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

Date Submitted:	14.03.2016
Award Name:	SAE Diploma
Course:	GPD 415
Name:	*Julian Hopp*
City:	Hamburg
Country:	Germany
Staffing:	*Fabio Anthony*
Word Count:	4532

Inhaltsverzeichnis

1 Einleitung....3
1.1 Themen Definition und These....4

2 Grundlagen....5
2.1 Grundlagen des Flockings....5

3 Methodik....8
3.1 Engine Auswahl....8
3.2 Programmiersprache....8
3.3 Variablen Benennung....9
3.4 Entwicklungsumgebung....10
3.5 Versionsverwaltung....11

4 Durchführung....11
4.1 Praxis Test....15

5 Ergebnisse....19
5.1 Auswertung....19

6 Zusammenfassung....20
6.1 Fazit....20

7 Literaturverzeichnis....21

8 Abbildungsverzeichnis....24

1 Einleitung

Computerspiele und animierte Kinofilme werden immer komplexer und realistischer. Daher gewinnt eine Echtzeit Simulation immer mehr an Bedeutung. Die Simulation eines Schwarmverhaltens, gerade in Kinofilmen, ist manchmal unverzichtbar. Viele große Schlachten, wie beispielsweise die aus „Der Herr der Ringe", werden nicht von tausenden Statisten ausgetragen, sondern von modellierten Charakteren.

> „Mitte der 1980er Jahre entwickelte Craig Reynolds ein solches interaktives [Schwarmverhalten, d. Verf.] Partikelsystem, nennt es aber - und in noch viel stärkerer Anlehnung an biologische Systeme - ganz anders. Reynolds ist ebenfalls Grafikdesigner und seinerzeit tätig für die Grafikabteilung der Firma Symbolics. Sein Schwarm-Animationsmodell, das er unter dem Titel Flocks, Herd, and Schools: A distributed behavioral model veröffentlicht, klingt nicht nur wie ein Text aus dem Kontext der Verhaltensbiologie. Es wird auch in fast jedem Papier der späteren, computergestützten biologischen Schwarmforschung als eine Art >Urtext< zitiert."[1]

Auch wenn das von Craig Reynolds erstellte Schwarmverhalten nicht für die Spieleindustrie entwickelt wurde, ist der Schwarm-Algorithmus für genau diese in vielen Situationen sehr nützlich.

> „Allgemein bezeichnet man einen Schwarm als einen Zusammenschluss von Tieren in großer Zahl, ohne soziale Bindung oder Hierarchie. Der wohl wichtigste und bemerkenswerteste Punkt des Schwarmes ist die fehlende Hierarchie. Ein Schwarm besitzt keinen Anführer. Jedes Mitglied des Schwarmes ist absolut gleichrangig."[2]

Die Zielplattform für die Implementierung des Schwarmverhaltens ist der Personal Computer. Für leistungsschwächere Zielplattformen, wie etwa mobile Endgeräte, müssen gegebenenfalls Anpassungen vorgenommen werden.

[1]Vehlken, 2012:315
[2]Metz, 2013:o.S.

1.1 Themen Definition und These

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

„Schwarm: größere Anzahl sich [ungeordnet,] durcheinander wimmelnd zusammen fortbewegender gleichartiger Tiere, Menschen“[3]

Es werden die drei grundlegenden „Steering Behaviours“ 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 unter Anwender-Interaktion performant abläuft. Dieser wird im Weiteren als Impact bezeichnet.

In dieser Facharbeit geht es darum, eine Anwender-Interaktion zu zeigen, welche zum Beispiel bei einem Hai Angriff auf einen Fischschwarm auftritt und die Reaktion des Schwarmes auf diesen. Der Schwerpunkt liegt darauf, diese Aktion performant zu gestalten.

Craig Reynold benannte das virtuelle Flugobjekt, welches er mit „Boid“ betitelte, aus dem Englischen von „Bird object“.

[3]Dudenredaktion, 2006:913

2 Grundlagen

Grundlegend baut die Flocking-Simulation auf dem Schwarm-Algorithmus auf. Dieser Algorithmus wird in Unity implementiert. Zum Implementieren wird eine Programmiersprache benutzt, sowie Unity interne Hilfsmittel. Diese Programmiersprache wird von einer Entwicklungsumgebung unterstützt, die es dem Benutzter erleichtert zu programmieren.

> "Scripting is an essential ingredient in all games. Even the simplest game will need scripts to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game."[4]

2.1 Grundlagen des Flockings

In der Informatik beschreibt das Schwarmverhalten die Bewegung eines einzelnen Individuums, auch Boid oder autonomer Agent genannt, innerhalb eines Schwarms, sowie das daraus resultierende Verhalten aller Boids als Schwarm. (vgl. Buckland, 2005:118)

Selbstverständlich gibt es nicht nur eine Art von Schwarm Verhalten, sondern viele verschiedene. In dieser Arbeit wird das Schwarmverhalten pauschal gehalten. Es geht lediglich um eine allgemeine Simulation eines Fisch Schwarms.

Es gibt diverse andere Ansätze, das Schwarmverhalten aus physikalischer Sicht zu behandeln. „Toner und Tu" untersuchten beispielsweise das Schwarmverhalten mit der Methode der Strömungsdynamik.
„Hierbei tritt aber das einzelne Partikel zurück, um einer gesamtheitlichen Beschreibung eines Schwarms als fließendes Gebilde in einem viskotischen Medium Platz zu machen."[5]
Einen anderen Ansatz wählte Reza Olfati-Saber in seiner Arbeit. Da er auf Theoreme Bezug nimmt, verwendete er zur Untersuchung der Schwarmformation eine Gitterstruktur, auf deren Knoten er die Schwarmstruktur anordnete (vgl. Kramper, 2010:9).

> „Brian L. Partridge konnte bei Untersuchungen an Elritzen (ein kleiner Süßwasserfisch) zeigen, dass bei zwei Fischen stets einer führt und der anderen folgt). Der Folgende passt sich in Tempo und Richtung dem Führer an, während der Führer in keiner Weise auf den anderen achtet. Dies mag wie ein Führungsprinzip aussehen, jedoch bricht es schon auseinander, sobald eine dritte Elritze hinzukommt. Jetzt richten sich die Tiere untereinander aus und einen Führer kann man nicht mehr ausmachen. Es entsteht das Schwarmprinzip: die Gesamtheit führt und jedes Individuum passt sich ihr an."[6]

[4]Unity, 2015:o.S
[5]Kramper, 2010:8
[6]Kramper, 2010:2

2.2 Alignment

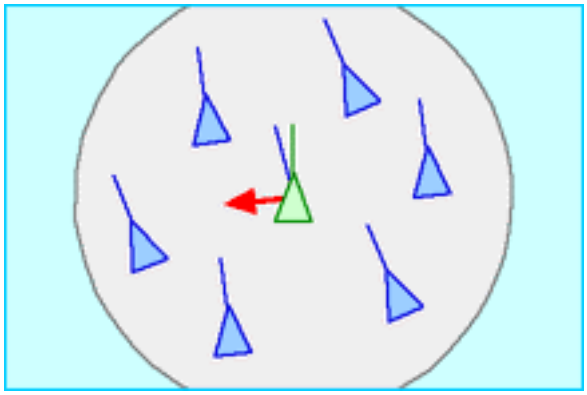


Abbildung 1: Alignment[7]

Das Alignment ist dafür zuständig, dass die Boids sich in eine vorgegebene Richtung und an einer vorgegebenen Geschwindigkeit orientieren und somit die Positionsänderung vorgenommen wird. Es wird beim Alignment dafür gesorgt, dass der individuelle Boid sich bei seinen Nachbar-Boids möglichst realistisch einreiht.

„Alignment is a behavior that causes a particular agent to line up with agents close by.“[8]

2.3 Separation

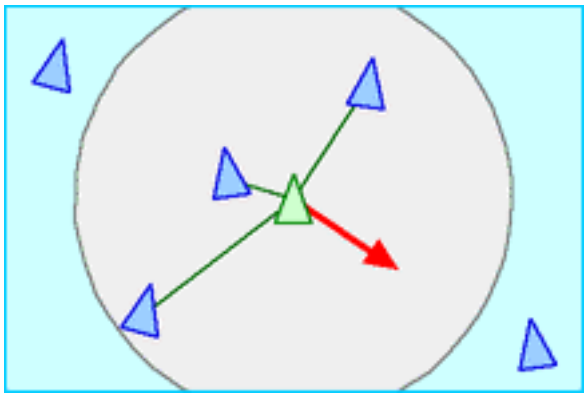


Abbildung 2: Separation[9]

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 (siehe Abb.2).

„Separation is the behavior that causes an agent to steer away from all of its neighbors.“[10]

[7]Quelle siehe Abbildungsverzeichnis
[8]Pemmaraju, 2013:o.S.
[9]Quelle siehe Abbildungsverzeichnis
[10]Pemmaraju, 2013:o.S.

2.4 Cohesion

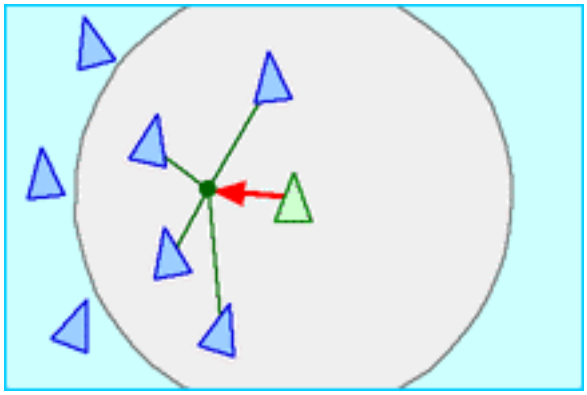


Abbildung 3: Cohesion[11]

Die Cohesion wird verwendet, um zu verhindern, dass die Boids sich zu weit voneinander entfernen, so dass der Schwarm keine Boids verliert. Die Verbindung von Cohesion und Separation wirkt sich somit auf den Abstand zwischen den Boids aus, sowie auf die Geschwindigkeit der Distanzveränderung zwischen den Boids im Schwarm (siehe Abb.3).

„Cohesion is a behavior that causes agents to steer towards the ‚center of mass' - that is, the average position of the agents within a certain radius."[12]

2.5 Impact

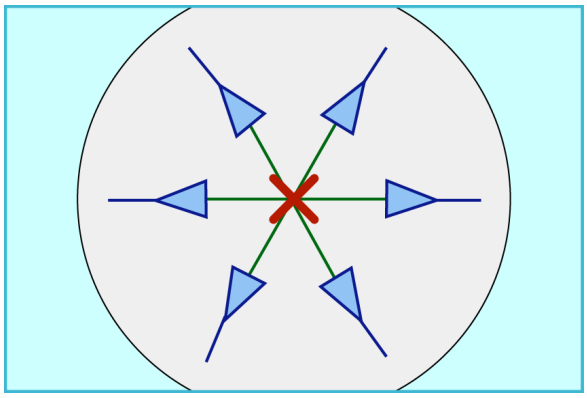


Abbildung 4: Impact[13]

Die Impact Simulation ist dafür verantwortlich, Interaktionen mit dem Schwarm realistisch zu gestalten. Dies würde unter anderem bei einem Haiangriff auf einen Fischschwarm zu finden sein (siehe Abb. 4). Es gibt noch weitere Arten der Flucht nach so einem Impact, beispielsweise die *Homogenitätsbedingung*. Bei der *Homogenitätsbedingung* entfernen sich nicht nur alle Boids vom Eintrittspunkt (mit rotem Kreuz auf Abbildung 4 gekennzeichnet), sondern auch von jedem umliegenden Boid.

> „Das zur Gruppenbedingung entgegengesetzte Bestreben, die Nähe anderer Agenten zu meiden, wird mit der Homogenitätsbedingung beschrieben. Hierbei versucht jeder Agent sich möglichst so weit wie möglich von allen anderen zu entfernen - und somit letztendlich eine homogene Verteilung aller Agent zu bewirken. Diesen Zusammenhang kann man sich veranschaulichen, wenn man bei jedem Agenten Abstoßungskräfte am Wirken sieht, die umso stärker sind, je geringer der Abstand zwischen zwei Agenten ist."[14]

[11]Quelle siehe Abbildungsverzeichnis
[12]Pemmaraju, 2013:o.S.
[13]Quelle siehe Abbildungsverzeichnis
[14]Kramper, 2010:19

3 Methodik

Bevor der Schwarm und die Performance getestet werden kann, muss untersucht werden, wie ein Schwarm auf einen Impact reagieren würde. In der Tierwelt ist dies besonders gut zu beobachten. Wenn zum Beispiel in einem Fischschwarm ein Fremdkörper, wie ein Hai oder ein Stein auf der Wasseroberfläche, auftaucht, zersprengt sich der Schwarm sofort. Er braucht dann einige Zeit, bis er sich wieder zusammen fügt.
„All the boids can be moving in one direction at one moment, and then the next moment the tip of the flock formation can turn and the rest of the flock will follow as a wave of turning boids propagates through the flock. Reynolds' implementation is leaderless in that no one boid actually leads the flock"[15]

3.1 Engine Auswahl

> „Unity wird üblicherweise als Game-Engine (deutsch Spiel-Engine) bezeichnet, und das ist auch korrekt. Beschreibender wäre aber Entwicklungswerkzeug zur Erstellung von 2D- und 3D-Spielen und -Anwendungen für eine Vielzahl unterschiedlicher Zielplattformen einschließlich Virtual-Reality-Umgebung. Streng genommen umfasst die Game-Engine an sich vor allem die einzelnen für Spiele notwenigen Systeme (wie Grafik, Physik, Audio, Steuerung und Skripting) sowie teilweise auch weiterführende Systeme, beispielsweise Netzwerkunterstützung oder auch das Speichern von Spielständen. Bis auf Letzteres (das Speichern von Spielständen) hat Unity das auch alles mit dabei."[16]

Es wird die Unity Engine (Version 5.3.2) benutzt, da Unity durch diverse Möglichkeiten dem Benutzer das Implementieren erleichtert. Besonders erwähnenswert ist der Profiler, mit dem sich die Performance permanent über ein extra Fenster kontrollieren lässt (vgl Unity, 2015:o.S.).
Desweiteren bietet Unity eine umfangreiche und offizielle Dokumentation[17] .
Ein weiterer wichtiger Punkt ist die kostenfreie Nutzung von Unity 3D für unkommerzielle Projekte.

3.2 Programmiersprache

> „Die Sprache Boo ist eine Programmiersprache mit Python-ähnlicher Syntax [...]. Inzwischen kann Boo auch für Android- und iOS-Spiele eingesetzt werden (das war am Anfang nicht so). *JavaScript* in Unity sollte eigentlich eher *UnityScript* heißen, und tatsächlich heißt die Sprache aus so, wurde aber lange Zeit von Unity Technologies offiziell als JavaScript bezeichnet. Es handelt sich auch um eine zumindest von der Syntax her an JavaScript angelehnte Sprache. [...] Wenn man zu Unity umsteigt und noch nicht mit Unity vertraut ist, gibt JavaScript-Code dem Einsteiger meistens relativ wenig Anhaltspunkte, um zu verstehen, welche Klassen genau verwendet werden. [...] Ein weiteres Problem für den Programmieranfänger ist, dass JavaScript in Unity eben nicht wirklich Java ist. Daraus resultiert, dass die meisten JavaScript-Tutorials eher Verwirrung stiften."[18]

Um die Performance des Schwarms unter Anwender-Interaktion vergleichen und testen zu können, muss diese erst implementiert werden. Durch die eben beschriebene Engine-Auswahl, gibt es nun die Programmierspra-

[15]Bourg, 2004:86
[16]Chittesh, 2015:1
[17]http://docs.unity3d.com
[18]Chittesh, 2015:115, Hervorheb. i.O.

chen Java, C#, UnityScript und Boo zum Implementieren des Schwarmverhaltens. Aus vielerlei Gründen wurde hier die Programmiersprache C# gewählt.

> „Bei C# steht in Unity im Gegensatz zu JavaScript/UnityScript nicht nur C# drauf, sondern es ist auch wirklich C# drin, und zwar genau so, wie man es aus der .NET- und Mono-Welt kennt. Somit kann man sich auch für Unity auf die offizielle Sprachspezifikation beziehen und findet auch ein entsprechendes Programmierhandbuch. Außerdem gibt es eine Vielzahl von Tutorials und anderen Möglichkeiten, die Sprache zu lernen; teilweise auch im Unity-Umfeld.“[19]

Durch das Microsoft Developer Network (MSDN) wird auch hier eine zuverlässige und offizielle Dokumentation der Programmiersprache und des .NET Frameworks ermöglicht. Desweiteren lassen sich mit C# auch neuere Softwareentwicklungen umsetzen, da die Sprache fortlaufend weiterentwickelt wird.

Eine Umfrage im Unity-Forum hat ergeben, dass von den 482 Teilnehmern ausschließlich C# verwenden, 29,8% benutzen ausschließlich Java und 28,4 verwenden sowohl JavaScript als auch C# in Unity. Die Popularität der Sprache ist insofern relevant, dass wenn man die Programmierung nicht komplett alleine erledigen möchte oder Fragen aufkommen, diese von der größeren Masse der Benutzer im Forum besser beantwortet bekommt (vgl. Chittesh, 2015:116).

3.3 Variablen Benennung

„Für die Benennung von Variablen sollten Sie sich eine bestimmte Konvention aneignen, um Unklarheiten hinsichtlich Ihrer selbst definierten Variablen von vorherein zu vermeiden“[20]
Es gilt jedoch, trotz eigens angeeignetem Style, der Richtigkeithalber die Common Language Infrastructure einzuhalten. Die Begründung der Wichtigkeit folgt im nächsten Abschnitt.

„ISO/IEC 23271:2003 defines the Common Language Infrastructure (CLI) in which applications written in multiple high-level languages may be executed in different system environments without the need to rewrite the applications to take into consideration the unique characteristics of those environments.“[21]

Es gibt diverse Gründe, warum es wichtig ist die CLI einzuhalten. Etwa, dass Variablen nicht nur durch unterschiedliche Groß- und Kleinschreibung deklariert werden, sondern auch mit verschiedenen Namen, da Case Insensitive Programmiersprachen diese andernfalls nicht unterscheiden können.
„Bezeichner die sich nur durch die Groß-/Kleinschreibung unterscheiden, können die Wiederverwendung von Klassen behindern, wenn Sie in Ihren Anwendungen auch andere Sprachen wie Visual Basic nutzen, bei denen die Groß-/Kleinschreibung keine Rolle spielt.“[22]

[19]Chittesh, 2015:116
[20]Frischalowski, 2008:52
[21]ISO, 2003:o.S.
[22]Frischalowski, 2008:52

3.4 Entwicklungsumgebung

Um die C# Bibliotheken und Anwendungen zu erstellen, wird in der Theorie lediglich ein Compiler und ein Texteditor benötigt. Jedoch beschleunigt eine professionelle Entwicklungsumgebung das Entwicklungstempo und den erhöht Komfort. Die bereits mitgelieferte Entwicklungsumgebung heißt MonoDevelop und sie hat einige Vorteile. MonoDevelop unterstützt alle von Unity unterstützen Programmiersprachen und Plattformen, also C#, JavaScript und Boo. In diesem Projekt wurde sich jedoch aus persönlichen Gründen gegen die Arbeit mit MonoDevelop entschieden.

„MonoBehaviour is the base class every script derives from. Using Javascript every script automatically derives from MonoBehaviour. When using C# or Boo you have to explicitly derive from MonoBehaviour.“[23]
Desweiteren unterstützen Entwicklungsumgebungen das automatische Vervollständigen des Codes, die sofortige Markierung von Syntax-Fehlern und die API-Dokumentation Einbindung.

„Visual Studio ist die Anwendung zur Entwicklung in C# oder auch in anderen .NET-sprachen, wie Managed C++ oder Visual Basic. Aber auch klassische Sprachen wie C oder C++ lassen sich sehr gut mit Visual Studio von Microsoft entwickeln.“[24]
Aus verschiedenen Gründen wurde die Microsoft Visual Studio Community 2015 Edition gewählt. Diese Entwicklungsumgebung wird ab Unity 5.2 kostenfrei mitgeliefert und stetig aktualisiert (vgl. Unity, 2015:o.S.).
„Der wesentliche Vorteil von Visual Studio besteht darin, dass dieses Werkzeug schon seit vielen Jahren entwickelt und konsequent verbessert wird und auch durch Plug-Ins von Drittherstellern erweiterbar ist.“[25]

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

Ein weiterer Vorteil, eine professionelle Entwicklungsumgebung zu benutzen, ist der Debugger, der dem Entwickler hilft Fehler zu finden und den Code besser zu verstehen.

> „The Visual Studio debugger helps you observe the run-time behavior of your program and find problems. The debugger works with all Visual Studio programming languages and their associated libraries. With the debugger, you can break execution of your program to examine your code, examine and edit variables, view registers, see the instructions created from your source code, and view the memory space used by your application.“[26]

[23]Unity, 2015:o.S.
[24]Wurm, 2010:99
[25]Chittesh, 2015:117
[26]Microsoft Developer Network, 2016:o.S.

3.5 Versionsverwaltung

> „Sind Sie in der glücklichen Situation, in einem kleinen Team arbeiten zu können, indem die verschiedenen zur Spielentwicklung notwendigen Talente vereint sind? Dann stellt sich natürlich die Frage, wie mehrere Leute möglichst reibungslos und am besten gleichzeitig an einem Projekt arbeiten können. Die Standardantwort darauf kommt aus der Softwareentwicklung und heißt *Versionsverwaltung*. Und das ist gleichzeitig durchaus auch für Einzelentwickler eine sehr nützliche Sache, wenn auch aus anderen Gründen."[27]

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

> „Vereinfacht gesagt bestehen Systeme zur Versionsverwaltung aus einem zentralen Repository, in dem der aktuelle Projektstand sowie die gesamte Versionshistorie abgelegt sind, so wie einer Software, die den Zugriff auf dieses Repository ermöglicht. Dabei arbeitet man mit einer lokalen Arbeitskopie des Projektes, die sich vom aktuellen Stand des Repository unterscheidet, aber mit verschiedenen Aktionen synchronisiert werden kann."[28]

4 Durchführung

Um zu testen ob der Algorithmus wie gewollt funktioniert, ist eine optische Prüfung notwendig. Diese optische Prüfung wird beim Praxis Test zeigen, ob die Implementierung des Schwarm-Algorithmus erfolgreich zum gewünschten Ziel geführt hat.
Um die Performance des Algorithmus testen zu können, muss dieser Algorithmus jedoch 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 gerade im 2D Bereich sehr von Vorteil. Die *NavMesh* Agent Klasse besitzt die Funktion *Destination*, die es dem Agent erlaubt, zum Zielobjekt zu laufen. Dies ist hilfreich bei Schwarmverhalten mit deinem Leader-Boid (vgl. Unity, 2015:o.S.).

[27]Chittesh, 2015:126, Hervorheb. i.O.
[28]Chittesh, 2015:127, Hervorheb. i.O.

> „Eine spezielle Komponente, über die ausnahmslos jedes GameObject verfügt, ist die sogenannte *Transform*-Komponente. Sie speichert neben den Positionsdaten (Position, Skalierung, Rotation) auch die hierarchischen Zusammenhänge zwischen den Objekten. Die hierarchische Struktur wird in der *Hierarchy View* dargestellt und bearbeitet. Die Positionsdaten sind natürlich in der Scene View zu erkennen, aber auch als numerischer Werte im *Inspector* dargestellt, und zwar jeweils in die Achsenkomponenten X,Y und Z dargestellt. [...] Beachten Sie dabei, dass hier alle Werte lokal sind, also relativ zum Elternobjekt, unter dem das bearbeitete Objekt hängt (sofern es nicht auf der höchsten Ebene liegt, also kein Elternobjekt hast).“[29]

Falls die Interaktion des Schwarmes es benötigt, ist es möglich, über die *LookAt* Funktion die Boids in Richtung des Impacts gucken zu lassen. Das ermöglicht es, anschließend mit dem Wert der *Transform.back* Funktion, die Bewegung einer Flucht zu simulieren. Ebenfalls ist es möglich mit der *LookAt* Funktion, die Kamera dynamisch auf den Schwarmmittelpunkt zu richten. Dies lässt die Boids zuerst auf den Impact Punkt schauen, bevor sie flüchten.

Nach der Anwender-Interaktion findet die erneute Gruppenfindung statt. Hierzu checkt jeder Boid sein Umfeld und bewegt sich zurück zu seinen Nachbarn.

> „Die Gruppenbedingung beschreibt das Vermögen, mit dem die Schwarmmitglieder sich zusammenziehen. Da jeder Agent die anderen in seiner Umgebung detektieren kann, so wird er auch eine Ansammlung von anderen als Häufung erkennen und je höher die Dichte dieser Häufung ist, desto attraktiver wird dieser Ort. Der attraktive Ort, zu dem man bei einer Zusammenziehung streben möchte, ist der Punkt des *geometrischen Schwerpunktes* im Zentrum des Schwarms.“[30]

Auch nimmt Unity dem Benutzer eine Vielzahl an mathematischen Berechnungen ab. Besonders wichtig hierbei sind die Vektor3-Klassen, vor allem im 3D Bereich.

Mit der Funktion *velocity* wird die Bewegungsgeschwindigkeit ermittelt. Diese Geschwindigkeit kann zur Überprüfung der erlaubten Maximalgeschwindigkeit des Boids genutzt werden. Die *velocity* wird gegen die vorher initialisierte Maximalgeschwindigkeit geprüft und falls überschritten, zurück auf die diese gedrosselt. Zum Drosseln muss die Funktion *Velocity* mathematisch mit der in der Funktion vorhandenen Funktion *normalized* normalisiert werden. Anschließend wird diese wieder, unter Berücksichtigung der *deltaTime*, mit der Maximalgeschwindigkeit multipliziert. Auf die deltaTime wird anschließend noch genauer eingegangen.

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.

[29]Chittesh, 2015:31, Hervorheb. i. O.
[30]Kramper, 2010:19, Hervorheb. i. O.

„In most cases you should not modify the velocity directly, as this can result in unrealistic behaviour. Don't set the velocity of an object every physics step, this will lead to unrealistic physics simulation."[31]

Das Unity Keyword Time ist ein Interface, welches genutzt wird, um Information bezüglich der Zeit zu erhalten. Die Funktion *deltaTime* dieses Interfaces sollte in Unity in der Regel mit allem multipliziert werden, dass einmal pro Frame etwas Addiert oder Subtrahiert, um die mathematische Berechnung frameunabhängig macht. „Use this function to make your game frame rate independent."[32]

Gerne genutzt im Bereich der *Time*-Klasse, 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.

Ein weiteres nützliches Tool im Bezug auf die Performance ist das *Level of Detail* (*LOD*). Hierbei handelt es sich um eine Technik, die es ermöglicht Performance zu sparen, indem Triangel reduziert werden. Je weiter das Objekt von der aktuell aktivierten Kamera entfernt ist, um so niedriger wird das *Level of Detail* gesetzt und umso mehr Triangels werden reduziert (vgl. Unity, 2015:o.S.).

> „When an object in the scene is a long way from the camera, the amount of detail that can be seen on it is greatly reduced. However, the same number of triangles will be used to render the object, even though the detail will not be noticed. An optimisation technique called Level Of Detail (LOD) rendering allows you to reduce the number of triangles rendered for an object as its distance from camera increases. As long as your objects aren't all close to the camera at the same time, LOD will reduce the load on the hardware and improve rendering performance."[33]

[31,32,33]Unity, 2015:o.S.

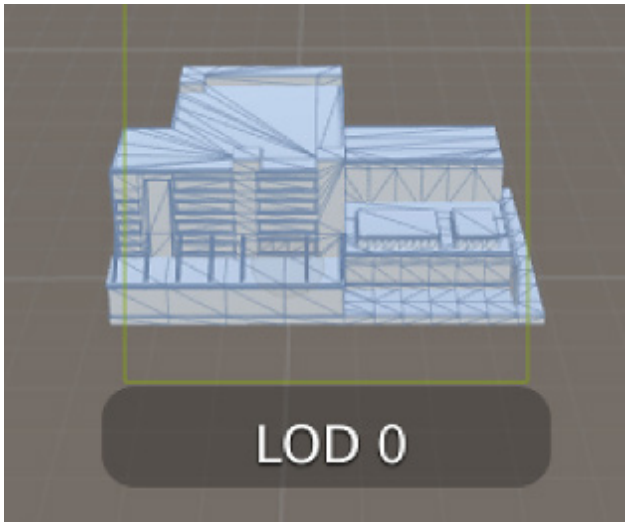


Abbildung 5: Gebäude Mesh mit LOD der Stufe 0[34]

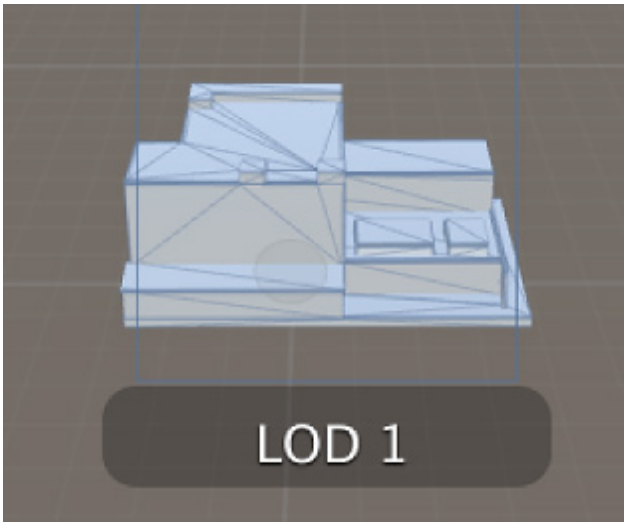


Abbildung 6: Gebäude Mesh mit LOD der Stufe 1[35]

Auf Abbildung 5 ist das Objekt mit voller Triangel Anzahl zu sehen (*LOD 0*). Je Höher die *LOD*, des so niedriger die Triangel Anzahl.

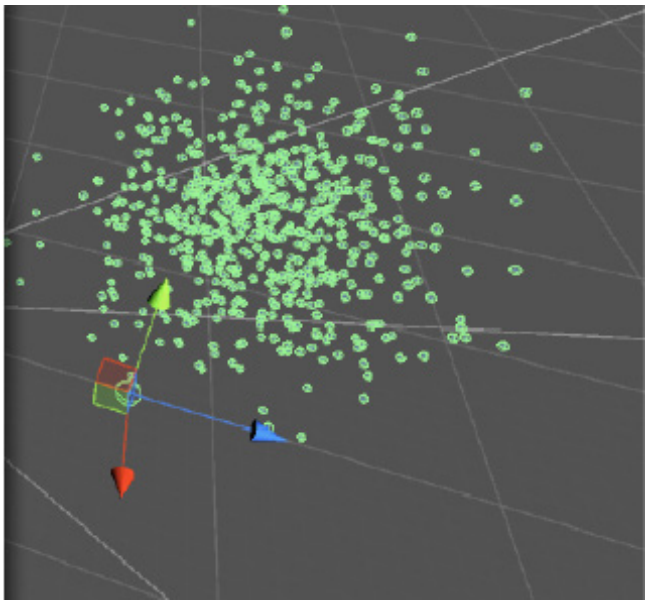
Bei Abbildung 6 wurden viele Triangels eingespart, deren Details bei einer weiter Distanzierten Kamera nicht-mehr sichtbar wären.

Desweiteren kann bei einem hohen *LOD* der *Collider* deaktiviert werden, was sehr zugute der Performance sein kann.

> „If you create a set of meshes with names ending in _LOD0, _LOD1, _LOD2, etc, for as many LOD levels as you like, a LOD group for the object with appropriate settings will be created for you automatically on import. For example, if the base name for your mesh is Player, you could create files called Player_LOD0, Player_LOD1 and Player_LOD2 to generate an object with three LOD levels. The numbering convention assumes that LOD 0 is the most detailed model and increasing numbers correspond to decreasing detail.“[36]

[34,35]Quellen siehe Abbildungsverzeichnis
[36]Unity, 2015:o.S.

4.1 Praxis Test



Auf Abbildung 7 zu sehen ist der Versuch der Implementierung eines möglichst realen Schwarms, 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*, einen *Shader Component* und einen *Transform Component*.

Die *Rigidbody* Komponente ist dafür zuständig, dass das GameObjekt mit dieser Komponente als physisches Objekt gewertet wird.

Abbildung 7: Schwarm in einer Unity Scene[37]

> „A Rigidbody is the main component that enables physical behaviour for an object. With a Rigidbody attached, the object will immediately respond to gravity. If one or more Collider components are also added then the object will be moved by incoming collisions.
> Since a Rigidbody component takes over the movement of the object it is attached to, you shouldn't try to move it from a script by changing the Transform properties such as position and rotation. Instead, you should apply forces to push the object and let the physics engine calculate the results."[38]

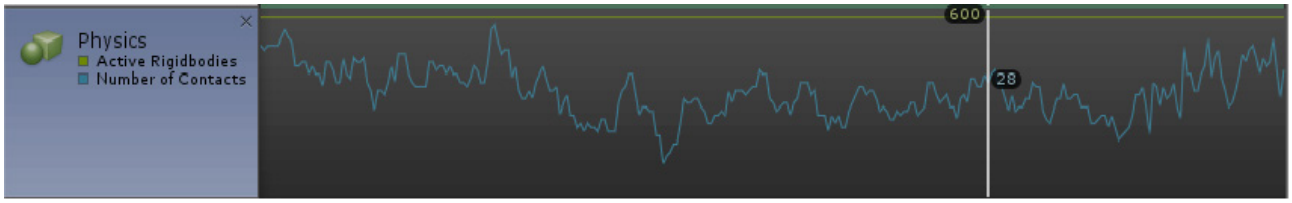


Abbildung 8: Unity Profiler mit 600 aktiven Boids[39]

Abbildung 8 zeigt die 600 aktiven *Rigidbodys* der vorherigen Abbildung und deren Kontakten per Frame. Wie auf dem Diagramm zu sehen ist, gibt es durchschnittlich 10 bis 60 Kontakte unter allen Boids pro Frame. Wenn der Schwarm sich konstant in eine gleichbleibende Richtung bewegt, kommt es zu den zu sehenden durchschnittlichen zehn Kontakten, bei raschen Bewegungsänderungen erhöht sich der Kontakt per Frame auf 60.

[37,39]Quelle siehe Abbildungsverzeichnis
[38]Unity,2015:o.S.

„Collider components define the shape of an object for the purposes of physical collisions. A collider, which is invisible, need not be the exact same shape as the object's mesh and in fact, a rough approximation is often more efficient and indistinguishable in gameplay."[40]

Wenn sich die Boids normal verhalten, nennt man das „Die Normalitätsbedingung". Dabei bewegt sich jeder Boid mit einer gewissen Geschwindigkeit vorwärts. Auch wenn kein anderer Agent in seiner Umgebung vorhanden wäre, so ist dies sein „Grundverhalten" und daher ist diese Geschwindigkeit als Default-Wert der Normalgeschwindigkeit mit 1 zu deklarieren. Dieser Wert könnte bei einer Flucht erhöht werden (vgl. Kramp, 2010:19).

Abbildung 4, aus dem Kapitel 2.5 Impact, stellt die Anwender-Interaktion dar. Der Schwarm simuliert auf Kommando ein Szenario, wie beispielsweise einen Fischschwarm, der von einem Hai angegriffen wird. Dieses Kommando kann unter anderem ein Maus-Klick in den Schwarm sein, oder das Auslösen eines Hotkeys.

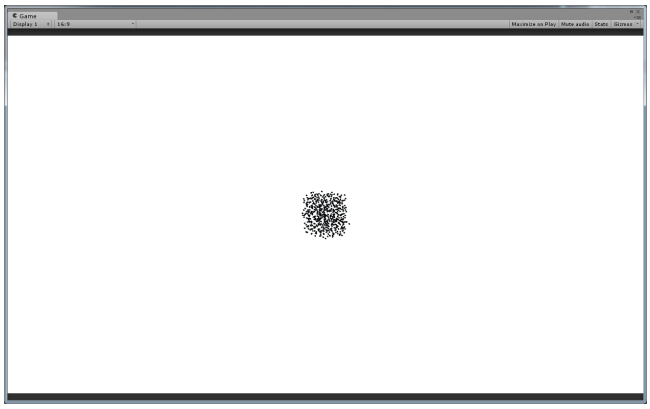


Abbildung 9: Status: Ausgangsposition[41]

Auf Abbildung 9 bleibt der Schwarm noch zusammen, da noch keine Interaktion passiert ist. Dies ist die Ausgangsposition mit 10 bis 60 Kontakten pro Frame.

Abbildung 10: Status: Impact[42]

Die Anwender-Interaktion passiert: Der Schwarm entfernt sich vom Eintrittsort des Impacts (siehe Abb. 10). Von Abbildung 10 , bis hin zu Abbildung 13 gibt es nahezu keine Boid Kontakte pro Frame, da sich die Boids auch untereinander leicht voneinander entfernen.

[40]Unity,2015:o.S.
[41,42]Quellen siehe Abbildungsverzeichnis

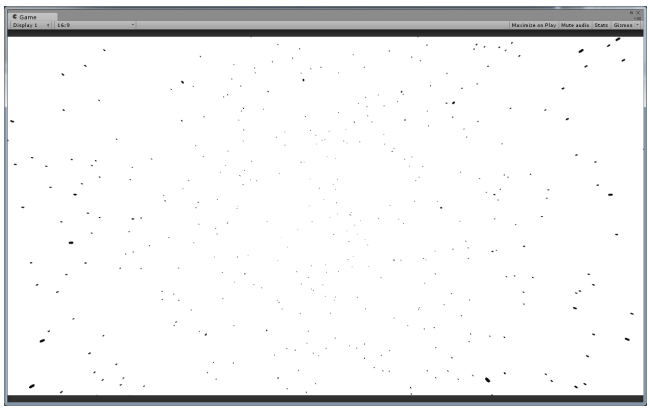


Abbildung 11: Status: Maximale Flucht Distanz[43]

Die individuellen Boids haben ihren maximalen Fluchtpunkt erreicht. Dies kann auf verschiedene Weisen passieren, zum Beispiel wenn die Boids eine Abstandsberechnung zum Impact haben, oder über eine Coroutine (siehe Abb.11).

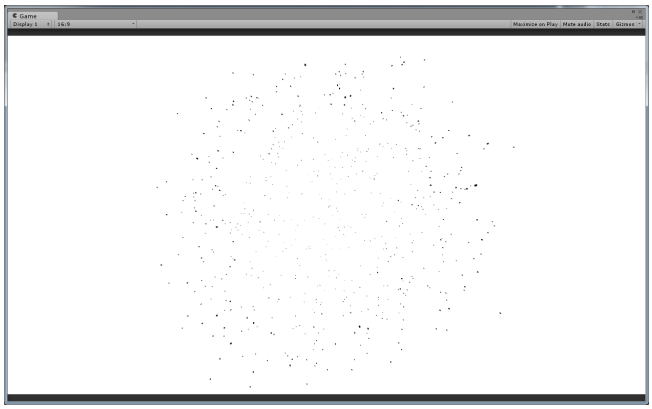


Abbildung 12: Status: Rückkehr[44]

Die Boids bewegen sich individuell erneut Richtung Schwarm-Mittelpunkt (siehe Abb. 12).

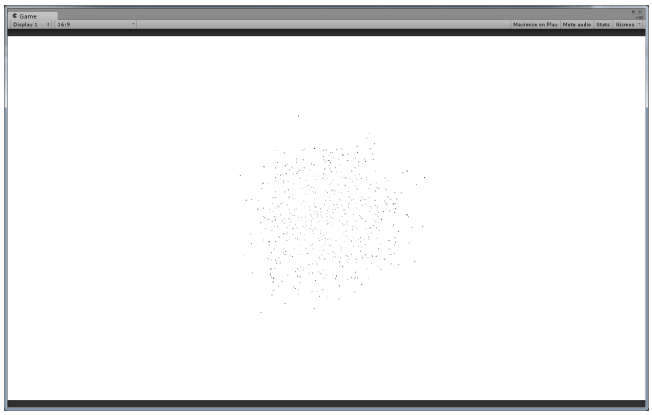


Abbildung 13: Status: Geschwindigkeitsdrosselung[45]

Der Mittelpunkt des Schwarms wurde von den Boids fast erreicht (siehe Abb. 13). Dadurch kommt es zu einer Geschwindigkeitsdrosslung, womit vermehrter Kontakt zwischen den Boids wieder möglich ist.

[43,44,45]Quellen siehe Abbildungsverzeichnis

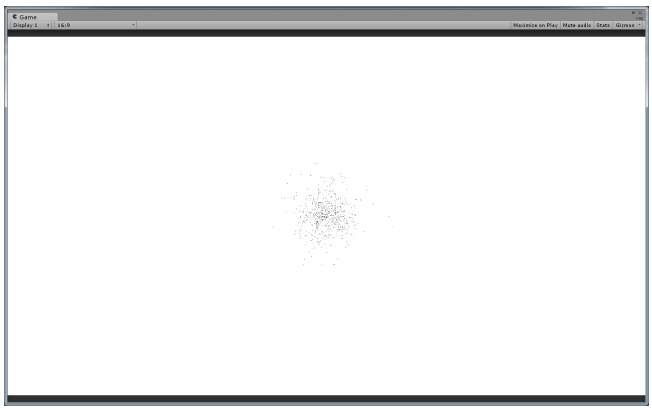


Abbildung 14: Status: Impact abgeschlossen[46]

Die Ausgangsposition ist fast erreicht und es pendeln sich wieder die 10 bis 60 Kontakte pro Frame ein. Die Anwender-Interaktion ist somit abgeschlossen (siehe Abb. 14).

Zuletzt im Bereich des Praxis Tests sollten die Quality Settings nicht vernachlässigt werden. Unter Menu: Edit > Project Settings > Quality können diese eingestellt werden. In diesen Quality Settings kann geregelt werden, ob das Projekt zugunsten der Optik an Performance verlieren soll, oder andersrum.

„Unity allows you to set the level of graphical quality it will attempt to render. Generally speaking, quality comes at the expense of framerate and so it may be best not to aim for the highest quality on mobile devices or older hardware since it will have a detrimental effect on gameplay.“[47]

Viele Einstellungen sind eher individuell auf das Projekt zuzuschneiden, etwa das Anti-Aliasen (AA). Auch hier muss vorab eingeschätzt werden, ob der visuelle Unterschied die Performancesenkung rechtfertigt. Beim AA, auch Kantenglättung genannt, berechnet die Grafikkarte ein höher aufgelöstes Bild und verrechnet dieses mit ihrer wirklichen Auflösung. Dies erzeugt ein weicheren Übergang zwischen der Umgebung und dem Objekt.

> „Anti aliasing improves the appearance of polygon edges, so they are not ‚jagged‘, but smoothed out on the screen. However, it incurs a performance cost for the graphics card and uses more video memory (there's no cost on the CPU though). The level of anti-aliasing determines how smooth polygon edges are (and how much video memory does it consume).“[48]

[46]Abbildung siehe Quellenverzeichnis
[47,48]Unity, 2015:o.S.

5 Ergebnisse

Abbildung 15: Unity Profiler bei Impact[49]

Nach Implementierung des Schwarm-Algorithmus ist auf Abbildung 15 zu sehen, dass bei der Anwender-Interaktion die Kollision pro Frame nahezu null ist. Das bedeutet, dass Unity intern keine zusätzlichen physikalischen Berechnungen machen muss, was der Performance zugutekommt.

Wie in Kapitel 4.1 schon angesprochen und auf Abbildung 8 zu sehen, sind bei realistischem Verhalten des Schwarmes durchschnittlich 10 bis 60 Kontakte pro Frame, bei aktiviertem *Collider component*, zu beobachten.

Diese Ergebnisse wurden auf verschiedenen Rechnern erzielt und vermehrt überprüft.

5.1 Auswertung

Die Implementiert des Schwarm-Algorithmus war erfolgreich und wurde durch optische Tests (Abbildung 7) belegt.

Bei einer Schwarmgröße von 600 Boids mit den Unity Standard Objekten, gibt es zur Runtime keinerlei Veränderungen. Sichtbar wird dies erst bei höherer Anzahl an Boids. Bei hoher Boid Anzahl kann es beim Projekt Start zu Ladeverzögerungen kommen. Hierbei hilft jedoch das Unity interne *SceneManager.LoadSceneAsync*, oder das *Object Pooling*. Seid Unity 5.3 ersetzt das *SceneManager.LoadSceneAsync* das nun Obsolete *Application.LoadLevelAsync*. Mit der *SceneManager.LoadSceneAsync* Funktion ist es möglich, eine Unity Scene asynchron im Hintergrund laden zu lassen (vgl. Unity, 2015:o.S.).

Es wird empfohlen beim Instantiaten der Boids zu achten, dass diese jeweils mit einem Abstand erstellt werden, um anfängliche physikalische Berechnungen durch Kollision zu verringern.

Beim abrupten Richtungswechsel des Schwarmes ist es möglich, dass es zu vermehrter Kollision kommt. Daher ist es empfehlenswert diesen Richtungswechsel möglichst langsam und geschmeidig zu gestalten.

[49]Quelle siehe Abbildungsverzeichnis

6 Zusammenfassung

Wie lässt sich das Ganze zusammenfassend am besten sagen? Im Grunde wurde das originale Schwarmverhalten von Craig Reynolds übernommen, in Unity implementiert und mit den Unity Internen Tools die Performance gesteigert. Vor allem erwähnenswert ist das *Level of Detail*, dass Triangels einspart, falls die Kamera weit entfernt ist, die *Vector*-Klassen die bei den Berechnungen behilflich sind und natürlich die Unity Dokumentation.

6.1 Fazit

Dass Unity eine leistungsfähige Engine ist, ist wohl kein Geheimnis. Da Craig Reynolds schon Mitte der 1980er Jahre die wohl erste Simulation eines Schwarmverhaltens dargestellt hat, überrascht es auch weniger, dass es möglich ist, in Unity ein durch Anwender-Interaktion beeinflussbares Schwarmverhalten performant zu implementieren. Jedoch war es spannend, nicht nur den Algorithmus programmiertechnisch performant zu gestalten, sondern sich mit den Unity internen Tools auseinander zu setzen, um das Gesamtbild des Schwarmverhaltens zu verbessern. Jedoch sollte man bedenken, dass der Versuch, wie auch schon in der Einleitung erwähnt wurde, die Zielplattform der Implementierung in Hinsicht auf die Leistung, den durchschnittlichen Personal Computer hat. Leistungsschwächere Geräte wie beispielsweise mobile Endgeräte können durchaus Probleme in Hinsicht der Performance haben.

Die codetechnische Umsetzung war unkompliziert und einfach. Auch wenn Schwärme meiner Meinung nach eher selten in Spielen zur Schau gestellt werden, ist es durchaus empfehlenswert einen einfachen Schwarm-Algorithmus zu implementieren.

7 Literaturverzeichnis

Bourg, David M. / Glenn, Seeman	AI for Game Developers. Sebastopol: O'Reilly, 2004
Buckland, Mat	Programming Game AI by Example. Texas: Wordware Publishing, 2005
Chittesh, Jashan	Das Unity-Buch. 2D- und 3D-Spiele entwickeln mit Unity 5. Heidelberg: pdunkt.verlag, 2015
Dudenredaktion	Duden. Mannheim: Die Dudenredaktion, April 2006
Frischalowski, Dirk	Visual c# 2008 Einstieg für Anspruchsvolle. München: Pearson Studium, 2008
ISO	2013: Common Language Infrastructure Online unter: http://www.iso.org/iso/catalogue_detail?csnumber=36769 (Zuletzt überprüft am 14.03.2016)
Kramper, Wolfgang	Simulation von Schwarmverhalten. Berlin: Mensch und buch Verlag, 2010
Unity	2015: Profiler Physics Online unter: http://docs.unity3d.com/Manual/ProfilerPhysics.html (Zuletzt überprüft am 04.03.2016)
Unity	2015: Unity 5.2 – your gateway to Unity Services Online unter: http://blogs.unity3d.com/2015/09/08/unity-5-2-easy-access-to-unity-services/ (Zuletzt überprüft am 03.03.2016)
Unity	2015: Time.deltaTime Online unter: http://docs.unity3d.com/ScriptReference/Time-deltaTime.html (Zuletzt überprüft am 01.03.2016)

Unity	2015: Upgrade Guide 5.3 Online unter: http://docs.unity3d.com/Manual/UpgradeGuide53.html (Zuletzt überprüft am 10.03.2016)
Unity	2015: Level of Detail Online unter: http://docs.unity3d.com/Manual/LevelOfDetail.html (Zuletzt überprüft am 16.02.2016)
Unity	2015: Quality Settings Online unter: http://docs.unity3d.com/Manual/class-QualitySettings.html (Zuletzt überprüft am 04.02.2016)
Unity	2015: Scripting Online unter: http://docs.unity3d.com/Manual/ScriptingSection.html (Zuletzt überprüft am 07.02.2016)
Unity	2015: Mono Behaviour Online unter: http://docs.unity3d.com/ScriptReference/MonoBehaviour.html (Zuletzt überprüft am 09.02.2016)
Metz, Felix	2013: Schwarmverhalten in der Informatik Online unter: http://felixmetzfacharbeit.de/schwarmverhalten.html (Zuletzt überprüft am 13.03.2016)
Microsoft Developer Network	2016: Debuggen in Visual Studio Online unter: https://msdn.microsoft.com/de-de/library/sc65sadd.aspx (Zuletzt überprüft am 09.03.2016)

Pemmaraju, Vijay 2013: Implementation. The Three Simple Rules of Flocking Behaviors: Alignment, Cohesion, and Separation
Online unter:
http://gamedevelopment.tutsplus.com/tutorials/the-three-simple-rules-of-flocking-behaviors-alignment-cohesion-and-separation--gamedev-3444
(Zuletzt überprüft am 24.02.2016)

Vehlken, Sebastian Zootechnologien. Zürich: Diaphanes, 2012

Wurm, Bernhard Programmieren lernen! Bonn: Galileo Press, 2010

8 Abbildungsverzeichnis

Abbildung 1-2	Seite 6 Alignment, Separation Online unter: http://www.red3d.com/cwr/boids/ (Zuletzt überprüft am 08.03.2016)
Abbildung 3	Seite 7 Cohesion Online unter: http://www.red3d.com/cwr/boids/ (Zuletzt überprüft am 08.03.2016)
Abbildung 4	Seite 7 Impact Erstellt von Katharina Schneiders
Abbildung 5-6	Seite 14 Gebäude Mesh mit LOD der Stufe 0 und 1 Online unter: http://docs.unity3d.com/Manual/LevelOfDetail.html (Zuletzt überprüft am 08.03.2016)
Abbildung 7	Seite 15 Schwarm in der Unity Scene Selbst erstellt
Abbildung 8	Seite 15 Unity Profiler mit 600 Boids Selbst erstellt
Abbildung 9-10	Seite 16 Schwarm Status Selbst erstellt

Abbildung 11-13 Seite 17
Schwarm Status
Selbst erstellt

Abbildung 14 Seite 18
Schwarm Status
Selbst erstellt

Abbildung 15 Unity Profiler bei Impact
Selbst erstellt

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Hamburg, 14.03.2016 Julian Hopp