BOSCH
Technik fürs Leben

DHBW
Duale Hochschule
Baden-Württemberg

Konzept und Implementierung einer KI für das Rundenstrategiespiel Pummelz

Dokumentation

Bachelor of Science

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Jona Krumrein, Ruben Hartenstein

19.05.2022

Bearbeitungszeitraum	07.02.-19.05.2022
Matrikelnummer, Kurs	8366074, 2746235, TINF19ITA
Ausbildungsfirma	Robert Bosch GmbH, Stuttgart
Betreuer	Prof. Dr. Falko Kötter

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Es ist ein in Unity und C# implementiertes Rundenstrategiespiel („Pummelz“) gegeben, in dem ähnlich wie beim Schach zwei Spieler mit blauen und roten Figuren gegeneinander antreten. Diese Spielfiguren haben dabei, jedoch unterschiedliche Lebens- und Angriffswerte und können sich anhand ihrer gegebenen Reichweite beliebig bewegen. Zusätzlich besitzen einige Einheiten Spezialfähigkeiten, welche durch unterschiedliche Ereignisse ausgelöst werden können.

Ziel des Programmentwurfs ist es, ein KI zu entwickeln, welche eine bereits vorhandene Greedy-KI in jeglichen Spielszenarien schlägt und eine sehr hohe Spielstärke erreicht. Zudem soll die KI in der Lage sein, auch mit unbekannten Spielfiguren und Szenarien zu spielen. Hierzu soll eine passender Entwurf erstellt und umgesetzt werden. Im nächsten Schritt sollen die konzeptionierten Algorithmen und Funktionen überarbeitet und mittels mehrerer Iterationsschritten verbessert werden. Zuletzt soll eine Evaluierung der implementierten KI sowie deren Spielstärke durchgeführt werden.

2 Entwurf

Zu Beginn des Entwurfs wurden sich die folgenden strategischen Grundlagen überlegt. Diese dienen als Grundlage der KI und ihr Handeln in bestimmten Situationen.

- Wenn ein Czaremir (König) im Spiel ist muss sein Überleben jede Runde garantiert werden, da sein Tod das Spiel beendet. Im Gegenzug sollte der gegnerische priorisiert angegriffen werden, um einen schnellen Sieg erzwingen zu können.
- Viel hilft viel: in jedem Zug muss mit jeder Figur angegriffen werden die kann. **Außnahme** sind Angriffe auf:
  Bummz wenn er einem selbst mehr Schaden als dem Gegner zufügt
  Chilly wenn er nicht „Oneshot“ ist
- Schaden den der Gegner austeilen kann minimieren:
  - Schaden auf einen Gegner zu konzentrieren lohnt sich mehr als Schaden auf mehrere Gegner zu verteilen, da so Schaden in der nächsten Runde vermieden werden kann.
  - Gegner müssen anhand ihrer Eigenschaften klassifiziert werden. Gegner die mehr Schaden austeilen sind früher zu töten, da auch hier Schaden in der nächsten Runde vermieden werden kann
  - Die KI soll unbedingt die Reichweite der eigenen Einheiten ausnutzen. So soll die Anzahl der gegnerischen Einheiten, die eigene Einheit mit großer Reichweite angreifen kann, gering gehalten werden.
  - Bummz sich in die Richtung der Gegner und weg von den eigenen Einheiten bewegen.
- Einheiten mit geringer Reichweite und hohen Lebenspunkten als „Tanks“ einsetzen, um andere Einheiten zu schützen.
- Einheiten passend gruppieren, sodass Einheiten mit Flächeneffekten (Buffy, Haley) möglichst viele Einheiten um sich stehen haben.

- **ABER:** Aufpassen, dass möglichst viele eigene Einheiten angreifen können, um keine Blockade zu erzeugen und Schaden zu „verlieren“

Als grundlegendes Konzept soll ein DefaultDecisionTree konzipiert werden, welcher anhand der Spielsituation eine Entscheidung über den Spielzug der jeweiligen Spielfigur trifft. Hierbei wird eine Unterscheidung zwischen einem Angriff und einer Bewegung getroffen. Der Angriffentscheidung liegen drei Abfragen zu Grunde: Kann eine gegnerische Einheit „geoneshotted“, getötet oder angegriffen werden. Diese werden in der angegeben Reiehenfolge durchlaufen. Beim Zutreffen des Ereignisses soll dann ein Angriff durchgeführt werden. Die Entscheidung bei mehreren möglichen Einheiten erfolgt über eine festgelegte Hierarchie.

Ein Bewegungszug hat die folgenden beiden Möglichkeiten: Wurden zuvor mehrere Züge ohne einen Angriff vollzogen, so läuft die Einheit aggressiv auf die nächste gegnerische Einheit zu. Ist dies nicht der Fall soll eine Standard-Bewegung ausgeführt werden, bei der die Einheit im besten Fall eine andere angreifen und von keiner gegenerischen Einheit angegriffen werden kann.

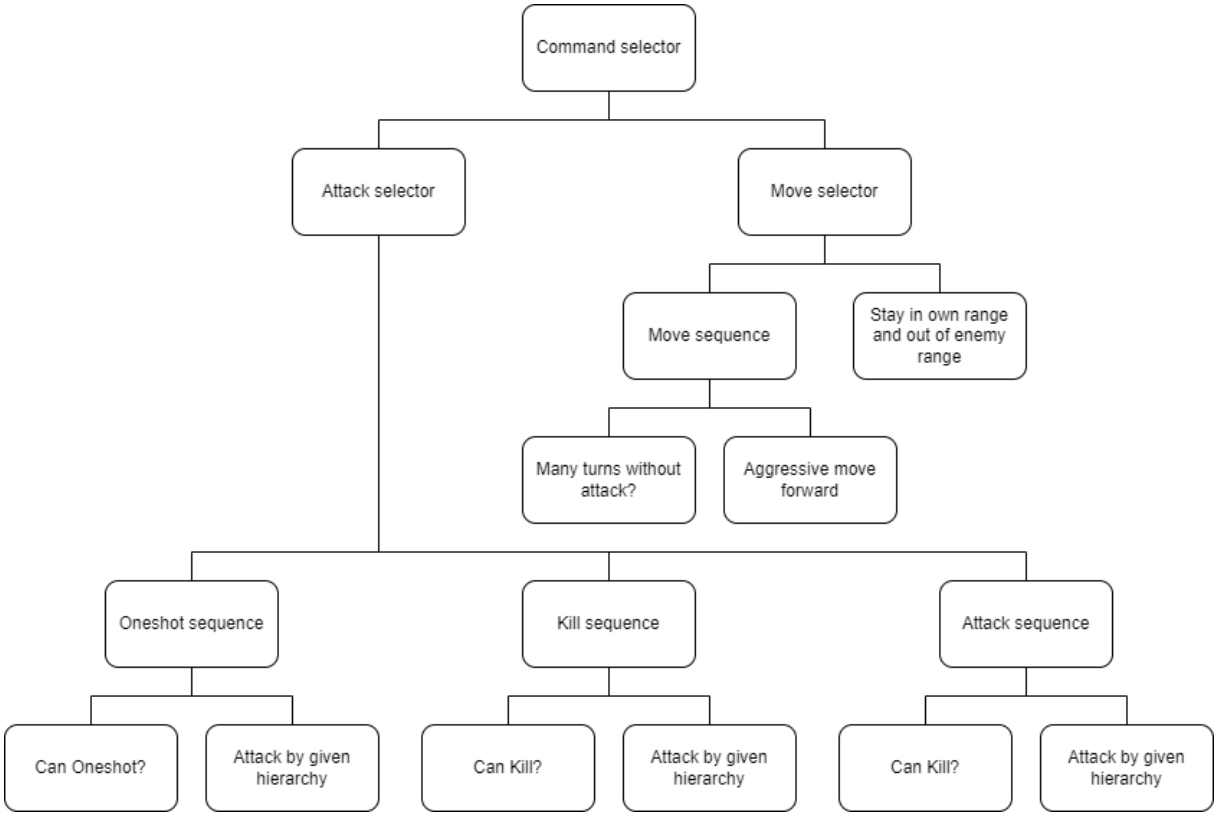

Abbildung 2.1: Konzept eines DecisionTrees zur Pummelz-Command-Entscheidung

Aufbauend auf diesem DefaultDecisionTree sollen weitere DecisionTrees erstellt werden, die auf die Eigenschaften und Spezialfähigkeiten der einzelnen Einheiten zugeschnitten sind und beim Zug der entsprechenden Einheit anstatt des DefaultTrees aufgerufen werden. So ist beispielsweise das Ziel von Bummz von den eigenen Einheiten weg in die gegnerischen zu Rennen. Sneip, Killy und Ängli hingegen sollen sich möglichst weit von gegnerischen Einheiten platzieren, um so wenig möglich Gefahr ausgesetzt zu sein.

3 Implementierung und Iteration

Zu Beginn unseres Spielzuges wird innerhalb des AIControllers über jede Einheit auf dem Spielfeld gelooped. Hierbei wird für jede Einheit der MGPumDecisisonTreeManager aufgerufen. Innerhalb des Managers wird anhand des Namens der Einheit der jeweilige DecisionTree aufgerufen, um einen Zug zu ermitteln. Falls für die Einheit kein spezieller DecisionTree umgesetzt wurde, wird auf einen DefaultTree zurückgegriffen. Der ermittelt Zug wird dann zurückgegeben und ausgeführt.
Das Grundgerüst legt dabei die abstrakte Klasse MGPumDecisionTree. Dies enthält alle wichtigen Funktionen, wie beispielsweise das Ermitteln aller Angriffszüge und Bewegungen oder des kürzesten Weg zwischen zwei Feldern. Alle speziellen DecisisonTrees erben später von diesem.

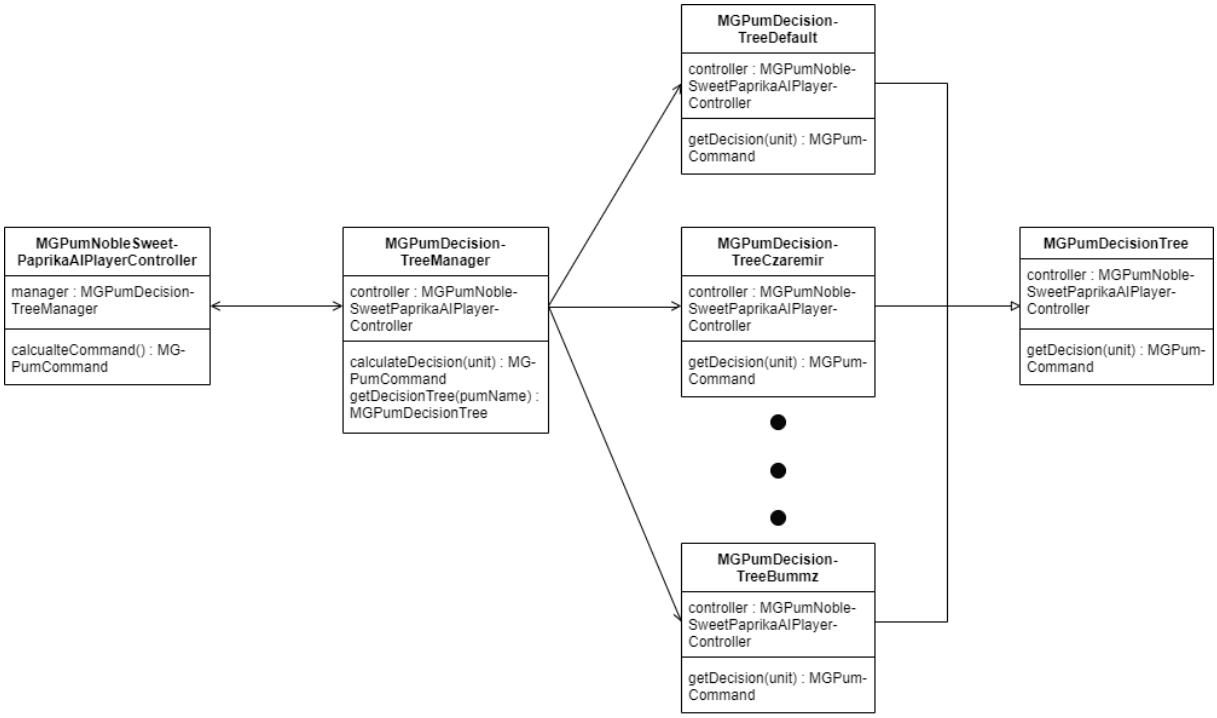

Abbildung 3.1: Klassendiagramm der vollständigen Implementierung

Das standardmäßige Angreifen erfolgt nach folgendem Ablauf. Im ersten Schritt werden mit Hilfe der Angriffsreichweite der Einheit alle möglichen Ziele und die Angriffszüge zu diesen

ermittelt. Nun wird für jedes Ziel berechnet, ob dieses durch den Angriff stirbt. Es ergibt sich eine Liste aller besiegbaren Einheiten. Diese Liste wird nun an den MGPumUnitComparer übergeben, welcher den bestmöglichen Angriff auswählt. Wird keine zu tötende Einheit gefunden, wird die Liste aller angreifbaren Einheiten verwendet. Die im Spiel enthaltenen Einheit befinden sich hier in einer Hierarchiestruktur und bekommen eine Punktzahl zugeordnet. Der Comparer vergleicht nun alle möglichen Ziele und sortiert diese nach dem besten Zug. Dieser kann dann ausgelesene und ausgeführt werden.

Das standardmäßige Laufen erfolgt nach einem ähnlichem Ablauf. Im ersten Schritt werden mit Hilfe der Bewegungsreichweite der Einheit alle möglichen Felder zu denen sich die Einheit bewegen kann ermittelt. Um das bestmögliche Zielfeld zu ermitteln, wird der MGPumMoveComparer verwendet. Dieser berechnet für jedes Feld eine Bewertung. Das Grundprinzip ist dabei genau eine gegnerische Einheit in Angriffsreichweite zu haben und von keiner Einheit angegriffen werden zu können. Hierzu wurde folgende Bewertungsfunktion eingesetzt:

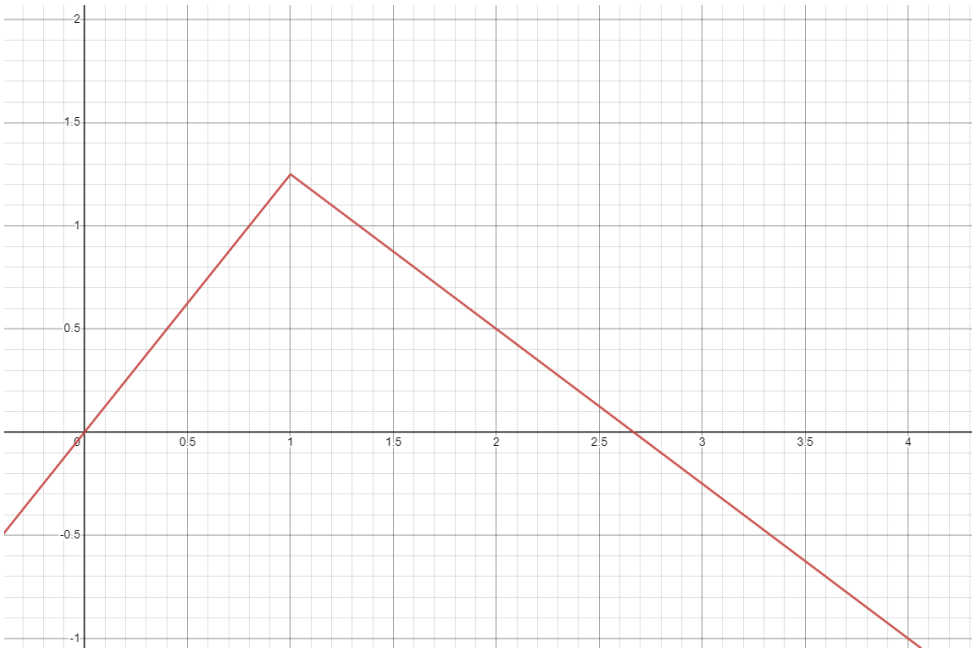

Abbildung 3.2: Funktion zur Bewertung der Anzahl der möglichen Ziele

Diese wurde so angepasst, das ein Ziel die beste Bewertung erhält und zwei Ziele immer noch besser als kein Ziel zu bewerten ist, um ein zu passives Spielen zu verhindern. Von diesem Wert wird dann noch die Anzahl der Angreifer in Reichweite abgezogen. Auch diese Bewegungen werden anhand ihrer Bewertung sortiert und die beste davon ausgeführt.

1. Iteration: Anpassung für Czaremir

Nach einigen Spielbeobachtungen lässt sich erkennen, dass der Czaremir aufgrund des DefaultTrees oft zu aggressive Bewegungen durchführt, sich zu nah an Gegnern positioniert und deshalb unnötigen Schaden nimmt. Dementsprechend wird für ihn ein separater DecisionTree implementiert, welcher ihn passiver spielen lässt. Das bedeutet, dass Felder mit weniger Angreifer noch besser bewertet werden.

2. Iteration: Anpassung für Range/Melee-Einheiten

Nach weiteren Beobachtungen lässt sich erkennen, dass das allgemeine Verhalten zu passiv ist. Vor allem für Einheiten mit kleiner Angriffsreichweite lohnt es sich nicht seine Ziel auf 1 zu beschränken. Geht man davon aus, dass der Gegner ebenfalls versucht mit jeder möglichen Einheit anzugreifen, lässt sich allgemein sagen, dass das aggressivere spielen der Melee-Einheiten sinnvoller ist. Außerdem lässt sich so die im Konzept überlegt Tank-Reihe umsetzen, welche das Schützen von Range-Einheiten ermöglicht. Hierfür wird Anzahl der Angreifer zur Bewertung hinzugefügt statt abgezogen. Zudem wird die Positionierung bei eigenen Einheiten belohnt.

3. Iteration: Anpassung für Bummz

Für die Bummz Einheit lässt sich schnell erkennen, dass eine passive Strategie sehr schlecht ist, da bei einer Explosion dem eigenen Team sehr viel Schaden zugefügt wird. Aus diesem Grund muss hier die Bewertung umgedreht werden. Felder sind dann gut bewertet, wenn wenig eigene und viele gegnerische Einheiten neben Bummz stehen. So kann seine Fähigkeit genutzt werden, um noch mehr Schaden am Gegner anzurichten.

Auch im Sinne der Angriffsbewertung wurde die Fähigkeit von Bummz betrachtet. In der Hierarchie der Einheit erhält dieser nun keinen festen Wert mehr sondern eine Funktion, welche den Angriff an der momentanen Stelle bewertet. Diese setzt sich zusammen aus dem verteilten Schaden an eigenen und gegnerischen Einheiten, wobei hier die Hierarchie der Einheiten und die mögliche rekursive Auslösung von anderen Bomben mit einbezogen wird. So erhält man je nach Feld und Spielstand eine individuelle Bewertung des Angriffs.

4. Iteration: Anpassung für Draw

Nach 50 Zügen ohne Angriff erreicht das Spiel den Draw-Zustand. Ein Draw zeugt jedoch nicht von einer hohen Spielstärke und sollte vermieden werden. Deshalb soll zusätzlich zum DecisionTree nach 20 Zügen ohne Angriff das Zulaufen auf gegnerische Einheiten belohnt werden. Dies ist vor allem deshalb wichtig, da bei wenigen Figuren auf dem Spielfeld die Felder ohne Angreifer zu gut bewertet werden und die Einheit vor dem Gegner wegläuft.

5. Iteration: Anpassung für Buff-Einheiten

Buff-Einheiten haben eine spezielle Fähigkeit, welche ihre oder die Eigenschaften von Team-Mitgliedern verbessert. Um diese aktivieren zu können müssen diese direkt neben anderen Einheiten stehen. Der DecisionTree wird dementsprechend angepasst, dass ein Feld neben eigenen Einheiten eine bessere Bewertung erhält als Felder ohne. Im Falle von Haley und Buffy werden hierbei alle eigenen Einheiten mit einbezogen. Da Links nur eigene Links verstärken werden hierfür auch nur die andere Links betrachtet.

4 Evaluation

Anhang