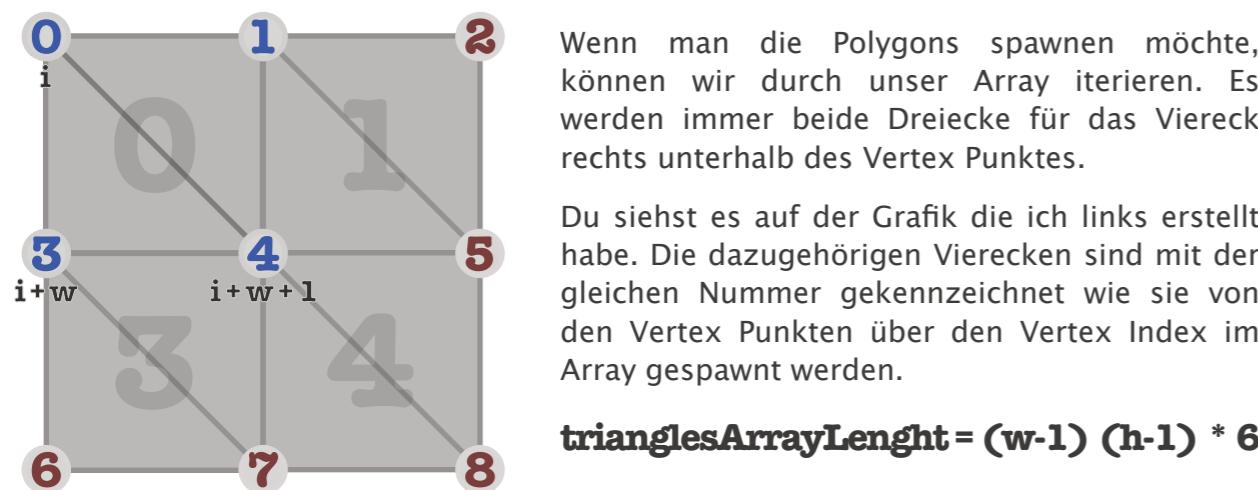
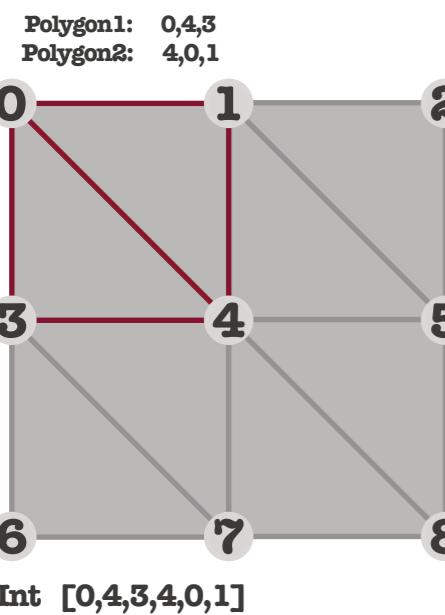
$$\begin{aligned} s &= g(x_0, y_0) \cdot a \\ t &= g(x_1, y_0) \cdot b \\ u &= g(x_0, y_1) \cdot c \\ v &= g(x_1, y_1) \cdot d \end{aligned}$$

dot products to get scalar values for the corners

Mesh Generation

Neben den standard 3D Objects von Unity oder den importierten Assets haben wir in Unity auch selbst die Möglichkeit Meshs von Hand oder besser gesagt mit Code zu generieren. Dies öffnet sehr viele neue Türen und bietet dir mehr Kontrolle über deine Spielwelt.

Ein Mesh besteht aus Vertex Punkten, Kanten und Flächen. Die Vertex Punkte sind mit den Kanten verbunden und bilden so Dreiecke, sogenannte Polygons. Um ein Mesh zu erstellen müssen wir die Vertex Punkte in einem Array speichern. Das sieht dann ungefähr so aus wie in der rot markierten Fläche.



Ein solch erstelltes Mesh kann für verschiedene Zwecke genutzt werden. Zum Beispiel kann man eine Landschaft generieren, oder ein Meer mit Wellen simulieren etc. Alles was man jetzt noch benötigt sind die passenden mathematischen Formeln.

Quellen

Titelbild

<https://cdn.wallpapersafari.com/74/16/CwyDcO.jpg>

Sail

<https://www.pexels.com/photo/boat-mast-2326961/>

Water Splash Sketch

<http://getdrawings.com/image/water-splash-drawing-56.png>

Bernoulli Gleichung

<https://vlytsus.medium.com/my-sailing-story-unity-game-development-part-1-6b6c5e7f3844>

E=MC₂

<https://www.pexels.com/photo/person-holding-a-chalk-in-front-of-the-chalk-board-714699/>

Chess AI

<https://www.pexels.com/photo/elderly-man-thinking-while-looking-at-a-chessboard-8438918/>

Compass

<https://www.pexels.com/photo/shallow-focus-photography-of-black-and-silver-compasses-on-top-of-map-1203808/>

Round Robin Scheduling

<https://www.youtube.com/watch?v=1IQQGZ9ZxfM>

Titan

https://easydraweverything.com/wp-content/uploads/2019/08/colossal_titan_step_11.png

Perlin Noise

<https://rmarcus.info/blog/assets/perlin/raw/octaves.png>

Perlin Noise Interpolated

https://en.wikipedia.org/wiki/Perlin_noise#/media/File:PerlinNoiseInterpolated.png

2D Perlin Noise

<https://i.stack.imgur.com/5NANE.png>

Dieses Dokument ist noch in Bearbeitung, weitere Quellen folgen.