

Steering Behavior Dokumentation

Hannes Höttinger - 1510585004

FH Technikum Wien, Game Engineering und Simulation, Wien, AUT

Abstract — Dieses Dokument dient als Dokumentation für die implementierten Steering Behaviors und als eine kurze Beschreibung der Programmausführung.

1 Aufgabenstellung

Folgende Funktionalitäten müssen implementiert werden:

Implementierung der im Unterricht vorgestellten Steering Behaviors

1. Ray Casting mit Center Ray + Whiskers
2. relevante Vektoren sichtbar darstellen
3. Trajektorie der bewegten Objekte visualisieren
4. alle Entitäten (Ziel, Hindernis, etc.) anzeigen

Gruppenbewegung

1. 4 NPCs in Wedge Formation
2. Formation folgt Ankerpunkt, der einem fixen Pfad folgt.
3. Priority Blending + Moderation
4. mehrere Rechteckige Hindernisse
5. 3 gegnerische Agenten die mit dem Pfad in Konflikt stehen

NICE-TO-HAVE

1. Implementierung des Beispiels mit Bitmap Grafiken
2. Realisierung der Steering KI in einer DLL

2 Umsetzung

Der Source Code wurde unterteilt in eine statische Library (*Steering_Lib.lib*) für das Unit Testing, welche die Behavior Logik beinhaltet und in ein Hauptprojekt für die Implementierung der Main Loop mit der Visualisierung in SFML.

Aufbau in Google Test Solution:

- GoogleTest (Main files für GoogleTest)
- Movement_Agent (Main Funktion für Steering)
- Steering_Lib (Steering Library)
- unitTest_Steering (Google Unit Testcases)

2.1 Steuerung

Die X-Wing Fighter folgen den Ankerpunkten der Wedgeformation, welche dem hellgrünen Pfad folgt. Lambda-Fähren agieren als Gegner die über die Map patrouillieren. Der Sternenzerstörer dient als Zielobjekt und erhält bei betätigen einer Taste eine Geschwindigkeit in x-Richtung. Der Millenium Falke (Behavior Agent), dient als Vorführgent für die implementierten Behaviors und wird erst bei betätigen einer Taste initialisiert. Die Objekte besitzen weiters folgende Behaviors:

- Obstacle Avoidance
- Collision Avoidance
- Separation

Eingabe	Steuerelement	
LMT	Ziel auf Mausposition setzen	→ Sternenzerstörer steuern
[F]	Flee	→ Millenium Falke initialisieren
[S]	Seek	→ Millenium Falke initialisieren
[A]	Arrive	→ Millenium Falke initialisieren
[W]	Wander	→ Millenium Falke initialisieren
[P]	Pursue	→ Millenium Falke initialisieren
[E]	Evade	→ Millenium Falke initialisieren
[B]	Follow Path	→ Millenium Falke initialisieren

Tabelle 1: Steuerelemente des Programms

2.2 Grafische Ausgabe und Update

Für die grafische Ausgabe mit den zur Verfügung gestellten Sprites wurde *SFML* (*Simple and Fast Multimedia Library* (<http://www.sfml-dev.org/>)) verwendet. In der Klasse Sprites werden die notwendigen Texturen und Spritenamen instanziiert und werden bei Programmausführung eingelesen.

In der Update Funktion werden die entsprechenden Behaviors aufgerufen und mittels Priority Blending ausgeführt. Die Sprites werden dementsprechend auf

die aktuelle Position mit der korrekten Orientierung in jedem Frame gesetzt. Weiters werden noch Hilfsvektoren und die entsprechenden Pfade in die Gamemap eingezeichnet. Die Obstacles werden auch mit Texturen aufgerufen und besitzen eine Bounding Box für das Obstacle Avoidance Behavior.

Listing 1: Google Unit Test Beispiel; Shadow Logik

```
// group dynamic
kinematic.update(wedge.anker, wedge.getTarget(), agent, ←
    obstacle, dt.asSeconds());
// agents
kinematic.update(agent[i], wedge.GetTargetSlot(i), agent, ←
    obstacle, dt.asSeconds());
```

In Abb. 1 sieht man ein Beispiel der Ausgabe mit eingefügten Texturen. Die Asteroiden beschreiben die eingefügten Obstacles. Der zusätzliche hellblaue Pfad dient zur weiteren Visualisierung von Follow Path mit dem Millenium Falke.

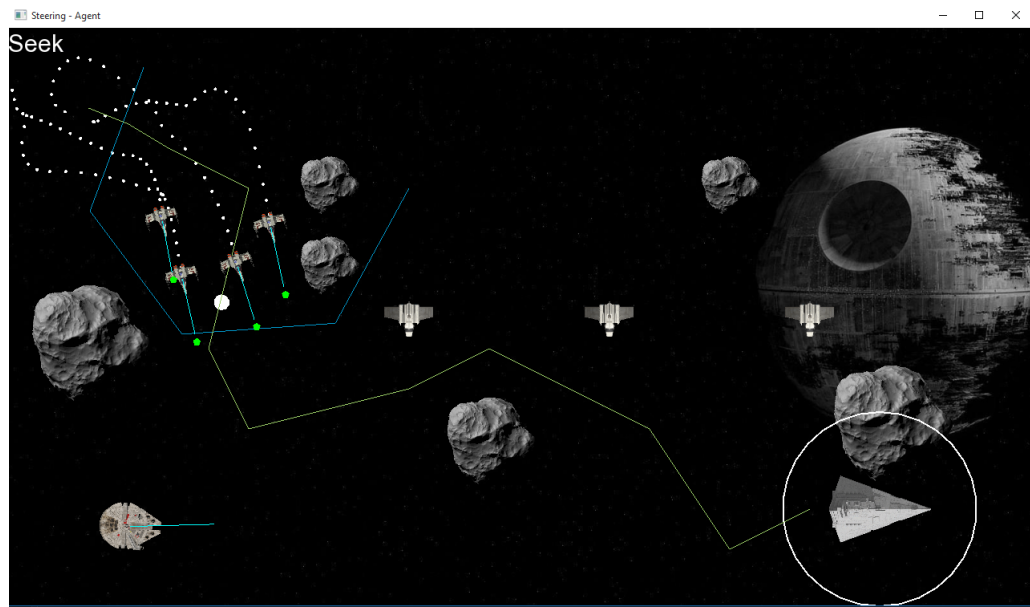
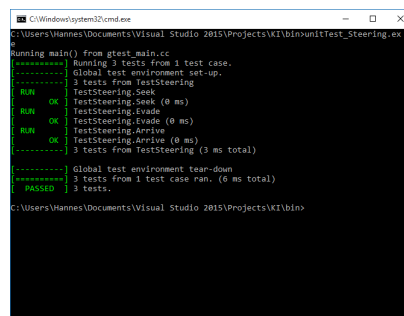


Abbildung 1: Steering Behavior visualisiert mit SFML

3 Google Unit Testing

Google Unit Testing ist verfügbar von <https://github.com/google/googletest>. Der Source Code wurde in eine statische Library kompiliert und ein neues Projekt für den Test angelegt. Um die KI im Google Framework zu testen, wurde der notwendige Code ebenfalls in eine statische Library kompiliert und in das Projekt, sowie das Hauptprojekt, eingebunden. Das Projekt *unitTest_Steering* hat nun Zugriff auf die verwendeten Klassen und Objekte. Das Hauptprogramm kann in diesem Testszenario weiterhin ausgeführt werden.

Google Unit Test für drei ausgewählte Tests:



```
C:\Windows\system32\cmd.exe
C:\Users\Hannes\Documents\Visual Studio 2015\Projects\KI\bin\unitTest_Steering.exe
Running main() from gtest_main.cc
***** Running 3 tests from 1 test case.
***** Global test environment set-up.
***** 3 tests from TestSteering
TestSteering.Seek
RUN OK TestSteering.Seek (0 ms)
TestSteering.Evade
RUN OK TestSteering.Evade (0 ms)
TestSteering.Arrive
RUN OK TestSteering.Arrive (0 ms)
***** 3 tests from TestSteering (3 ms total)
***** Global test environment tear-down
***** 3 tests from 1 test case ran. (6 ms total)
PASSED 3 tests.
C:\Users\Hannes\Documents\Visual Studio 2015\Projects\KI\bin>
```

Abbildung 2: Google Unit Test Ergebnis → drei ausgewählte Behaviors

4 Programmausführung

In dem Abgabeordner im Verzeichnis *bin* befindet sich ein .EXE Files (x64), **Movement_Agent.exe** das Hauptprogramm, welches die gesamte Funktionalität beinhaltet.

Ein zweites .EXE File **unitTest_Steering.exe** im Verzeichnis *test* führt die Unit Tests aus. Das Programm sollte mittels der Command Line ausgeführt werden:

unitTest_Steering.exe liefert anschließend die Ergebnisse der Tests in der Konsole (siehe Beispiel Abb. 2).

Die Sprites befinden sich in dem Unterverzeichnis *Sprites*. Die notwendigen DLLs für SFML und die Library für den Google Test sind inkludiert. Die Solutions sind mit relativen Pfaden angegeben, somit sollte das Programm ohne weitere Einstellung ausgeführt bzw. neu kompiliert werden können.

5 NICE-TO-HAVE

Folgende Punkte wurden implementiert:

1. Implementierung mit Bitmap Grafiken.
2. Steering KI in Library realisiert.