HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Implementierung einer einfachen Schwarmintelligenz mithilfe des Boid Algorithmus

Implementation of a simplified swarm intelligence with the boid algorithm

Fabio Gimmillaro

Projektarbeit

Betreuer: Prof. Dr. Christoph Rezk-Salama

Trier, den 28. Februar 2018

Inhaltsverzeichnis

1

Einleitung

Eine Schwarmintelligenz umfasst das kollektive Denken einer Gruppe von mehreren Individuen. Viele kennen den Begriff in Verbindung mit bestimmten Tieren, wie Vögel oder Fische, die sich fast synchron in einer riesigen Gruppe bewegen.
Da dieses Phänomen sehr interessant ist, gibt es mehrere Ansätze, um es zu simulieren. Eines davon ist der Boid Algorithmus, der sicherstellt, dass sich die einzelnen Instanzen eines Schwarms im Flug ausrichten und so nah wie möglich beieinander bleiben, um zusammen in eine bestimmte Richtung zu fliegen.
Im Folgenden wird die Implementierung dieses Algorithmus und das Verhalten des Schwarms beschrieben.

2

Implementierung

Um ein Schwarmverhalten zu simulieren, habe ich mich für die Unreal Engine 4 entschieden, da diese viele Vorteile in Sachen Performanz liefert. Das Projekt wurde größtenteils in der Programmiersprache C++ entwickelt. Die Verhaltenslogik des Schwarms wurde mithilfe des knotenbasierten *Behavior Tree* aus der Engine zusammengebaut.

2.1 Aufbau

In der Unreal Engine 4 gibt es sogenannte ***Controller***, die für die Kontrolle eines ***Pawns*** zuständig sind. Diese Controller kommen für alle Arten von Objekte in Frage, die durch externe Einflüsse wie Behavior Trees oder Tastatur -und Controllereingaben gesteuert werden.
Im Folgenden werden alle Klassen beschrieben, die für die Kontrolle eines Schwarms zuständig sind. Dabei ist zu beachten, dass nicht alle Klassen nur in C++, sondern auch mithilfe von sogenannten Blueprints, implementiert wurden. Diese Blueprints sind eine Node-basierte Programmiersprache, die mithilfe des Unreal Reflection Systems zusammengebaut werden können.
Objekte, die in einem Level vorhanden sind, werden ***Actors*** genannt. Diese werden mithilfe des Unreal internen *Entity Component System* zusammengebaut. Die oben erwähnten Pawns (Klasse APawn) sind von der Klasse AActor abgeleitet. Diese haben den Vorteil, dass sie von einer künstlichen Intelligenz gesteuert werden können.

2.2 Die Klasse *USwarmInstancedComponent*

Diese Klasse repräsentiert den kompletten Schwarm und ist auch für die Bewegung aller einzelnen Individuen, welche durch einfach Sphären dargestellt werden, zuständig. Da es Tausende von Individuen geben kann, werden diese durch die ***USwarmInstancedComponent***, welche von der sogenannten ***UInstancedStaticMeshComponent*** erbt, instanziiert. Das heisst, sie teilen sich einen Draw Call, um die Performanz des Spiels drastig zu erhöhen.

2.2.1 Die Struktur *FSwarmInstance*

Jedes Individuum wird intern von einem FSwarmInstance struct repräsentiert, das einzelne Variablen besitzt, um ein Individuum zu bewegen oder rotieren.
Diese ist wie folgt zusammengebaut:

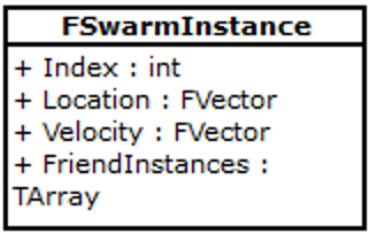


Abbildung 2.1. UML FSwarmInstance

Index:
Dieser Wert repräsentiert den Index, den diese Instanz in der Instanzliste eines ***UInstancedStaticMeshComponent*** besitzt

Location:
Die momentane Position in Weltkoordinaten

Velocity:
Der momentane Geschwindigkeitsvektor, mithilfe dessen auch die Rotation gesetzt werden kann.

FriendInstances:
Andere Instanzen, die sich momentan in der Nähe dieser Instanz befinden.

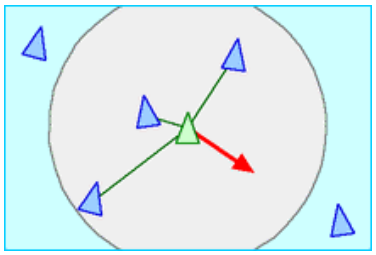
2.2.2 Aktualisierung der Bewegung eines Individuums

Um ein Individuum zu bewegen und zu rotieren werden zwei Ansätze benutzt:

Boid Algorithmus

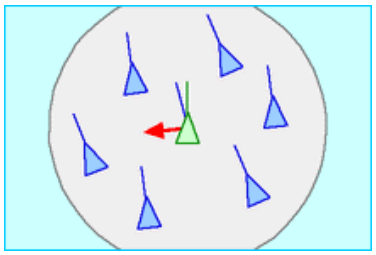
Dieser Algorithmus bestimmt Regeln, die die Transformation eines Individuums manipuliert. Im Folgenden werden die drei Regeln beschrieben, die normalerweise bei einem Boid Algorithmus eingesetzt werden:

1. **Kollisionsvermeidung:**



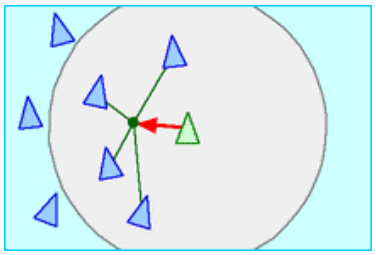
Damit die Instanzen nicht oder nicht komplett miteinander kollidieren wird ein Richtungsvektor ausgerechnet, der sich von den anderen Individuen in einem bestimmten Radius entfernt.
Um diesen Vektor auszurechnen, muss man über alle Individuen in diesem Radius iterieren und deren Richtungsvektor zu unsere Instanz zu einem neutralen Richtungsvektor hinzuaddieren. Nach der Iteration wird dieser durch die Anzahl der Individuen im Kreis geteilt.

2. **Flug ausrichten:**



Es wird ein weiterer Richtungsvektor ausgerechnet, der sich mit den Flugrichtungen der anderen Individuen ausrichtet.
Wir iterieren wieder durch alle Individuen in dem Radius und addieren den Geschwindigkeitsvektor aller Individuen auf einen neutralen Vektor und dividieren erneut durch die Anzahl der Individuen.

2. Zentrieren:



Um trotzdem immer nah an seinen Nachbarn zu fliegen bestimmt die letzte Regel einen Richtungsvektor, der zum Mittelpunkt der anderen Individuen in dem Radius zeigt.
Wieder wird ein neutraler Vektor mit den Positionen der einzelnen Individuen gefüllt und durch die Anzahl der Individuen dividiert.
Um diese Regeln anzuwenden, werden sie miteinander addiert und dem Geschwindigkeitsvektor einer Instanz zugewiesen.

2.2.3 Eine weitere Regel

Damit der Schwarm sich auf einen bestimmten Punkt zubewegt, gibt es eine weitere Regel, die einen Richtungsvektor zum Ziel ausrechnet und auf die Geschwindigkeit des Individuums hinzuaddiert wird.

Kugelformation

Wie der Name bereits verrät, formieren sich die einzelnen Individuen zu einer Kugel und schweben entlang der Oberfläche bis sie weitere Befehle erhalten.
Der folgende Algorithmus stellt sicher, dass eine Instanz flüssig zu der Kugeloberfläche interpoliert:

```
1  for(FSwarmInstance* Instance : Instances)
2  {
3    //Richtungsvektor von der Instanz zu dem Ziel
4    Vector DiffVector = Instance->Location - TargetLocation;
5
6    //Laenge des Richtungsvektors
7    float CurrentRadius = DiffVector.Size();
8
9    //Projektion der Position auf die Kugeloberflaeche
10   Vector PointOnSphere = TargetLocation
11             + DiffVector.GetNormalized() * SphereRadius;
12
13   //Projektion des Geschwindigkeitsvektors, addiert mit der
14   //momentan Position, auf die Kugeloverflaeche
15   Vector ProjectionVector = Instance->Location + Instance->Velocity;
16   ProjectionVector = TargetLocation
17            + (ProjectionVector - TargetLocation).GetNormalized()
18            * SphereRadius;
19
20   //Falls die momentane Position einen bestimmten Radius
21   //zur Oberflaecheerreicht, setzen wir die neue Position
22   //auf die Oberflaeche, da es sonst zu Oszillationen kommen kann
23   if ((Instance->Location - PointOnSphere).Size() < (SwarmSpeed * 0.1f))
24   {
25     Instance->Location = PointOnSphere;
26     Instance->Velocity = ProjectionVector - PointOnSphere;
27   }
28   else
29   {
30     //Richtungsvektor von der momentan Position
31     //zum Punkt auf der Kugeloberflaeche
32     Vector DiffToSphereOverlay = (PointOnSphere - Instance->Location);
33
34     //Berechnung der neuen Geschwindigkeit
35     Vector OnSphereVelocity = (ProjectionVector - PointOnSphere);
36     OnSphereVelocity.Normalize();
37
38     Instance->Velocity = (DiffToSphereOverlay.GetNormalized()
39                + OnSphereVelocity) * 0.5f * SwarmSpeed;
40   }
41
42   //Aktualisieren der neuen Position abhaengig von der Geschwindigkeit
43   Instance->Location += Instance->Velocity * DeltaTime;
44 }
```

2.3 Die Klasse *ASwarmOrigin*

Dieses Objekt leitet die Klasse APawn ab, und wird einem ***Controller*** zugewiesen. Außerdem wird dieser Klasse ein Ziel gegeben, zu dem der Schwarm hingezogen wird und dieses angreift, sobald es sich bewegt.

2.4 Das Blueprint *SwarmController*

Der SwarmController startet einen BehaviorTree, der festlegt, wie sich der Schwarm verhalten soll.

2.4.1 Behavior Tree

Das Verhalten des Schwarms ist sehr simpel gehalten. Sobald sich das Ziel bewegt, greift der Schwarm an. Andernfalls formieren sich die Instanzen zu einer Kugel, um das Ziel herum.
Der Behavior Tree sieht wie folgt aus:

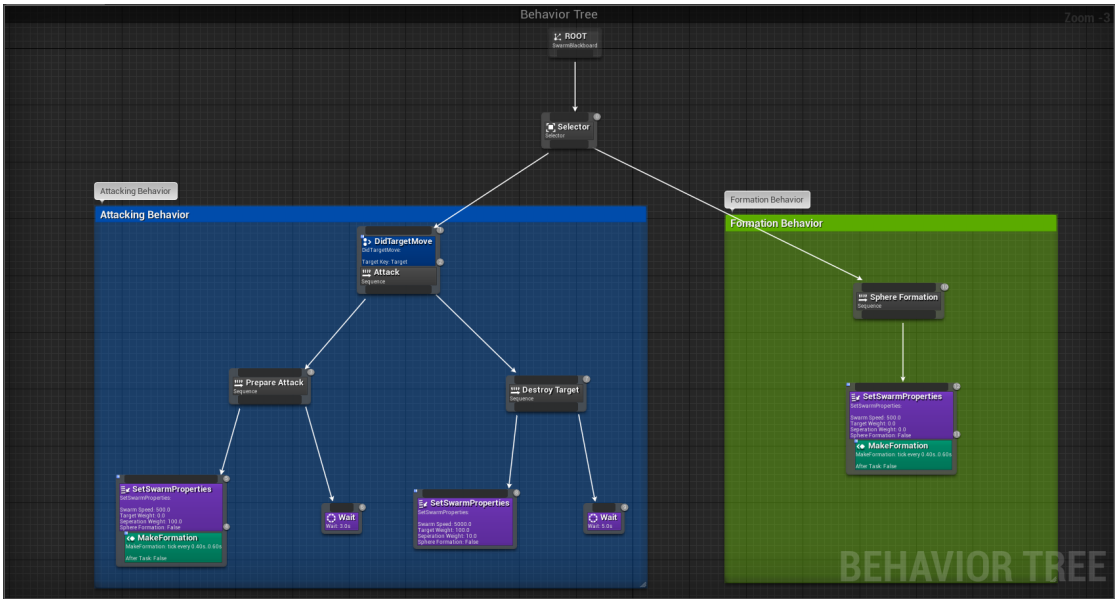


Abbildung 2.2. Behavior Tree des Schwarms

Wie man sehen kann werden lediglich einzelne Variablen neu gesetzt, um dem Schwarm andere Verhaltensweisen zu ermöglichen.

Literatur

https://www.red3d.com/cwr/boids/