Informatik
Hauptcampus

HOCHSCHULE TRIER

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

Kommentiert [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.

Kommentiert [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.

Implementierung einer sich selbst verteidigenden Utility-based Game AI

Tobias Forve

963491

Dokumentation

Betreuer: Prof. Dr. Christof Rezk-Salama

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

Ort, 14.8.2019

Kommentiert [FBI5]: Aktuelles Systemdatum

Inhaltsverzeichnis

Kommentiert [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

Kommentiert [tf7R6]:

1 Einleitung und Motivation 3

2 Allgemein 4
2.1 Was ist Utility AI? 4
2.2 Berechnung der Score allgemein 4
2.3 Verwendete Programme und Assets 5
2.4 Steuerung 5

3 Umsetzung 6
3.1 Umsetzung in Code 6
3.1.1 AI Behaviors 6

4 Fazit 8

1 Einleitung und Motivation

Im Rahmen der Veranstaltung Künstliche Intelligenz für Spiele, sollte eine Spiele KI entwickelt werden. Bei meiner Recherche nach einer geeigneten Methode, stieß ich auf den Ansatz der „Utility-based Game AI“. Es faszinierte mich, dass trotz des doch sehr simplen Grundkonzepts, nichts an der Vielfallt und den Möglichkeiten verloren ging, eine weitreichende Spiele KI zu entwickeln. Weiter motivierte mich, dass es eine Alternative zu typischen Konzepten zur Erstellung einer Spiele KI darstellte. So zum Beispiel in der Unreal Engine 4[1] eingebaut *Behavior Trees*, an denen seit einigen Jahren kaum noch Veränderung oder Verbesserungen stattfanden.

Spiele wie Killzone 2[2], welches zu einer der besten Game AIs zählt haben bereits vorgemacht. Auch die AI von F.E.A.R.[3], Civilization[4] und auch die Bots von Quake[5] greifen auf ein ähnliches Utility System zurück.

[1] Unreal Engine 4 – 2005 von Epic Games
[2] Killzone 2 – 2009 von Guerilla Games
[3] F.E.A.R. – 2005 von Monolith Productions
[4] Civilization – 1991 von MicroPose, Activision etc.
[5] Quake – 1996 von id Software

2 Allgemein

2.1 Was ist Utility AI?

Utility bedeutet übersetzt Nützlichkeit und genau danach wird bewertet. Wie viel Nutzen hat eine Aktion für die KI und gibt es andere Aktionen von der sie mehr Nutzen hätte.

Dabei ist der Kern des Systems derart einfach. Man gibt einer KI eine Liste von möglichen auszuführbaren Aufgaben (*Task*), so zum Beispiel, Angreifen, Fliehen, nach einem Gegner suchen etc. Diese Aufgaben werden mit einem Wert (hier weiter *score* oder *weight* bezeichnet) zwischen 0.0 und 1.0 versehen, wobei ein höherer *score* gleich besser bedeutet und 0.0 heißt, dass die Aufgabe nicht in Betracht gezogen wird. Daraufhin werden sämtliche *scores* betrachtet und die Aufgabe mit dem höchsten *score* wird ausgeführt. Wie die Werte berechnet werden, kann von sehr einfachen Umständen abhängen, zum Beispiel: gibt es einen Gegner in der Nähe der angegriffen werden kann? Wenn ja – *score* für den *attack state = 1.0*. Oder aber es können sehr komplexe Variablen einfließen, Distanz zum Gegner, Anzahl der Gegner, Stärke des Gegners etc. Weiter können die Aufgaben auch weitere Gewichtungen erhalten. So dass selbst bei gleichem *score* eine Aufgabe stärker gewichtet wird, als eine andere und darum ausgeführt wird.

2.2 Berechnung der Score allgemein

Um eine gewisse Varianz und eine somit mehr sinnvolle Berechnung der einzelnen scores zu erhalten, greift man auf Funktionen zurück. Da die Nützlichkeit eines Tasks oft nicht linear steigt, sondern bedingt durch verschiedene Faktoren.
Während beispielsweise der Verlust der ersten 10% 20% 30% der Lebensenergie des Players noch keine große Bedrohung für ihn darstellt, nimmt die Bedrohung zu desto großer dieser Prozentsatz wird und kommt er in einen Fehlenden Bereichen von 70% 80% oder gar 90% ist die Bedrohung der Art hoch, dass das suchen nach Lebensenergie eine enorm hohe Nützlichkeit aufweist. Wie in diesem Projekt die einzelnen scores exakt berechnet werden folgt in Kapitel 3.1.1.

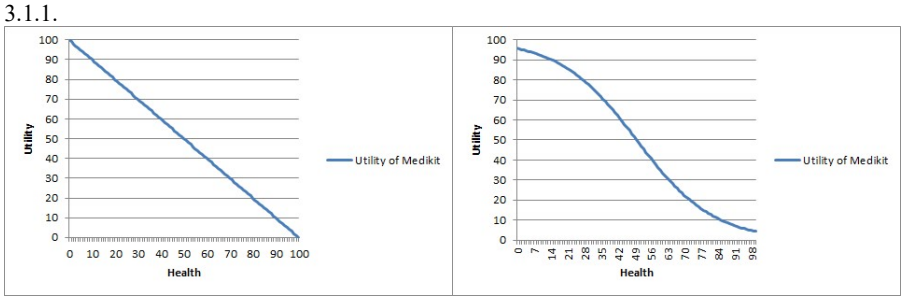

Abbildung 1: Verdeutlichung von sinnvollem Wachstum der Utility beim Verlust von Lebensenergie.

2.3 Verwendete Programme und Assets

Die Applikation wurde in Unity 2018.03f2[6] umgesetzt. Für die Darstellung wurden verschiedene Assets aus dem Unity Assetstore verwendet. Effekte, Playermodel sowie Level vom Realistic Effects Pack 4[7] sowie Gegnermodel und Animation vom Fantasy Monster – Skeleton Paket[8].

2.4 Steuerung

Lediglich die Kamera lässt sich durch Drücken der linken Maustaste drehen, sowie mit dem Mausrad hinein oder heraus zoomen.

2.5 UI

Das UI ist recht simple gehalten und soll veranschaulichen, was die KI gerade tut. So existiert im unteren Bild ein Slider, welcher anzeigt in welchem Task sich die KI befindet. In der oberen linken Ecke werden die aktuellen scores der einzelnen AI Behaviors dargestellt. Dies ist besonders hilfreich, um nachzuvollziehen, wieso die KI sich in einem Task befindet. Ein Killcount oben rechts existiert auch.

[6] Unity Engine – 2005 von Unity Technologies
[7] https://assetstore.unity.com/packages/vfx/particles/spells/realistic-effects-pack-4-85675 , letzter Aufruf 13.08.2019
[8] https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy-monster-skeleton-35635 , letzter Aufruf 13.08.2019

3 Umsetzung

Relevant sind vor Allem, der im Projekt existierende `GameManager`, welcher neben dem UI, vorwiegend das Spawn System für Gegnerwellen, wie Heil- und Manatränke verwaltet. Die `World` welche sämtlichen Levelassets inne hat, wodurch sich das Navmesh berechnet und dadurch die KI sich bewegen kann. Und der `Player`, welcher die eigentliche Spiele KI in Form eines Avatars darstellt.

Im nachfolgenden Kapitel wird weiter nur auf die wichtigsten Aspekte der KI eingegangen. Die Welt, Kamera, Licht, UI, sowie Spawn Systeme u.Ä. werden nicht tiefer erläutert.

3.1 Umsetzung in Code

Neben den Standard Komponenten, wie einem Animator, Rigidbody, Nav Mesh Agent und Collider besitzt der Player sechs weitere C# Scripts.

Der `HeroAI_Controller.cs` übernimmt die Aufgabe, in einem gewählten Radius Gegnerobjekte wahrzunehmen und den sich am nächsten zu ihm befindlichen in den Variablen `targetEnemy`, sowie `targetLookAt` zu speichern. Weiter wird in `float maxScore` der höchste score hinterlegt. Der `HeroAI_Controller` ist es der den Player von Enemy hauptsächlich unterscheidet.

Das `Character.cs` Script besitzen sowohl der Player also auch die Gegner *(EnemySkeleton)*. Es regelt unter anderem die `health`, das `mana` und die `runSpeed` des Characters. Zusätzlich verfügt es über ein `public Dictionary<string, List<Character>> characterByType`, welches dazu verwendet wird, das KI Verhalten *(AI Behavior)* auf einen Character Typ nach string zu lenken.
Weiter beinhaltet es eine `public List<WeightedDirection> desiredWeights` Liste, welche sämtliche, durch die einzelnen AI Behaviors errechneten scores und directions beinhaltet. In der `Update()` Methode, wird diese Liste jedes Frame neu erstellt und durch eine `BroadcastMessage`, an alle AI Behaviors, neu bevölkert, woraufhin mit einer `foreach` Schleife durch die Liste gelaufen wird um die Richtung zu errechnen in die der Character sich bewegen soll.
Die Methode `MoveTo()` bewegt den Character in die entsprechende Richtung. In der `Hit()` Methode wird nach einem Treffer die neue `health` berechnet, sowie für die Lebensanzeige der neue wert berechnet. `RestoreHealth()` und `RestoreMana()` werden aufgerufen, wenn der Character über ein Healthpotion oder Manapotion läuft.

3.1.1 AI Behaviors

Der Aufbau sämtlicher AI Behaviors ist sehr ähnlich. Sie alle besitzen eine Methode `void DoAIBehavior()` und private `void CalculateWeight()`. Weiter besitzen sie alle einen `string charType`, welcher angibt, welchen Character das Behavior betreffen soll, ein `float weight`, welches später zurückgegeben wird sowie ein `float weigtCalculated`, welches durch die `CalculateWeight()` Methode frame by frame neu berechnet wird. Der Boolean `veto`, kann aktiviert werden, um das `weight` dauerhaft auf *0.0f* zu setzen und somit das Verhalten auszuschließen.

Die `DoAIBehavior()` Methode wird durch den oben erwähnten Broadcast des Characters aufgerufen und dient am Ende dazu, ein `WeightedDirection` zu erzeugen und dies an die `desiredWeights` Liste zu übergeben.

`AI_EvadeEnemy.cs`: dient dazu vor dem Gegner zu fliehen, sollte dieser sich in einer definierten Nähe (`rangeOfCare`) des Players aufhalten. Das weight steigt hierbei exponentiell. Im Graph mit *Threatlevel* dargestellt.
Formel: ***10 / Distanz zum Gegner^2***

`AI_SeekHealth.cs`: entscheidet, wie relevant es ist nach einem Healthpotion zu suchen. Die Steigung der Utility hier, lehnt sich an eine Sigmoid Kuve. Im Graphen mit *Health* dargestellt.
Formel: ***0.5* sin (PI / max Health * (current Health + max Health/2)) + 0.5***

`AI_SeekMana.cs`: entscheidt, wie relevant es ist nach einem Manapotion zu suchen. Im Graphen mit *Mana* dargestellt.
Formel: ***2^(current Mana * -0.05)***

`AI_ShootEnemy.cs`: entscheidet auf welchen Gegner geschossen werden soll. Hierbei steigt die Nützlichkeit mit jedem Gegner in Reichweite einfach um 0.2. Im Graphen mit *Enemy Strength* dargestellt (wobei hier nur exemplarisch, da keine 100 Gegner vorhanden sind).

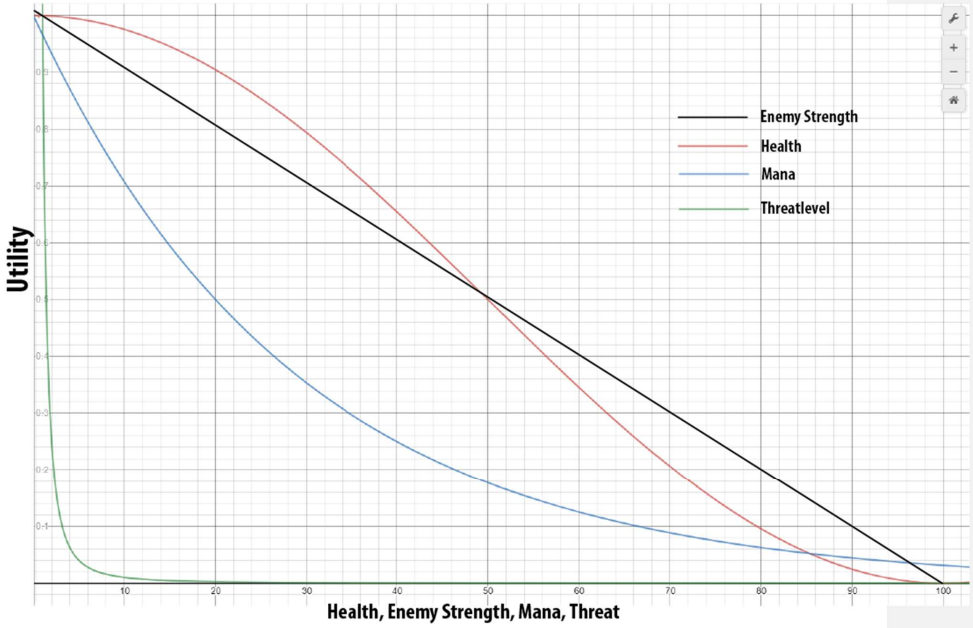

Abbildung 2: Darstellung der einzelnen Utilities im Graphen

4 Fazit

Den Ansatz einer Utility based Game AI finde sticht besonders durch seine Modularität sehr sinnvoll hervor. Es ist sehr schnell und leicht, durch weitere AI Behaviors erweiterbar, ohne die Gefahr, dass dadurch andere Verhaltensweisen beeinflusst werden. Weiter hat es einen Vorteil, im Vergleich zum Behavior Tree darin, dass es leichter durchschaubar ist. Jedes Behavior ist klar abgetrennt von den anderen und kann verändert bzw. angepasst werden, die Anzahl an Tasks bzw. Scripts insgesamt ist nicht so hoch, wie die eines Behavior Trees, bei dem jeder custom Decorator, Service etc. ein einzelnen Task/ Script bedeutet.

Schwierigkeiten liegen vor allem im Balancing der einzelnen scores. Um ein wirklich gut funktionierendes System zu kreieren bedarf es enorm viel Zeit durch viele experimentieren und testen.

Durch seine Einfachheit, Anpassbarkeit und den Fakt, dass die KI bereits funktioniert ohne die gesamte Spielwelt oder sämtliche Umstände zu kennen, stellen Utility based Game AIs eine absolut adäquate Lösung dar.