Julian Hopp

Ist es möglich eine realistische Simulation eines, durch Anwender-Interaktion beeinflussbares, Schwarmverhaltens in der Unity-Engine performant zu gestalten?

GPD415

SAE Hamburg

Abgabe: 14.03.2016

Betreuender Fachlehrer: Fabio Anthony

(Wortanzahl)

1.1 Einleitung

"Schwarm: größere Anzahl sich [ungeordnet,] durcheinander wimmelnd zusammen fortbewegender gleichartiger Tiere, Menschen" * (http://www.duden.de/rechtschreibung/Schwarm)

"Allgemein bezeichnet man einen Schwarm als einen Zusammenschluss von Tieren in großer Zahl, ohne soziale Bindung oder Hierarchie." * (http://felixmetzfacharbeit.de/schwarmverhalten.html)

" Der wohl wichtigste und bemerkenswerteste Punkt des Schwarmes ist die fehlende Hierarchie. Ein Schwarm besitzt keinen Anführer. Jedes Mitglied des Schwarmes ist absolut gleichrangig." * (http://felixmetzfacharbeit.de/schwarmverhalten.html)

1.2 Themen Definition

In dieser Arbeit werden die drei Grundlegenden "Steering Behaviours" begriffe von Craig Reynold beschrieben, "Separation", "Alignment" und "Cohesion", um eine natürliche Animation von Tierschwärmen in Unity zu simulieren. Anschließend wird in der Unity-Engine getestet ob dies Performant abläuft unter Anwender-Interaktion, desweiteren als Impact benannt.

1.3 These

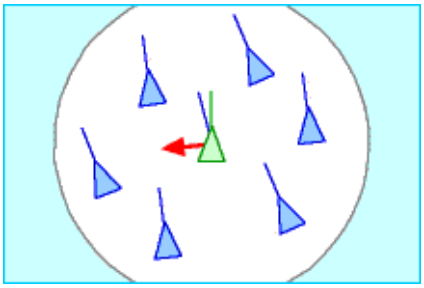
These: Ist es möglich eine realistische Simulation eines, durch Anwender-Interaktion beeinflussbares, Schwarmverhaltens in der Unity-Engine performant zu gestalten?

-(veraltet)In dieser Facharbeit geht es darum, die Anwender-Interaktion dazulegen, welche bei einem Bombardements im 2D Bereiches auftritt und die Reaktion des Schwarmes auf dieses. Der Schwerpunkt liegt darauf, dies Performant zu gestalten.

2. 1 Grundlagen des Flockings

Schwarmverhalten beschreiben die Bewegung eines einzelnen Individuums, auch Boid oder autonomer Agent genannt, innerhalb eines Schwarms mehrerer Boids, sowie das daraus resultierende Verhalten aller Boids als Schwarm. (Buckland, 2005)

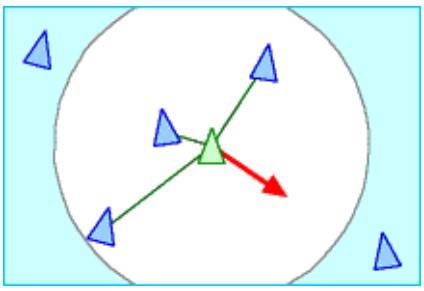
2.2 Alignment



(Abbildung 1, Alignment) *

Das Alignment ist dafür zuständig, dass die Boids sich in eine vorgegebene Richtung und Geschwindigkeit Orientieren und somit die Positionsänderung vorgenommen wird.

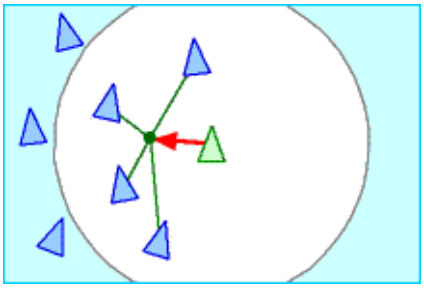
2.3 Separation



(Abbildung 2, Separation) *

Die Separation sorgt dafür, dass die Boids untereinander einen vorgegebenen Abstand einhalten. Dadurch kann die Separation genutzt werden eine Kollisionserkennung nicht notwendig zu machen, um so Performance zu sparen.

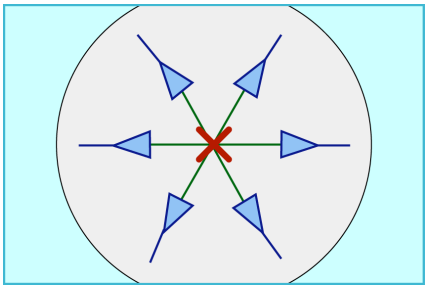
2.4 Cohesion



(Abbildung 3, Cohesion) *

Die Cohesion wird verwendet, um zu verhindern dass die Boids sich zu weit voneinander entfernen, sodass der Schwarm keine Boids verliert. Die Verbindung von Cohesion und Separation wirkt sich somit auf den Abstand zwischen den Boids aus und die Geschwindigkeit der Distanz Veränderung zwischen den Boids im Schwarm.

2.5 Impact



(Abbildung 4, *)

Die Impact Simulation ist dafür verantwortlich, um Interaktion mit dem Schwarm realistisch zu gestalten, wie beispielsweise bei einem Bombardement.

Für die Berechnung des Impacts hilft beispielsweise das Unity interne NavMesh, die Unity "Transform.LookAt" Funktion und "Transform.forward" Funktion, genaueres dazu in Kapitel "4.1 Praxis Test ".
3 Methodik

Bevor der Schwarm implementiert werden kann und die Performance getestet werden kann, muss geguckt werden wie ein Schwarm auf einen Impact Reagieren würde. In der Tierwelt ist das gut zu beobachten, beispielsweise einem Fischschwarm, wenn in diesem Fischschwarm ein Fremdkörper (wie ein Hai oder ein Stein auf der Wasseroberfläche) auftaucht, zersprengt sich der Scharm sofort, und erst später fügt dieser sich wieder zusammen.

3.1 Engine Auswahl

Er wird die Unity Engine (Version 5.3.2) benutz, da Unity auf unzählige Art- und weisen dem Benutzer das Implementieren erleichtert. Besonders erwähnenswert ist der Profiler, mit dem sich die Performance permanent über ein extra Fenster kontrollieren lässt *(http://docs.unity3d.com/Manual/ProfilerPhysics.html).
Desweiteren bietet Unity eine zuverlässige und Offizielle Dokumentation* (http://docs.unity3d.com)
Zudem ist Unity 3D für unkommerzielle Projekte Kostenlos.

3.2 Programmiersprache

Um die Performance des Schwarms unter Anwender-Interaktion vergleichen und testen zu können, muss diese erst Implementier werden. Da die Engine Auswahl auf Unity fiel, gibt es nun die Programmiersprachen Java, C#, UnityScript und Boo zum implementieren des Schwarmverhalten. Aus diversen gründen wurde hier die Programmiersprache C# gewählt, beispielsweise wird C# fortlaufend Weiterentwickelt.
3.3 Entwicklungsumgebung

Um die C# Bibliotheken und Anwendungen zu erstellen wird in der Theorie lediglich ein Compiler und ein Texteditor benötigt. Da eine Professionale Entwicklungsumgebung jedoch das Entwicklung tempo beschleunigt und den Komfort erhöht, wurde aus verschiedenen Gründen die Microsoft Visual Studio Community 2015 Edition gewählt. Diese Entwicklungsumgebung wird ab Unity 5.2 kostenfrei mitgeliefert * (http://blogs.unity3d.com/2015/09/08/unity-5-2-easy-access-to-unity-services/).
Desweiteren wird die Microsoft Visual Studio Community 2015 Version vorlaufend aktualisiert.

Zusätzlich wird das Plug-In ReSharper benutzt, um die Naming Convention zu beschleunigen und zu vereinfachen.

3.4 Versionsverwaltung

Als professionelles Versionsmanagement mit Cloud Speicherung wurde Github gewählt, um das Schwarmverhalten auf verschiedenen Rechnern nicht nur synchronisieren, sondern auch testen zu können. Somit ist sichergestellt das jederzeit eine funktionsfähige Version besteht und diese falls benötigt nur noch Aktualisiert werden muss. Als grafische Benutzeroberfläche zur Bedienung des Versionskontrollsystems wurde SourceTree benutzt.
4 Durchführung

4.1 Praxis Test

Um die Performance des Algorithmus testen zu können, muss dieser Algorithmus vorerst in Unity Implementiert werden.

Hierfür bietet Unity viel Unterstützung, beispielsweise das NavMesh und den NavMesh Agent. Das NavMesh bietet unter anderem die Möglichkeit, eine bereits vorhandene Oberfläche für den NavMesh Agent begehbar zu gestalten. Dies ist grade im 2D Bereich sehr von Vorteil. Die NavMesh Agent Klasse besitzt die Funktion "Destination", die es dem Agent erlaubt zum Zielobjekt zu laufen.

Auch nimmt Unity eine Vielzahl an Mathematischen Berechnungen ab, besonders wichtig hierbei sind die Vektor Klassen, vor allem im 3D Bereich.

Mit der Funktion "velocity" wird die Bewegungsgeschwindigkeit ermittelt. Diese Geschwindigkeit kann zur Überprüfung der erlaubten maximal Geschwindigkeit des Boids genutzt werden. Die "velocity" wird gegen die vorher Initialisierte Maximal Geschwindigkeit geprüft und falls überschritten, zurück auf die diese gedrosselt. Zum Drosseln muss die Funktion "Velocity" Mathematisch normalisiert werden, mit der in der Funktion vorhandenen Funktion "normalized", und anschließend wieder unter Berücksichtigung der "deltaTime" (dazu mehr im übernächsten nächsten Absatz) mit der Maximal Geschwindigkeit multipliziert werden.

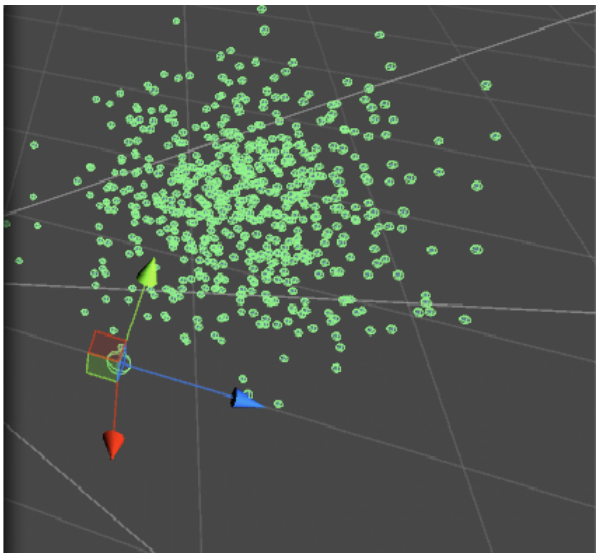
Noch zu überprüfen im Bezug auf die Geschwindigkeit ist die ebenfalls vorher Initialisierte Mindestgeschwindigkeit. Hierbei wird ganz ähnlich wie im Absatz zuvor zuerst überprüft, ob die Mindestgeschwindigkeit unterschritten wurde, falls dies eintrifft wird diese auf die Mindestgeschwindigkeit angehoben. Dazu wird wieder die Magnitude durch das mathematische Normalisieren ermittelt und mit der Mindestgeschwindigkeit multipliziert.

Das Unity Keyword "Time" ist ein Interface, dass genutzt wird um Information bezüglich der Zeit zu erhalten. Die Funktion "deltaTime" des Interfaces Time sollte in Unity meistens mit allem multipliziert werden, dass jeden Frame etwas Addiert oder Subtrahiert. "Use this function to make your game frame rate independent." * (http://docs.unity3d.com/ScriptReference/Time-deltaTime.html)

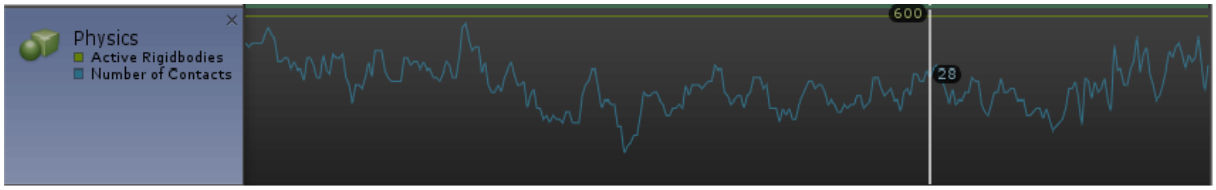
Gerne genutzt im Bereich der Zeit, ist die Coroutine. Die Coroutine Funktion ermöglicht es, die Ausführung im Skript solange zu verzögern bis die Bedingung der Coroutine erfüllt ist. Diese Bedingung muss als "Yield" der "WaitForSeconds" Funktion deklariert werden.

Falls die Interaktion des Schwarmes es Benötigt, ist es möglich über die LookAt Funktion die Boids in Richtung des Impacts gucken zu lassen, um anschließend mit dem negativen wertes der Transform.forward Funktion die Bewegung einer Flucht zu simulieren. Ebenfalls möglich ist es, mit der LookAt Funktion die Kamera Dynamisch auf den Flock Mittelpunkt zu richten.

Ein Optischer Test zeigt, ob die Implementierung des Schwarm Algorithmus erfolgreich zum gewünschten Ziel geführt hat.


(Abbildung 5, selbst erstellt, Unity Scene *)

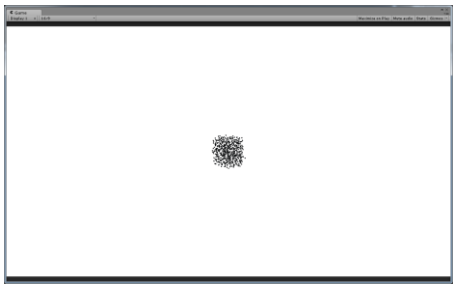
Auf Abbildung 5 zu sehen ist der Versuch eines möglichst realen Schwarmes mit einer Schwarmgröße von 600 Boids im 3D Bereich. Jeder Boid besitzt einen Collider Component, einen Rigidbody Component, einen Mesh Filter Component, einen Mesh Renderer Component, eine Shader Componente und eine Transform Componente.



(Abbildung 6, selbst erstellt, Profiler *)

Abbildung 6 zeigt die 600 Aktiven Rigidbodys auf Abbildung 5, und deren Kontakten per Frame. Wie auf dem Diagramm zu sehen, gibt es durchschnittlich 10 - 60 Kontakte unter allen Boids pro Frame. Wenn der Schwarm sich konstant in eine gleichbleibende Richtung bewegt, kommt es zu den zu sehenden durchschnittlichen 10 kontakten, bei raschen Bewegung Änderungen erhöht sich der Kontakt per Frame auf 60.

Die Anwender Interaction richtet sich an Abbildung 4, aus dem Kapitel 2.5 Impact. Der Schwarm Simuliert auf Kommando ein Szenario wie beispielsweiße ein Fisch Schwarm der von einem Hei angegriffen wird. Dieses Kommando kann beispielsweise ein Maus-Klick in den Schwarm sein, oder ein Hotkey.


(Abbildung 7, selbst erstellt, Unity Game Scene *)

Auf Abbildung 7 bleibt der Schwarm noch zusammen, da noch keine Interaction passierte, dies ist die ausgangs Position, mit 10-60 Kontakten pro Frame.


(Abbildung 8, selbst erstellt, Unity Game Scene *)

Die Anwender Interaction passiert, der Schwarm entfernt sich vom eintrittsort des Impacts. Ab hier gibt es keine Boid Kontakte pro Frame bis zu Abbildung 11, da sich die Boids auch untereinander voneinander entfernen.


(Abbildung 9, selbst erstellt, Unity Game Scene *)

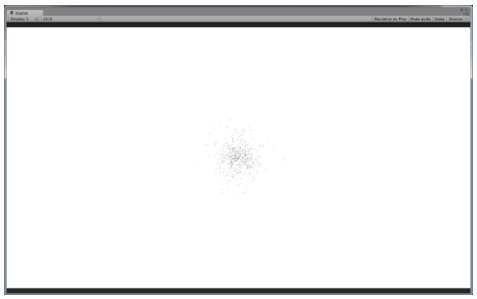
Die Individuellen Boids haben ihren Maximalen Fluchtpunkt erreicht. Dies kann auf verschiedene weisen passieren, beispielsweise wenn die Boids eine Abstands Berechnung zum Impact haben.


(Abbildung 10, selbst erstellt, Unity Game Scene *)

Die Boids bewegen sich individuell erneut Richtung Schwarm-Mittelpunkt.


(Abbildung 11, selbst erstellt, Unity Game Scene *)

Der Mittelpunkt des Schwarmes wurde von den Boids fast erreicht, dadurch kommt es zu einer Bewegung Drosslung, somit sind Kontakte zwischen den Boids wieder möglich.


(Abbildung 12, selbst erstellt, Unity Game Scene *)

Die ausgangs Position ist fast erreicht, und es pendeln sich wieder die 10-60 Kontakten pro Frame ein. Die Anwender Interaktion ist somit abgeschlossen.

- weitere Mathe Klassen erwähnen
5 Ergebnisse

5.1 Auswertung

Nach Implementierung des Schwarm Algorithmus, mit einer Schwarmgröße von 600 Boids, zeigt der Performance Profiler keinerlei veränderung der Performance.

5.2 Tabellen & Diagramme

6 Zusammenfassung

6.1 Fazit

Inhaltsverzeichnis

1.1 Einleitung 2
1.2 Themen Definition 2
1.3 These 2
2. 1 Grundlagen des Flockings 2
2.2 Alignment 2
2.3 Separation 3
2.4 Cohesion 3
2.5 Impact 3
3.1 Engine Auswahl 4
3.2 Programmiersprache 4
3.4 Versionsverwaltung 4
4.1 Praxis Test 5

5.1 Auswertung 8
5.2 Tabellen & Diagramme 8
6.1 Fazit 8

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Abbildung 1, Abbildung 2, Abbildung 3: http://www.red3d.com/cwr/boids/ by Craig Reynolds

Literaturverzeichnis:

Abbildungsverzeichnis:

Abbildung 1, Abbildung 2, Abbildung 3: http://www.red3d.com/cwr/boids/ by Craig Reynolds
Abbildung 4 ist selbstgemacht