HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Comment [FBI1]: Dokumentansicht ohne Kommentare:
-Klicken Sie in der Menüleiste auf „Überprüfen“->“Markup anzeigen“ Dort die Auswahl „Kommentare“ inaktiv setzen.

Comment [FBI2]: Ansicht aller Formatvorlagen:
Klicken Sie in der Menüleiste auf „Start“. Bei „Formatvorlagen ändern“ finden Sie rechts unterhalb einen kleinen Pfeil der nach rechts unten zeigt. Diesen anklicken.

Comment [FBI3]: Titel der Arbeit Kann im Attribut „Titel“ gepflegt werden. Zu erreichen über Datei-> Informationen. Rechts in der Spalte sehen Sie den Titel, den Sie editieren können.

English Title kann im Attribut „Betreff“ angepasst werden.

Autorname Kann im Attribut „Autor“ gepflegt werden.

Schwarm Intelligenz Demonstration

Swarm intelligence demonstration

Niklas Becht (MatrNr.957226), Johann Hahn(MatrNr.957311)

Dokumentation & Übersicht

Comment [FBI4]: Kann im Attribut „Manager“ gepflegt werden.
Zu erreichen über Datei-> Informationen. In der rechten Spalte ist das Attribut „Manager“

Betreuer: Prof. Dr. Christof Rezk-Salam

Ort, 31.8.2015 Deutschland Trier

Comment [FBI5]: Aktuelles Systemdatum

Vorwort

In dieser Arbeit wird das Verhalten einer einfachen Schwarm-Intelligenz zwei verfeindeter Gruppen simuliert.
Dem Verhalten der Schwarmintelligenz liegt eine Statemachine zu Grunde, welche sich über das Laden externer Lua-Script Dateien beliebig verändern lässt.

Comment [FBI6]: Das Inhaltsverzeichnis kann automatisch generiert werden:
-Rechter Mausklick ins Inhaltsverzeichnis
-Felder aktualisieren wählen
-Gesamtes Verzeichnis aktualisieren

Einträge werden erzeugt wenn die Formatvorlagen
-1 Überschrift 1
-1.1 Überschrift 2
-….
Verwendet werden

Inhaltsverzeichnis

Abbildungsverzeichnis

Comment [FBI7]: Das Abbildungsverzeichnis kann automatisch generiert werden:
-Rechter Mausklick ins Abbildungsverzeichnis
-Felder aktualisieren wählen
Gesamtes Verzeichnis aktualisieren

Einträge werden erzeugt wenn die Abbildungen beschriftet werden (siehe. Abbildung 2.1 Bezeichnung der Abbildung)

Einleitung

In der Natur lassen sich immer wieder beindruckende,
gleichförmige und elegante Bewegungen von Tierschwärmen beobachten.
Auch wenn keine feste Absprache zwischen den einzelnen Tieren statt findet
ergeben sie in der Masse oft einen kompakten Körper, was in der Natur verschiedenste Vorteile mit sich bringt.

Dieses Verhalten beruht auf einfachen Prinzipien (bzw. Entscheidungen), welche
die Individuen in Abhängigkeit der Umwelt auf sich anwenden.

Um dieses Verhalten zu simulieren wurden in dieser Arbeit
so genannte „Steering Behaviours“ und das Boid-System von Craig Reynolds implementiert.

Diese Modelle finden ihre Anwendung unter anderem in verschiedensten Filmen als auch in Videospielen von „Batman 2“ bis zu „König der Löwen“.

1 Das Projekt

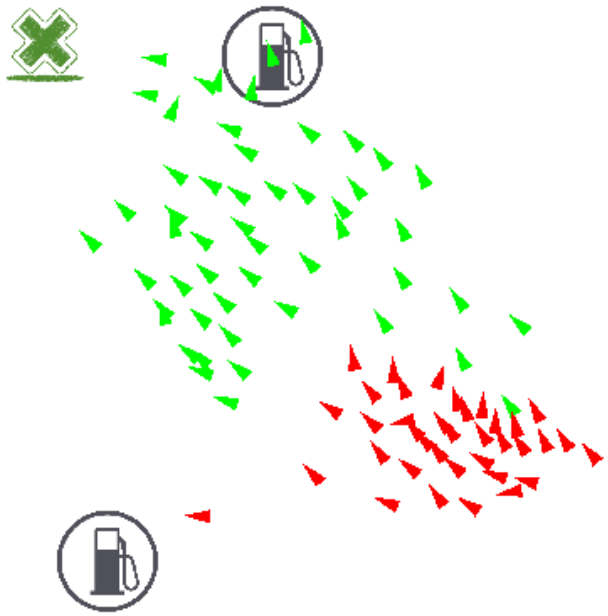

Abbildung 1 Screenshot laufendes Programm

Beim Starten des Programms sind zwei verfeindete Teams zu sehen, Team Rot und Team Grün.

Die Anzahl der Boids (Dreiecke) lässt sich mit dem Drücken der Taste „G“ zum Hinzufügen von Mitgliedern von Team Grün und mit dem Drücken der Taste „R“ zum Hinzufügen von Mitgliedern des Team Rots, beeinflussen.
Durch das Drücken der Taste „D“ werden zwei Kreise um jedes einzelne Boid angezeigt. Der rote, innere Kreis stellt dabei die bevorzugte Distanz zu seinem Teamkollegen im Schwarm dar.
Der blaue Kreis definiert dabei das Sichtfeld eines Boids (siehe Abbildung 2).

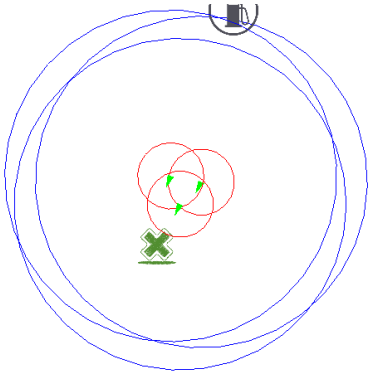

Abbildung 2 Programm beim gedrückter "D"-Taste

Die Boids (die einzelnen Dreiecke) verfügen über die Attribute „Leben“ und „Benzin“.
Das Ziel der grünen Boids stellt dabei das grüne Kreuz dar, welches bei Kontakt mit dem Boid verschwindet sowie diesen heilt und einen weiteren grünen Boid erzeugt.
Das Ziel der roten Boids ist es, das nächste grüne Boid möglichst lange zu berühren und dessen Lebens-Wert zu verringern.
Durch den Kontakt mit ihrer Tankstelle können die Boids ihr Benzin Attribut wieder auffüllen, um nicht mitten auf der Strecke liegen zu bleiben.
Um zum allgemeinen Setup zu kommen muss die Taste “S“ gedrückt werden.
Hier können einzelne States in Form von Luascript-Dateien für die zugrundeliegende State machine geladen und entfernt werden, als auch der Start States der einzelnen Teams gesetzt werden.

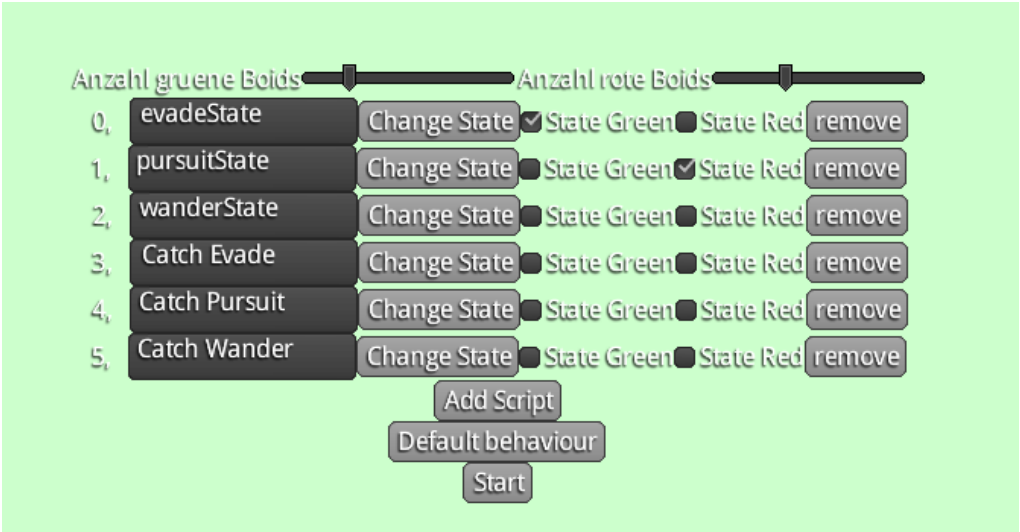

Abbildung 3 SetupScreen

2 Aufbau & Frameworks

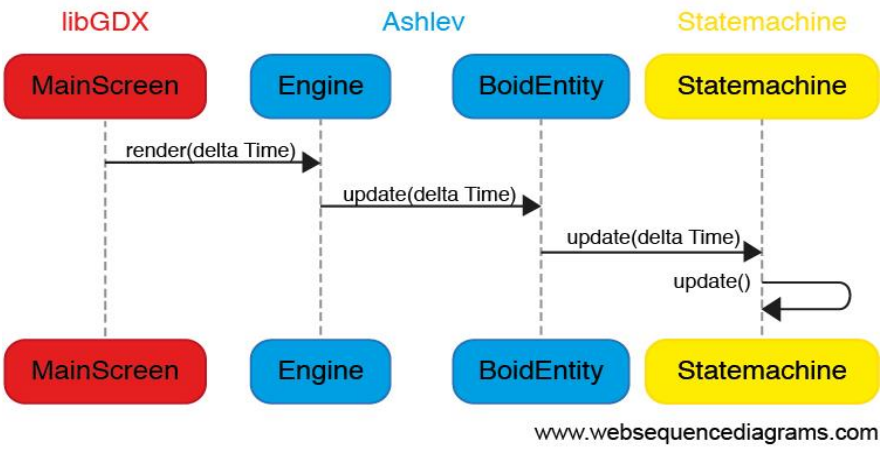

Abbildung 4 Ineinander greifen der Frameworks

Diese Arbeit wurde mit „libGDX“, einem java-game-development Framework umgesetzt.

2.1 Ashley Framework

Das Ashley Framework ist ein leichtgewichtiges Entity-Component-System.
Das Konzept ist dabei sehr einfach und besteht aus 3 Hauptkomponenten, diese sind:

- Die Components, welche Informationen speichern
- Die Entities, eine Art Container, welcher einzelne Components besitzt
- Die Systems, welche Aktionen auf die ausgewählt Entities ausführen

2.2 Ablauf

In der MainScreen Klasse wird eine Instanz des Ashley Framework instanziiert und mit Entities und Systems gefüllt.
Nun wird in regelmäßigen Zeitabständen (deltaTime) die Methode render(deltaTime) der MainScreen Klasse aufgerufen. In dieser Methode wird das Ashley Framework aktualisiert und angetrieben.
Diese wiederum führt die update(delatTime) Methode des Entities aus.
Jedes Entity besitzt seine eigene "Statemachine", die bei jedem Aufruf der update(delatTime) Methode das Verhalten des aktuellen States ausführt.

3 Steering Behaviors

Steering Behaviors sind eine einfache, aber effektive Möglichkeit realistische Bewegung von Objekten zu realisieren. Dabei werden keine komplexen Berechnungen durchgeführt, sondern mit Informationen über die Umgebung des Objektes Kräfte-Vektoren bestimmt, die das Objekt in die gewünschte Richtung lenken.

Somit können z.B. Charaktere in einem Spiel auf beliebige Situationen spontan und dynamisch reagieren.

Dieses Prinzip lässt sich auf verschiedenstes Verhalten anwenden wie, Weglaufen/Verfolgen von Gegner, Patrouillieren oder dem folgen eines Pfades, anwenden.

In dieser Abreit wurden verschiedene Steering Behaviors implementiert.

3.1 Wander-Behavior

Der Charakter wandert zufällig über den Bildschirm. Um zu sprunghaft und unrealistisch Bewegungen zu vermeiden, wird ein Kreis vor dem Charakter berechnet (siehe Abbildung 5). Ein Vektor wird in diesem Kreis zufällig um einen sehr kleinen Winkel rotiert. Dieser Vektor wird dann zum Bewegungsvektor des Charakters dazu addiert.

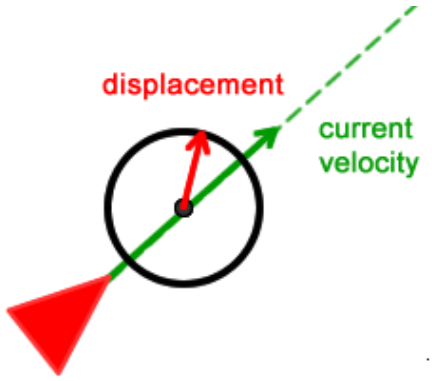

Abbildung 5 Der displacement Vektor

Dadurch entsteht bei jeder Berechnung nur eine kleine Abweichung, die die Bewegungsrichtung des Charakters nur gering ändert. Die dadurch entstandene Bewegung wirkt natürlich.

In dieser Arbeit wird das Wander-Behavior für Boids eingesetzt, die gerade kein Ziel haben und sonst leblos stehen bleiben würden. Sobald ein Gegner oder ein erreichbares Ziel im Sichtfeld des Boids auftaucht , ändert dieser sein Verhalten.

3.2 Seek- und Flee-Behavior

Die verhältnismäßig leicht zu implementierend Verhalten Seek und Flee steuern einen Boid auf dem direkt Weg auf ein Ziel zu. Als Kombination mit der schon vorhandenen Geschwindigkeit eines Boids entsteht ein Seek- oder Flee- Path der das Objekt zum Ziel führt.

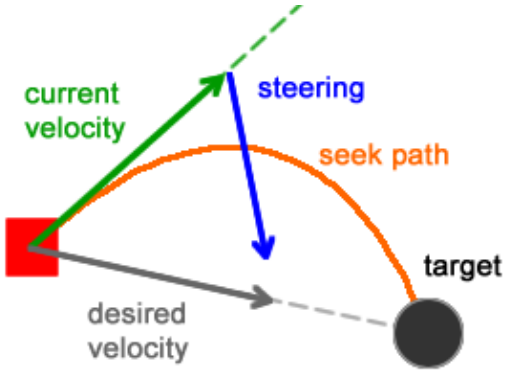

Abbildung 6 Das Seek-Behavior

Das Seek-Behavior wird für Boids benutzen, die auf dem schnellsten Weg ein Ziel erreichen wollen. In dieser Arbeit ist eines dieser Ziele die Tankstation .Diese verhalten lässt sich auch in Kombination mit anderen Steering Behaviors wie dem Pursuit-Behavior einsetzen.

Das Gegenstück zum Seek-Behavior stellt das Flee-Behavior da. Wird der Steering-Vektor der Abbildung umgedreht erhält man einen Pfad, der von dem Ziel wegführt.

Das Flee-Behavior wird in dieser Arbeit auch als Teil des Evade-Behavior genutzt

3.3 Arrival

Das Arrival-Behavior sorgt dafür das Charaktere, welche sich ihrem Ziel auf eine gewisse Distanz nähern, langsam abbremsen und an dem Ziel stehen bleiben.

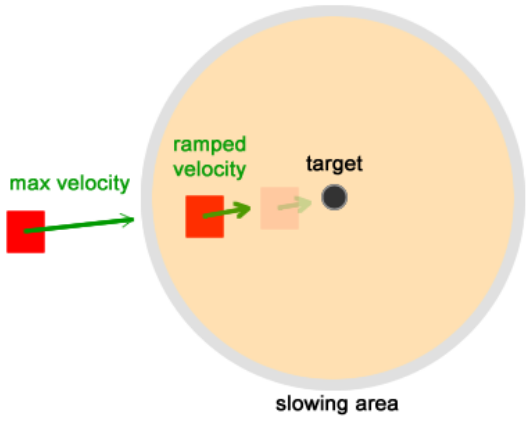

Abbildung 7 Arrival-Behavior

Auf Abbildung 7 ist ein „slowing Area“ zusehen, bei durchqueren dieser fängt der Boid an zu bremsen. Die Geschwindigkeit des Boids nimmt proportional zu der Distanz ab und bleibt im Ziel stehen.

Das Arrival-Beahvior wird in dieser Arbeit als Teil des Seek-Behaviors eingesetzt, um ein pendeln um Ziel zu verhindern.

3.4 Pursuit- und Evade-Behavior

Das Pursuit-Behavior ist eine Erweiterung des Seek-Behaviors. Um ein sich bewegendes Ziel realistisch und effektiv verfolgen zu können wird die zukünftige Position des Ziels mit Hilfe der Position und des Geschwindigkeits-Vektors des Ziels berechnet.

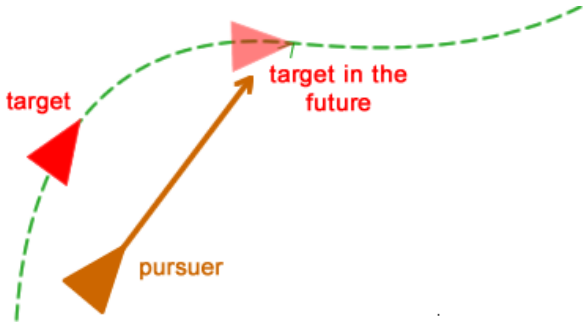

Abbildung 8 Pursuit-Behavior

Die ermittelte zukünftige Position wird dann als Ziel des Seek-Behavior angewendet.In dieser Arbeit wird dieses Verhalten als das Startverhalten des roten Schwarms gesetzt

Evade-Behavior ist das Gegenstück zum Pursuit-Behavior und somit eine Erweiterung des Flee-Behaviors. Nach dem gleichen Prinzip wird die zukünftige Position des Zieles berechnet. Auf welches dann das Flee-Behavior angewendet wird.

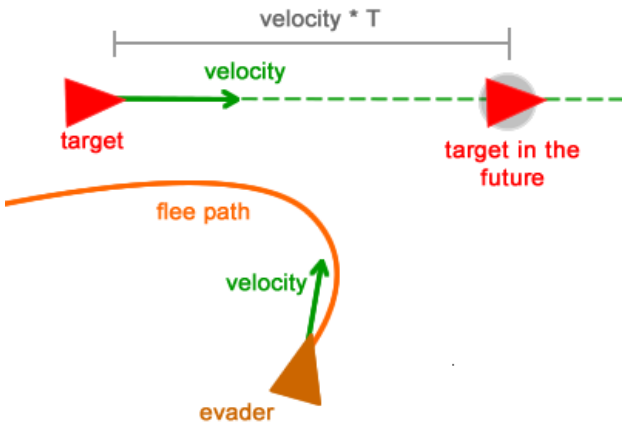

Abbildung 9 Evade-Behavior

4 Boid-System

4.1 Boids

Boids stellen eine Simulation einer Schwarmintelligenz da.
Hierbei werden die durch das Steering-Behaviours berechneten Bewegungs-Vektoren um drei Komponenten erweitert.

Damit die einzelnen Boids einen Schwarm bilden können, müssen jedes einzelne Boid vermeiden mit einem andren Boid aneinander zustoßen(Seperation),
sich möglichst nahe am Zentrum des Schwarms aufhalten(Cohesion) und deren Geschwindigkeit und Orientierung annehmen(Alignment).

4.2 Anziehung(Cohesion)

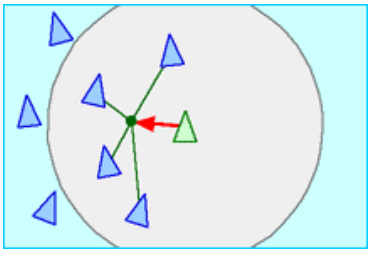

Abbildung 5 Anziehung (Cohesion)

Ziel der Anziehung ist es einen möglichst kompakten Schwarm zu bilden.
Dafür schaut jeder Boid welche Schwarmmitglieder in seinem Sichtfeld liegen (siehe Abbildung 5 & Abbildung 2) und berechnet das so genannte „Center of Mass“, also den Mittelpunkt der Schwarmmitglieder in seinem Sichtfeld.
Zu diesem Punkt wird nun ein Vektor berechnet und dieser dem im Steering Behavior berechneten Bewegungs-Vektor hinzugefügt.

4.3 Abgrenzung(Seperation)

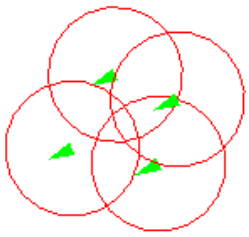

Abbildung 6 Bevorzugte Distanz zu Schwarmmitglied

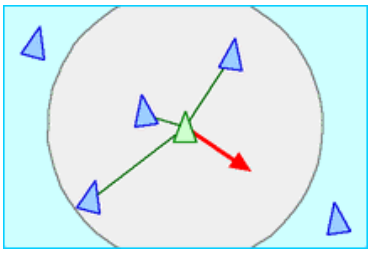

Abbildung 7 Abbgrenzung (Seperation)

Bei der Abgrenzung wird dafür gesorgt, dass die Boids durch den berechneten Anziehungsvektor nicht kollidieren. Sobald ein Schwarmmitglied zu nahe kommt, berechnet das jeweilige Boid für jedes zu nahe Schwarmmitglied einen gewichteten Distanz Vektor.
Die Gewichtung in dieser Arbeit wird berechnet indem die bevorzugte Distanz durch die tatsächliche Distanz dividiert wird.
Die daraus berechneten Vektoren werden gemittelt und ergeben einen Abgrenzung. Dieser wird dem Bewegungs-Vektor hinzugefügt.

4.4 Ausrichtung(Alignment)

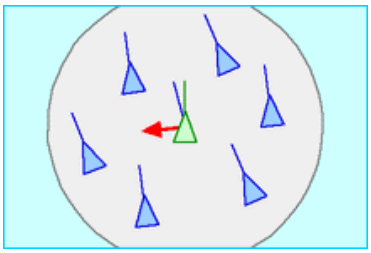

Abbildung 8 Aussrichtung (Alignment)

Die Ausrichtung sorgt dafür, dass die Ausrichtung und die Geschwindigkeit der Boids aneinander Angeglichen werden. Dabei betrachtet das einzelne Boid den Geschwindigkeits-Vektor aller Schwarmmitglieder in seinem Sichtfeld, mittelt diese und fügt den so entstehenden Ausrichtungs-Vektor seinem Bewegungs-Vektor hinzu. Dadurch gleicht sich die Ausrichtung und Geschwindigkeit der Schwarmmitglieder immer weiter an.

5 Ashley-Systeme:

Wie bereits im Kapitel 3.1 erklärt, werden in unserer Simulationen Daten in Components gespeichert, die den Entities hinzugefügt werden. Diese Daten werden mithilfe folgender Systeme bearbeitet.

In den Systemen wird der Hauptteil der Berechnungen des Programms getätigt.

In dieser Arbeit wurden drei verschiedene Systems implementiert.

5.1 Render-System

Das RenderSystem erzeugt den optischen Teil des Programmes, indem es die Boid-Objekte, den Hintergrund und die "Point of Interest Objekte" zeichnet. Die Engine des Ashley-Frameworks ruft für jeden Frame die update-Methode des Systems auf, die den Frame zeichnet.
Dafür wurde in dieser Arbeit eine spezielle RenderComponent implementier, die Informationen über Höhe, Breite und Textur des Entities enthält. Entities ohne Textur sind Boid-Entities und werden als Dreiecke dargestellt.

Nur Entities mit einer RenderComponent werden von dem RenderSystem betrachtet.
Ebenso stellt das RenderSystem die Möglichkeit, mit der Taste „**V**" die Boid-Vektoren zu zeichnen, zur Verfügung. Desweiteren wird mit dem drücken der Taste „**D**" das Sichtfeld und die optimale Distanz der Boids gezeichnet .

5.2 Movement-System

Der Kernteil der Simulation wird im MovementSystem berechnet , dass jeden Frame die Boid-Vektoren und die Steering Behavior-Vektoren berechnet. Die Update-Methode des Systems besteht dabei aus drei folgenden Methoden:

updateVectors():

Berechnet die Boid-Vektoren und Steering Behaviour-Vektoren für den aktuellen Frame. Diese werden in den einzelnen Components eines Entities gespeichert. Dabei werden nur Vektoren berechnet, für die eine entsprechende Component vorhanden ist.
Wenn Beispielsweise ein Entity eine SeekComponent enthält aber keine FleeComponent, dann wird für dieses Entity ein Seek-Vektor aber kein Flee-Vektor berechnet.

Somit lässt sich das aktuelle Verhalten eines Entities beliebig einstellen, indem es um Components erweitert oder erleichtert wird.
Die Vektoren können beliebig gewichtet werden, um gewünschte Effekte zu erzielen, wie es bei berechneten Abgrenzungs-Vektor eines Entities der Fall ist.

setPosition():

Diese Methode berechnet die neue Position aller Boids. Dafür werden alle in der Methode updateVectors() berechnete Vektoren eines Boids zu seinem Velocity-Vector addiert.
Die neue Position eines Boids ergibt sich wie folgt:

Alte Position + Velocity-Vektor= new Postion

Die in der updateVectors() Methode berechneten Vektoren repräsentieren also als eine Kraft, die den Boid in eine neue Richtung lenkt. Die“ alte“ Kraft wird durch obige Formel auch miteinbezogen, was zu einer flüssigen und gleichmäßigen Bewegungsablauf führt.
Der Velocity-Vektor wird hier bei einer maximal Geschwindigkeit abgeschnitten, da die Geschwindigkeit durch die Addition sonst unbegrenzt steigen würde.
Die Boid-Entities werden nach dem setzen der neuen Position im Bildschirm gebunden, indem die Boids bei Austritt auf der anderen Seite des Bildschirms wieder eintretten.

Entity.update():

Ruft die update-Methode der State-Machine jedes Entities auf.

5.3 Ressource-System

Um zusätzliche Dynamik in die Simulation zu bringen, wurde jedes Boid um die Attribute *fuel* und *health* erweitert. Diese werden in einer RessourceComponent gespeichert.
Das Ressource-System verwaltet diese Attribute.
Jede Zeiteinheit (einstellbar als Attribut in der RessourceComponent) verlieren die Boids Sprit, der *fuel*-Wert wird um einen festen Wert zurückgesetzt.
Befindet sich ein Boid an einer Tankstation, dann wird *fuel* im jeden Frame wieder hochgerechnet.
Ein Boid ohne *fuel* (<= *0*) kann sich noch bewegen, jedoch wird die Geschwindigkeit stark reduziert, dadurch werden den Boids noch eine Chance eine Tankstation zu erreichen ermöglicht.
Falls ein “Verfolger“-Boid einen fliehenden Boid berührt, verliert der fliehende Boid einen Teil seines *health*-Wert. Fällt der *health*-Wert <= *0*, ist der Boid „gestorben“ und wird aus der Engine entfernt.

6 Lua

Lua ist eine Skriptsprache mit einer C ähnlichen Syntax. Die oft für Spiele und andere Projekte verwendet wird, da Lua schnell und mächtig ist.

Diese Arbeit ermöglich das laden externer Lua-Dateien, um die Statemachine eines Boids zu erweitern .Diese Schnittstelle wurde mittels des **LuaJ**Interpreters von James Roseborough implementiert.

Jede Boid-Entity in unserer Simulation besitzt eine eigene Statemachine, die das Verhalten eines Boids in einer beliebigen Situation steuert. Diese beinhaltet bereits einige in Java implementierten States wie Wander, Evade, Pursue oder. Diese Sates bilden die Default State Machine dieser Arbeit.Um die Simulation von außen umprogrammieren zu können wird Lua verwendet. Für die Einbindung der Lua-Skripte in Java benutzen wir den **LuaJ**Interpreter.

6.1 LuaScript und LuaState

Die Klasse LuaScript kapselt wichtige Funktionen der LuaJ Bibliothek und stellt Methoden zum laden und ausführen von Lua-Script-Dateien zur Verfügung.

Mit einem Script lässt sich in unserem Programm ein neuer State definieren, der auch andere States aufrufen kann. Somit erhält man die Möglichkeit eine neue Statemachine aufzubauen und das Verhalten der Boids komplett neue zu definieren.

Um beliebige neue States mit Lua definieren zu können , wird die Klasse **LuaState** genutzt. Diese besitzt einen LuaScript-Attribut, einen Namen und wie alle anderen States die Enter-, Update- und Exit-Methoden:

```
@Override
publicvoid enter(BoidEntity entity) {
	script.executeFunction("enter", entity);
}

@Override
publicvoid update(BoidEntity entity) {

	script.executeFunction("update", entity);
}

@Override
publicvoid exit(BoidEntity entity) {
	script.executeFunction("exit", entity);
}
```

In den Methoden lässt sich erkennen, dass die ExecuteFunction-Methode des Scripts aufrufen wird.Diese ist in der LuaScript Klasse definiert und dient dazu eine im Skript definierte Methode aufzurufen. Als Argumente nimmt die Methode ExecuteFunction den Namen der auszuführenden Skript-Methode und eine beliebige Anzahl an Java-Objekten auf.

Um einen neuen State in Lua zu definieren muss der Skript also die drei Methoden *enter(entity), update(entity), exit(entity)* implementierten. Das entity-Argument ist ein BoidEntity, welches verschiedenste Methoden zur Verfügung stellt um mit einer Lua-Datei dasVerhalten der Boids zu verändern.

Diese Methoden der Klasse BoidEntity können aus einer Lua-Datei aufgerufen werden:

```
public void switchTeams()

public Entity searchTarget()

public void setPointOfInterest(PointOfInterestEntity e)

public void changeStateByName(String state)

public void setTexture(String path)

public void setRealtivPostion(float x,float y)

public boolean gotHit()

public void setHit(boolean b)

public void addComponent(String name)

public void removeComponent(String name)

public boolean checkFuel()

public PointOfInterestEntity getGlobalTarget(Engine engine)

public boolean setTarget(Entity target, String action)

public boolean gotComponent(String name)

public void resetRessources()
```

Besonders der Zugriff auf die Komponenten und das Entfernen und Hinzufügen dieser ermöglicht es komplett neue Verhaltensweisen zu definieren. Desweiteren ist der Name eines LuaStates sehr wichtig da er als „Identifier“ genutzt wird.

```
public LuaState(LuaScript script){
this.script = script;
        LuaValue nameValue = script.callForReturn("setName", name);
name = nameValue.tojstring();
}
```

Der Name wird mit der Methode callForReturn() aus dem Script gelesen. Diese ruft die Lua-Funktionen auf, welche den Namen des Scripts als LuaValue Objekt zurück geben sollte.
Aus diesem LuaValue Objekt wird der Wert in ein Java Attribut umgewandelt.

Der Skript muss also den Namen des States definieren und eine Methode setName() implementieren, die den Namen des Scripts zurückgibt. Dies wird benötigt um verschiedene LuaStates voneinander unterscheiden zu können.

Die Klasse **ScriptHolder** verwaltet alle States, beim Laden eines States in Form eines JavaObjects oder eines Luascripts , werden diese in einer ArrayList gespeichert.

6.2 Statemaschine in Lua-Script Form.

Im Verzeichnes **assets/data/scripts** der Arbeit befindet sich eine auf Luascripts basierte State Machine „Fangen“ , die drei States „Catch Wander“,“Catch Evade“, „Catch Pursuit“ beinhaltet States welche Default Statemachine sehr ähnlich sind.

Der Unterschied besteht in der Interaktion des Boids, bei Berührung werden grüne Boids „gefangen“ und wechseln das Team.

```
1
2 name = "Catch Evade"
3
4 function init()
5 end
6
7 function enter(entity)
8 end
9
10 -- UPDATE --
11 function update(entity)
12
13     if not entity:setTarget(entity:searchTarget(), name) then
14         entity:changeStateByName("Catch Wander")
15     end
16
17     if entity:gotHit() then
18         entity:switchTeams()
19         entity:changeStateByName("Catch Pursuit")
20         entity:resetRessources()
21     end
22
23     if entity:checkFuel() then
24         entity:removeComponent("Seek")
25         entity:changeStateByName("Catch Wander")
26     end
27
28
29 end
30
31 function exit(entity)
32     entity:removeComponent(name)
33 end
34
35
36 function setName(inputName)
37     return name
38 end
39
```

Abbildung 9 Catch Evade in LuaScript

Literatur

Comment [FBI8]: Formatvorlage Überschrift mit Eintrag ins Inhaltsverzeichnis

Comment [FBI9]: Kopieren Sie die Datei ZSEOF_Abschlussarbeit.xsl in den Ordner „..MicrosoftOffice\Office14\BibliographyStyle“. Damit steht Ihnen nach Neustart von Word unter „Verweise“->“Formatvorlage“ die Bibliographievorlage „Zweispaltig EOF“ zur Verfügung. Damit formatieren Sie eine zweispaltige Anzeige der Literaturangaben.

Boids by Craig Reynolds:
http://www.red3d.com/cwr/boids/
Ashley Framework:
https://github.com/libgdx/ashley/wiki/Getting-started-with-Ashley
LibGDX:
https://github.com/libgdx/libgdx/wiki
Stearing Behaviour:
http://gamedevelopment.tutsplus.com/series/understanding-steering-behaviors--gamedev-12732
Lua/LuaJ:
http://www.lua.org/manual/5.3/
http://www.luaj.org/luaj/3.0/README.html

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt.

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Name der Mitverfasser: Niklas Becht, Johan Hahn.....
.....

31.08.2015
Datum

Unterschrift der Kandidatin / des Kandidaten