Otto-Friedrich-Universität Bamberg

Smart Environments und Kognitive Systeme

SME-Projekt-B: AI-Birds

Projektbericht des Teams „Meta“

Autoren: Anne Schwarz, Susanna Cao
Betreuer/-innen: Prof. Dr. Diedrich Wolter, Prof. Dr. Ute Schmid
Bamberg, Sommersemester 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Abstract

Dieser Bericht stellt die Arbeit des Meta-Teams des Projekts SME-Projekt-B im Sommersemester 2017 vor. Begleitet wurde dies durch Prof. Dr. Diedrich Wolter sowie Prof. Dr. Ute Schmid. Ziel des Projekts ist die Entwicklung eines intelligenten Agenten, der am Wettbewerb „Angry Birds AI Competition“[1] teilnimmt.
Das erste Kapitel gibt eine Einführung in das Thema und beschreibt die Problemstellung. Außerdem wird darauf eingegangen, warum es von großer Bedeutung ist, genau für Angry Birds einen intelligenten Agenten zu entwickeln.
Das darauffolgende Kapitel umfasst zuerst das Thema, warum vor allem Angry Birds eine Herausforderung im Bereich der künstlichen Intelligenz darstellt. Danach folgt ein Überblick über den IJCAI 2017 Angry Birds AI Wettbewerb und es wird auf die Aufgaben des Meta-Teams eingegangen. Dabei wird auch eine grobe Beschreibung des Gesamtverlaufs des Agenten und seine einzelnen Komponenten neben dem Meta-Bereich gegeben.
Anschließend werden die wichtigsten Arbeitsbereiche der Meta-Gruppe und ihres Arbeitsablaufs im Detail vorgestellt. Zum Schluss erfolgt ein Vergleich mit dem Agenten des letzten Jahres, wobei zusätzlich Vorschläge für einen besseren Agenten aufgeführt werden.

[1] `https://aibirds.org/` (zuletzt abgerufen: 14.09.2017)

2 Einführung

Angry Birds gilt als eines der beliebtesten und bekanntesten physik-basierten Simulationsspiele (engl. physics-based simulation game (PBSG)) [4]. Seit 2009 hat das Spiel über 200 Millionen Spieler weltweit angezogen [2]. Das liegt u.a. daran, dass das Spielprinzip leicht zu verstehen und die Spielmechaniken leicht erlernbar sind. Das Ziel von Angry Birds ist es, alle Schweine und so viele Hindernisse wie möglich mit einer begrenzten Anzahl an Vögeln zu zerstören. Dazu werden diese mit Hilfe einer Schleuder auf eine Struktur geschossen, die sehr kompliziert sein kann und aus einer Reihe von verschiedenen Objekten mit unterschiedlichen Eigenschaften besteht. Wie in Abbildung 1 ersichtlich gibt es unter anderem Bausteine aus Holz, Eis und Stein. Außerdem gibt es auch unterschiedliche Vögel, die jeweils spezifische Eigenschaften besitzen. So kann der gelbe Vogel z.B. effektiv Holz zerstören oder der schwarze Vogel Stein.[2]

Abbildung 1: Ausschnitt aus Level 21 Angry Birds [2]
Das Bild zeigt ein Level aus Angry Birds mit einer komplizierten Struktur, die aus verschiedenen Objekten mit unterschiedlichen Eigenschaften besteht, hier dargestellt durch Bounding-Boxen.

Aus Sicht von Künstlicher Intelligenz vereint das Spiel Aspekte aus den Bereichen Computer Vision, Maschinelles Lernen, Wissensrepräsentation, Planen, Heuristische Suche und Schlussfolgern unter Unsicherheit [4]. Einen intelligenten Agenten für dieses Spiel zu entwickeln, der besser sein soll als menschliche Spieler, ist genau deshalb für die Forschung von großer Bedeutung und stellt das Team vor schwierige Herausforderungen.

[2] `https://de.wikipedia.org/wiki/Angry_Birds` (zuletzt abgerufen: 10.09.2017)

3 Hintergrund und Verwandte Arbeiten

In diesem Abschnitt werden weitere Informationen zu Angry Birds und dem IJCAI Angry Birds Wettbewerb vorgstellt. Insbesondere wird auf die Frage eingegangen, warum gerade Angry Birds als eine Herausforderung im Forschungsbereich Künstliche Intelligenz anzusehen ist. Zudem wird auf verwandte Arbeiten hinsichtlich der Problemstellung eingegangen.

3.1 Angry Birds als Herausforderung im Bereich Künstliche Intelligenz

[Jochen Renz et al.] beschäftigt sich im Zuge seines Berichts „AiBirds: The angry birds artificial intelligence competition.“ über Angry Birds als Herausforderung im Bereich der Künstliche Intelligenz. Seine Ergebnisse werden in diesem Abschnitt vorgstellt. [4]
In physikbasierten Simulationsspielen wie Angry Birds ist die gesamte Spielwelt typischerweise parametrisiert. So sind beispielsweise alle physischen Parameter, wie die Masse, die Reibung, die Dichte von Objekten und die Gravität sowie alle Objekttypen und deren Eigenschaften und Lage intern bekannt. Jede gewählte Aktion kann deshalb mit einem zugrundeliegenden Physik-Simulator sehr realgetreu nachgebildet werden. Einfache Aktionen können wie folgt beschrieben werden: der Spieler kann erstens entscheiden an welchem Punkt <x,y> der Vogel aus der Schleuder gelassen werden soll und zweitens wann während des Flugs die Superkraft des Vogels aktiviert werden soll. In der Praxis ergibt dies eine sehr große Anzahl an möglichen Aktionen. Während des Spielens gilt ein Level immer dann als gelöst, wenn eine Reihe von Aktionen zu einem Spielstand führt, der bestimmte Siegbedingungen erfüllt. [4]
Aus dem einfachen Spiel und den einfachen Aktionen resultiert, so Jochen Renz weiter, dass auch kleine Kinder das Spiel schon erfolgreich durchspielen können. Die Herausforderung des Wettbewerbs besteht also darin einen intelligenten Agenten zu bauen, der neue Level genauso gut oder sogar erfolgreicher spielen kann als die besten menschlichen Spieler. Im Vergleich zu scheinbar komplizierteren Spielen, wie z.B. Schach, klingt dies nach einer relativ einfachen Aufgabe, allerdings müssen einige Herausforderungen gemeistert werden. Unter der Annahme, dass alle Parameter der Spielwelt bekannt und parametrisiert sind, könnten Aktionen und deren Folgeaktionen solange simuliert werden bis der Siegzustand erreicht ist. Wenn die Handlungen dann noch intelligent ausgewählt werden, kann dies zu einer erfolgreichen Lösungsstrategie führen.
Hier liegt laut dem Artikel aber das Hauptproblem physikbasierter Simulationsspiele: das Ergebnis von Aktionen ist immer erst dann bekannt, wenn diese simuliert werden, was wiederum bedeutet, dass auch alle dafür benötigten Parameter bekannt sein müssen. Es liegt also eine andere Grundlage vor als bei Spielen wie Schach, in denen die Ergebnisse jeder Aktion schon im Voraus bekannt sind. Hinzu kommt, dass sich die Spielwelt nach jedem Schuss enorm verändert. Dies erfordert also präzise Vorhersagen oder Annäherungen an die resultierenden Konsequenzen, um eine Strategie aus Aktionen entwickeln

zu können. Die Kombination aus einem potentiell unendlichen Aktionsraum und dem möglichen Mangel an Informationen über alle Parameter stellt hinsichtlich des Treffens von Vorhersagen eine der größten Herausforderungen dar. Für den Menschen eine relativ einfache Aufgabe, allerdings ist das Kombinieren in unbekannten Umgebungen noch immer eine Forschungsfrage.
Die Forschung an physikbasierten Simulationsspielen wie Angry Birds ist deshalb so wichtig, weil die gleichen Probleme auch noch für AI-Systeme gelöst werden müssen, damit diese erfolgreich mit der physischen Welt interagieren können. Während Menschen diese Fähigkeit von Natur aus besitzen und ständig einsetzen, ist aus der Sicht von AI noch nicht das ganze Potential ausgeschöpft. Der IJCAI 2017 Angry Birds AI Wettbewerb bietet dazu eine geeignete Plattform an, um diese Herausforderungen und neuen Fähigkeiten in einer vereinfachten und kontrollierten Umgebung zu testen und zu entwickeln.

3.2 Verwandte Arbeiten

2013 veröffentlichte [Calimeri et al.], 2013 den Artikel „AngryHEX: an Artificial Player for Angry Birds Based on Declarative Knowledge Bases“. Diese Arbeit präsentiert ein gemeinsames Projekt der Universität von Kalabrien (UniCal) und der TU Wien, mit dem Ziel, einen intelligenten Agenten zu entwickeln, der am 2013 Angry Birds AI Wettbewerb teilnimmt. AngryHEX, so die Autoren, basiert auf einer Programmierung mit Hilfe von ASP (Answer Set Programming). Die grundlegende Spielsoftware, die von den Veranstaltern zur Verfügung gestellt wird, wurde ebenfalls verwendet. Diese wurde allerdings noch durch einige Erweiterungen modifiziert. [1]
Im selben Jahr wird die Aufgabe des Projekts von [Ferreira et al.] wie folgt beschrieben: das Ziel ist es, „einen autonomen Agenten zu entwickeln, der in der Lage ist, ohne menschliches Eingreifen [Angry Birds] zu spielen“ [3]. Dabei ist dieser in der Lage, seine Umgebung zu betrachten und mit einzubeziehen. Außerdem ist der Agent in der Lage das beste Objekt als Ziel auszuwählen. Der Artikel beschreibt weiterhin die Entwicklung eines intelligenten Agenten namens FEI2, der für einen Wettbewerb 2012 entwickelt wurde. Der Schwerpunkt, so [Ferreira et al., 2013] weiter lag dabei auf der Auswahl des bestmöglichen Ziels. Zudem wurden für die Entwicklung drei Konzepte kombiniert. Einmal lag die Konzentration auf der räumlichen Darstellung der einzelnen Objekte, weil die Positionen der Objekte und die Lagebeziehungen derer untereinander wichtig seien. Die Autoren schreiben weiter: „The second concept is that of Utility Function, which allows the agent to represent preferences for the options that are given to it“. Das zweite Konzept gibt dem Agenten also die Funktion, Präferenzen für ein Zielobjekt abzugeben, was wichtig bei der Entscheidungsfindung ist. Schließlich wurde als letztes Konzept Schlussfolgern unter Unsicherheit verwendet. [3]
Mihai Polceanu und Cedric Buche konzentrieren sich in ihrem Artikel auf einen bestimmten Bereich für die Ausarbeitung eines intelligenten Agenten. Sie schreiben über verschiedene Entscheidungsmechanismen. Dabei liegt der Fokus vor allem auf der Erwartungsfähigkeit des Menschen und seiner Fähigkeit sich anzupassen, während diese

interagieren. Da dieser Bereich auch für den intelligenten Agenten erarbeitet und implementiert werden soll, ist auch dieser Artikel sehr hilfreich bei der Ausarbeitung. [2]

3.3 Der IJCAI 2017 Angry Birds AI Wettbewerb

Im Zuge des Wettbewerbs sollen die Fähigkeiten des entwickelten intelligenten Agenten für Angry Birds getestet werden. Dazu werden von Seiten der Veranstalter eine Vielzahl von Level angeboten, um in einer vereinfachten und kontrollierten Umgebung die Entwicklung und Erprobung des Agenten voranzutreiben. Langfristig gesehen, soll der Wettbewerb beim Aufbau eines AI-Agenten unterstützen, der für ihn unbekannte Level besser spielen kann als die besten menschlichen Angry Birds Spieler. [5]
Wie oben bereits beschrieben, ist dies ein sehr schwieriges Problem, da es erfordert, dass der Agent Handlungen vorhersagen kann, ohne aber die Spielwelt komplett zu kennen. Zusätzlich muss eine gute Handlungsauswahl implementiert werden.
Zu beachten ist, dass Level in beliebiger Reihenfolge gespielt und auch so oft wie nötig wiederholt werden können [4]. Die Agenten werden danach anhand der insgesamt erreichten Punkte bewertet. Während des Wettbewerbs „Mensch gegen Maschine“ wird dann getestet, ob der Agent besser ist als menschliche Spieler. Folgende Probleme müssen dafür effizient gelöst werden [2]:

- Erkennen und Klassifizieren bekannter und unbekannter Objekte,
- Erlernen von Eigenschaften der (unbekannten) Objekte und der Spielwelt,
- Vorhersage des Handlungsergebnisses,
- Auswahl guter Aktionen in vorgegebenen Situationen und
- Planung einer erfolgreichen Aktionssequenz sowie
- der Reihenfolge in der Level gespielt werden sollen.

Gespielt wird die Google Chrome Version von Angry Birds, die unter chrome.angrybirds.com öffentlich zugänglich ist. Der zur Verfügung gestellte Wettkampfserver verbindet sich dadurch mit der zuvor eingerichteten Chrome Browser Erweiterung, durch die es möglich ist Screenshots des Spiels während der Laufzeit aufzunehmen. Außerdem können darüber auch die verschiedenen Aktionen per Mausklick ausgeführt werden. Teilnehmende Agenten, die auf verschiedenen Clients laufen müssen, können nur über ein festes Kommunikationsprotokoll mit dem Server interagieren. Dies ermöglicht dem Agenten das Anfordern von Screenshots, Aktionen und anderen Befehlen vom Server und das Ausführen dieser im Live-Spiel. U.a. kann auch der aktuelle Highscore jederzeit vom Server abgerufen werden. Den Agenten wird so die gleiche Information wie dem Menschen zur Verfügung gestellt. Die genaue Lage oder Parameter von Objekten bleiben allerdings trotzdem unbekannt.
Um das Problem zu vereinfachen und sich ganz auf die Entwicklung des Agenten zu

konzentrieren, verwenden wir die von den Organisatoren zur Verfügung gestellte grundlegende Spielsoftware. Das Grundgerüst dieser Software besteht aus drei Komponenten [2]:

- Computer-Vision-Komponente (engl.: computer vision component), die einen Ausschnitt des Videospiels analysieren kann und den Ort, die Kategorie und die Begrenzungsbox (Bounding Box) aller relevanten Objekte sowie den Spielstand indentifiziert
- Trajektorien-Komponente (engl.: trajectory component), die Trajektorien von Vögeln berechnet und vorgibt, wohin man schießen muss, um ein bestimmtes Objekte oder einen bestimmten Ort zu treffen
- Spielkomponente, die Aktionen ausführt und Screenshots erfasst.

3.4 Meta-Strategie des Agenten „BamBird“ 2017

Für die Teilnahme am Wettbewerb hat sich das gesamte Projektteam in insgesamt vier Untergruppen geteilt. Jede Gruppe übernahm eine andere Aufgabe für die Entwicklung des Agenten. Die Aufteilung war wie folgt: Physiksimulation, Entwicklung neuer Schuss-Strategien, Anpassung des Schussmoduls und Meta-Strategie.
Das Zusammenspiel der Komponenten ist in folgendem Diagramm dargestellt:

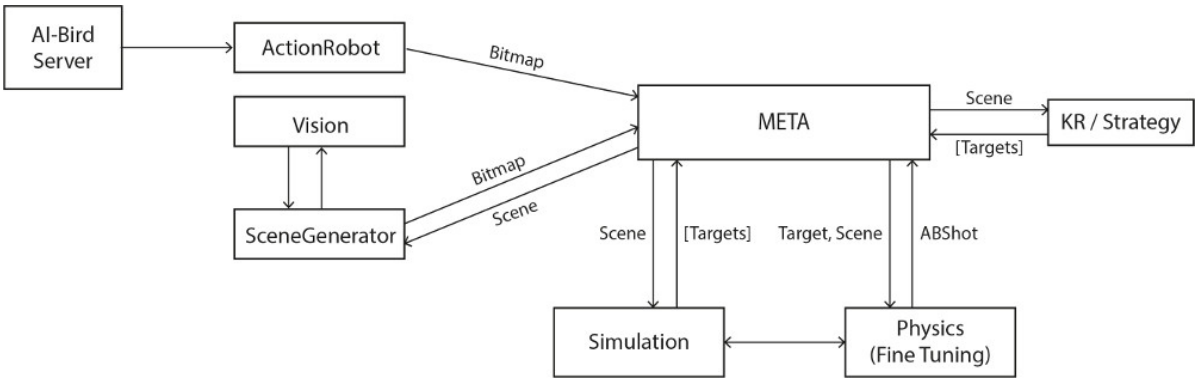

Abbildung 2: Wechselspiel der einzelnen Komponenten des Wettbewerbs.

In diesem Bericht soll insbesondere auf die Arbeit des Meta-Teams eingegangen werden. Diese übergreift alle anderen Komponenten und führt diese zusammen, die Gruppe „Meta“ ist also eine Art Schnittstelle für die anderen Gruppen.
Wir haben uns insbesondere folgende Arbeitsfeldern angenommen. Als Erstes haben wir uns mit der Auswahl der verschiedenen Level während des Spiels beschäftigt. So nehmen wir Einfluss auf den Spielfluss und entscheiden in welcher Reihenfolge und wie oft die einzelnen Level gespielt werden sollen. Eine weitere wichtige Aufgabe ist es, die richtigen Aktionen auszuwählen. Wir haben uns in dieser Hinsicht auf die Auswahl der Schüsse

spezialisiert. Aus einer Liste, die wir von der Strategie-Gruppe erhalten, wählen wir, den als besten markierten Schuss aus. Zur Vorhersage einer guten Handlung haben wir uns des weiteren eine Form der Evaluation einzelner Schüsse überlegt, um besonders gute Schüsse zu kennzeichnen und diese in ähnlichen Situationen wieder anzuwenden. Alle Informationen werden zu jedem einzelnen Schuss in einer Datenbank gespeichert, die wir implementiert haben.
Genaue Informationen über den Vorgang unserer Implementierung werden im folgenden Kapitel näher ausgeführt.

4 Umsetzung der Aufgaben des Meta-Bereichs

Dieser Abschnitt beschäftigt sich mit unserer Umsetzung der Aufgaben des Meta-Bereichs. Unser Ziel war es, nach jedem Durchlauf eines Levels zu „lernen“, indem man das vorherige Handeln speichert, evaluiert und dies beim wiederholten Durchlauf berücksichtigt.

4.1 Meta

Die Hauptklasse `Bambird`, die sich im Ordner `main` befindet, stellt die Verbindung vom Client zum Server her und ihre Instanz entspricht unserer Version des intelligenten Agenten „BamBird“, der Angry Birds spielt.
Darin wird ein Objekt der Klasse `Meta` angelegt, in der das eigentliche Zusammenspiel aller im vorigen Kapitel genannten Komponenten (jedoch ohne Physik-Simulation) stattfindet – sie entspricht also der tatsächlichen „Main-Klasse“. Zur Veranschaulichung des Aufbaus der Hauptmethode `startMeta`, wird deren Ablauf in folgendem Pseudocode 1 dargestellt:

Algorithm 1 Meta

```
1: loop
2:     if GameState != Playing then
3:         choose new Level
4:     end if
5:     take a screenshot from the scene and analyse
6:     generate plans
7:     choose one plan to be executed as a shot
8:     execute chosen shot
9:     add shot to the shot-list of the current level
10:    if GameState == WON or GameState == LOST then
11:        add level to database
12:    end if
13: end loop
```

Die Schleife wird ständig durchlaufen bis das Spiel zu Ende ist bzw. die vom Wettbewerb vorgegebene Zeit abgelaufen ist. Das Level wird einmal durchgespielt und anschließend mit den ausgeführten Schüssen in einer Datenbank abgespeichert. Erst wenn der Spielstatus nicht mehr auf „Playing“ steht, wird ein neues Level ausgewählt und der gleiche Ablauf folgt.
Wir, im Meta-Team, beschäftigten uns gemeinsam mit drei Hauptproblemen für die Entwicklung des intelligenten Agenten. Auf diese wird im Folgenden genauer eingegangen.

4.2 Level und Datenbank

4.2.1 Level

Zur Speicherung der ausgeführten Aktionen innerhalb eines Levels wird eine Datenbank benötigt. Dazu muss vorab geklärt werden, welche grundlegenden Informationen über ein Level gespeichert werden sollen. Außerdem muss beachtet werden, dass diese so angelegt und gespeichert werden, dass sie mit anderen Komponenten kompatibel sind.
Während der Entwicklung des Agenten haben sich nachstehende Informationen als sinnvoll erwiesen. Diese befinden sich in der Java-Klasse `Level`, welche sich im Ordner `Meta` befindet. Diese besteht aus

- der Level-ID,
- den geschätzten maximal zu erreichenden Punkten dieses Levels,
- den tatsächlich erreichten Punkten,
- der Anzahl der gespielten Durchgänge und
- einer Liste von ausgeführten Schüssen.

Das Abspeichern der Schüsse erfolgt in Form der Klasse `Triplet`, die sich auch im `Meta`-Ordner befindet. Hierbei wird neben dem eigentlichen Schuss (`shot`) zusätzlich noch das anvisierte Zielobjekt (`target`) und die allein aus diesem Schuss erreichten Punkte (`damagePoints`) gespeichert, welche später in der `ShotSelection` (siehe Kapitel 4.4) relevant sind.
Die `Level`-Klasse an sich dient als Grundlage und Speicherobjekt und enthält dementsprechend nur wenige Methoden.

4.2.2 Datenbank

Nach jedem Durchlauf eines Levels werden die oben genannten Informationen in ein `Level`-Objekt gespeichert, wobei jedes einzelne davon wiederum in eine Datenbank hinzugefügt wird. Die dazugehörige Klasse ist unter database » LevelStorage zu finden.
Der Ordner `database` enthält ebenso eine enum-Klasse `LevelState`, welche wir nur zur Markierung der Level einsetzen, weiterhin jedoch keine höhere Verwendung findet.
Die `LevelStorage` besteht aus einer privaten Map, die die `Level` und den dazugehörigen `LevelState` beinhaltet. Zusätzlich enthält die Klasse eine öffentliche Liste aus Integer, die in der gleichen Reihenfolge wie die Map die Level-IDs der gespielten Level speichert. Dadurch kann von außen schnell auf die Informationen von Leveln zugegriffen werden, die bereits gespielt wurden. Außerdem lassen sich auch die Indizes der Level leichter abfragen.
Im Falle eines wiederholten Versuchs, muss beim Speichern darauf geachtet werden, dass die Lebel nicht doppelt hinterlegt werden. Daher wird in der öffentlichen Methode `addLeveltoStorage` geprüft, ob das übergebene Level bereits in der Datenbank

enthalten ist. Falls es einen Eintrag mit dieser Level-ID gibt, wird die Hilfsmethode `updateLevelInfo` aufgerufen, welche nur die geänderten Einträge aktualisiert, anstatt einen komplett neuen Eintrag zu erstellen.
Die `LevelStorage` ist in der Evaluation von großer Bedeutung und wird in den Klassen der nachfolgenden Kapitel verwendet.

4.3 Level Selection

Die Klasse `LevelSelection`, die sich im Ordner `Meta` befindet, ist für die Levelauswahl zuständig und wird zu Beginn der Schleife aus dem `Meta`-Code aufgerufen (siehe Pseudocode 1). Sie hält die Information über die gesamte Anzahl der zu spielenden Level und über das aktuelle Level selbst, das gerade gespielt wird.
Die Hauptmethode `selectNextLevel` lässt alle Level beginnend mit einer zufällig ausgewählten Levelnummer der Reihenfolge nach durchspielen. Sobald all diese einmal vollendet wurden, muss entschieden werden, welche Level in welcher Reihenfolge und wie oft wiederholt werden sollen.
Die Auswahl erfolgt nach einer simplen Wahrscheinlichkeitsberechnung, welche aussagt, wie wahrscheinlich es ist, mit einem bestimmten Level noch mehr Punkte zu erzielen bzw. sich in diesem Level zu verbessern:

$$Probability = 1 - (actualScore/maximalReachableScore)$$

Wir richten uns also danach, wie viele Punkte wir von der maximal erreichbaren Punktzahl im ersten Durchgang bereits abdecken konnten. Das Level mit der höchsten Wahrscheinlichkeit wird dann ausgewählt.
Für verlorene Level liegt eine Besonderheit vor, da ihre bereits erbrachte Punktzahl bei 0 liegt. Somit beträgt deren Wahrscheinlichkeit sich zu verbessern 1 und sie werden deshalb immer als Erstes ausgewählt. Da man für jedes Level im Durchschnitt mindestens drei Minuten erhält[3] und wir von einer Durchschnittsspieldauer von 1 - 1,5 Minuten pro Level ausgingen, entschieden wir uns, die verlorenen Level zunächst höchstens zweimal wiederholen zu lassen. Falls die Quote der verlorenen Level im Bezug auf die Gesamtanzahl der zu spielenden Level dann immer noch zu hoch ist, soll der Agent ein weiteres verlorenes Level auswählen. In unserem Fall haben wir die Grenze auf 15% gesetzt, d.h. der Agent würde bei einer Gesamtanzahl von 21 Level einen erneuten Versuch an verlorenen Level starten, wenn mehr als drei Level noch verloren sind. Falls dann immer noch nicht alle Level gewonnen wurden, sollen diese ignoriert werden.
Zur Veranschaulichung, wie der Algorithmus implementiert ist, dient der Pseudocode 2. Hierbei haben wir einen boolean-Wert `ignoreLostLevels` integriert, der erst dann auf **true** gesetzt wird, wenn alle verlorenen Level zweimal wiederholt wurden und die Quote der verlorenen Level unter 15% beträgt. Dieser Wert wird dann in der zweiten if-Schleife

[3] `https://aibirds.org/angry-birds-ai-competition/competition-rules.html` (zuletzt abgerufen: 05.09.2017)

geprüft.

Algorithm 2 Level selection after every level was played at least once

```
1:  boolean ignoreLostLevels = false
2:
3:  calculateProbabilities()                    ▷ calculates the probability for every level
4:
5:  if lost levels exist then
6:      if all lost levels were repeated twice then
7:          if Amount of lost levels > 15% of total number of levels then
8:              return next lost level                  ▷ redo the lost levels one more time
9:          else
10:             ignoreLostLevels = true
11:         end if
12:     else
13:         return next lost level that has not been played twice yet
14:     end if
15: end if
16:
17: if no lost levels exist or ignoreLostLevels == true then
18:     return level with the highest probability
19: end if
```

Ebenso verhindert werden soll, dass nicht das selbe Level immer und immer wieder ausgewählt wird. Dazu haben wir eine Klausel implementiert, die bewirkt, dass ein Level nur ausgewählt werden kann, wenn die Differenz zwischen seiner Spielanzahl und der minimalen Anzahl, wie oft ein Level schon gespielt wurde, die Zahl 2 nicht übersteigt. D.h. wenn das ausgewählte Level zum fünften Mal ausgewählt werden soll, ein anderes Level aber erst zweimal gespielt wurde, wird das Level mit der zweithöchsten Wahrscheinlichkeit ausgewählt. All dies nachdem alle verlorenen Level abgearbeitet wurden.

4.4 Shot Selection

Um bei wiederholten Versuchen von Leveln nicht ständig dieselben Schüsse zu nehmen und um unsere Spieltaktik zu verbessern, haben wir uns desweiteren auch mit der Schuss- oder auch Handlungsauswahl beschäftigt. Dafür bekommt unsere Klasse `ShotSelection`, die sich ebenfalls im `Meta`-Ordner befindet, von der Strategie-Gruppe für jeden Vogel eine Liste von Zielobjekten (`Targets`), auf die der Vogel schießen kann (siehe Methode `chooseShotWithOneList`). Eines davon wandeln wir dann in ein `Shot`-Objekt um woraufhin der Schuss dann vom Agenten im Spiel ausgeführt wird. Nachdem ein Schuss getätigt wurde, wird dieser dann für das entsprechende Level in eine Liste von Schüssen

gespeichert (vgl. Kapitel 4.2.1).
Zu den jeweiligen `Targets` wird eine Konfidenz zwischen 0 und 1 von der Strategie-Gruppe mitübergeben, nach welcher der auszuführenden Schuss ausgewählt wird.

4.4.1 Verhindern der Wiederholung von Schüssen

Es besteht die Gefahr, dass immer derselbe Schuss genommen wird. Dies kann eintreten, wenn ein Schuss einer hohen Konfidenz zugeordnet wird, im echten Spiel jedoch wenig oder sogar keine Wirkung auf die Schweine und Strukturen hat. Da nach jedem Schuss neue Pläne, in Bezug auf den jetzigen Stand der Umgebung, generiert werden und sich die Szene in dem Fall nicht verändert, erhalten wir von der Strategie-Gruppe stets die gleiche Liste von Plänen. Würden wir also den Algorithmus ohne Einschränkung lassen, würde immer das Zielobjekt mit der höchsten Konfidenz ausgewählt werden, welches bei gleichen Szenen immer demselben Schuss entspricht. Deswegen haben wir einen Algorithmus integriert, der in Bezug auf die noch übrig gebliebenen Vögel entscheidet, ob der Schuss ein zweites Mal probiert werden soll. Dies ist sinnvoll, da beispielsweise oft der Fall eintritt, dass der erste Schuss die Gebäude zum Wackeln bringt und ein zweiter Schuss auf das gleiche Ziel den Endstoßbringt. Dies lassen wir jedoch nur zu, wenn die Chance noch hoch genug ist, mit den restlichen Vögeln das Level trotz eines zweiten Fehlschusses noch gewinnen zu können. Veranschaulicht wird der Algorithmus in folgendem Pseudocode 3. Wir richten uns dabei nach der Gesamtzahl der Vögel:

Algorithm 3 Prevention of endless repetition in ShotSelection

```
1: if shotCandidate.equals(previousShot) then
2:     if (totalBirdAmount <= 3 && executedShots >= 2) or
3:      (totalBirdAmount > 3 && previousPreviousShot.equals(previousShot)) then
4:         choose another shotCandidate
5:     else
6:         proceed with the current shotCandidate
7:     end if
8: end if
```

- *shotCandidate*: der zu prüfende Schuss, der ausgeführt werden soll
- *previousShot*: der Schuss, der als letztes tatsächlich ausgeführt wurde
- *previousPreviousShot*: der Schuss, der vor dem zuletzt ausgeführten Schuss ausgeführt wurde

In unserem Algorithmus wird also nur derselbe Schuss, im Fall von mehr als drei Vögeln, noch einmal ausgeführt, wenn die zwei Schüsse vor dem aktuellen Schuss nicht schon identisch waren, d.h. ein Schuss darf nicht mehr als einmal wiederholt werden. Im Falle von höchstens drei Vögel, wird ein anderer Schuss ausgewählt, wenn es sich gerade um den letzten Vogel handelt.

4.4.2 Schussevaluation

In diesem Abschnitt geht es um die Bewertung eines Schusses. Um eine Aktion zu evaluieren, überprüfen wir zunächst, ob ein Level bereits gespielt wurde und ob der bestimmte Schuss schon im letzten Durchgang zum Einsatz kam. So können die Schüsse verglichen werden und es kann gesagt werden, ob der jeztige „gut“ oder „schlecht“ war. Wird ein Schuss als **BAD** markiert, soll ein anderes Zielobjekt ausgewählt werden.
Handelt es sich also bei dem momentanen *shotCandidate*, um einen Schuss, der beim vorigen Durchgang bereits verwendet wurde, ruft die Methode `chooseShotWithOneList` die private Methode `evaluate` auf, in der die Schussevaluation erfolgt. Die Methode gibt dann ein enum – **GOOD** oder **BAD** – zurück. Bei der Evaluation werden zwei Aspekte berücksichtigt:

1. **Punktzahl von den einzelnen Schüssen**
   Hierbei liegt der Fokus auf dem Schaden, den der einzelne Schuss im Bezug auf den aktuellen Levelzustand angerichtet hat. Nach jedem Schuss wird ausgerechnet, wie viele Punkte maximal noch erreicht werden können, teilen dies dann durch die Anzahl der noch bestehenden Vögel, sodass wir einen Durchschnittswert erhalten, und geben zurück, wie viel der ausgeführte Schuss von diesem Durchschnittswert abdeckt. D.h. falls man ein Level mit anfangs fünf Vögeln hat und einer davon bereits geschossen wurde, behandelt man das Level beim nächsten Schuss, als wären in dem Level nur vier Vögel zur Verfügung gestanden und rechnet von dem Stand aus, wie viele Punkte noch maximal zu erreichen sind. Die max. Punktezahl ist stets sehr hoch und unserer Meinung eigentlich unmöglich zu erreichen, da bei der Berechnung des Maximalwerts davon ausgegangen wird, dass man mit einem einzigen Vogel alle Schweine und Konstruktionen, sowie die Bonuspunkte für übrig gebliebene Vögel erhält. Dementsprechend ist der errechnete Durchschnittswert für einen Schuss auch sehr hoch und je näher ein Schuss diesem Wert kommt, desto besser wird der Schuss bewertet. Unserer Meinung nach, ist die erreichte Punktzahl eines Schusses ein ausschlaggebender Faktor, weshalb wir ihm eine Gewichtung von 0.7 in der Gesamtevaluation zuteilen (siehe `optimalShot`).

2. **Anzahl der Vögel bzw. Gesamtverlauf eines Levels**
   Da das Ziel von Angry Birds ist, mit wenig Vögeln alle Schweine zu vernichten und dabei so viel Schaden anzurichten wie möglich, muss auch die Gesamtbewertung, wie das Level beim letzten Durchgang ausging, in Betracht gezogen werden. Daher liegt der Fokus beim zweiten Aspekt auf der Anzahl der beim letzten Durchgang verwendeten Vögel. Wir rechnen dazu das Verhältnis aus, indem wir die tatsächlich verbrauchten Vögel durch die Gesamtanzahl teilen. Diesen Wert verrechnen wir dann mit einer Gewichtung von den restlichen 0.3 in die Gesamtevaluation mit ein (siehe `calculateRatioPoints`).

Ist die Summe der beiden errechnetenWerte dann kleiner als 0.45, markieren wir den Schuss als **BAD** und ein anderer zufälliger Schuss, aus der übergebenen Liste des

Strategie-Teams, wird ausgeführt.
Um auf diese Grenze zu kommen, haben wir einige Werte miteinander verglichen und sind der Fragestellung nachgegangen, was genau als guter Schuss definiert wird. Zuerst haben wir Bezug auf den ersten Punkt – Punktzahl der einzelnen Schüsse – genommen: Wie hoch muss die Punktzahl sein, damit der Schuss als **GOOD** bewertet wird? Dazu haben wir den Agenten Level 19 spielen lassen und die Werte in einer Tabelle zusammengefasst. Ebenso haben wir von den maximal erreichbaren Punkten nach jedem Schuss die Bonuspunkte für übrig gebliebene Vögel abgezogen, da wir nur den Schuss und dessen Punktzahl alleine betrachten müssen. Tabelle 3 zeigt die Ergebnisse:

	Schuss					
Level	Vögel	Punkte	**Max. Punktzahl**	**Max. Punktzahl - Vögel**	**Max. Durchschnitt**	**Punkte / Max. Durschnitt**
19	1	5180	56400	26400	6600	0,78
	2	5310	44450	24450	8150	0,65
	3	8630	31050	21050	10525	0,82
	4	/				
Gesamt	4	20120				

Abbildung 3: Level 19

- *Max. Punktzahl*: Maximal zu erreichende Punktzahl des gesamten Levels bevor der Schuss ausgeführt wurde
- *Max. Durchschnitt*: maximal zu erreichender Durchschnittswert pro Schuss bezogen auf dem momentanen Zustand des Levels

Da unserer Meinung nach alle Schüsse in diesem Level gut waren und wir den zweiten Schuss noch an der Grenze zu „gut“ sahen, stellten wir folgende Regel auf: der Schuss muss mind. 65% von dem maximal zu erreichenden Durchschnittswert abdecken, um als „guter Schuss“ zu gelten.

Mit der selben Strategie sind wir auch in Bezug auf den zweiten Aspekt – Anzahl der verwendeten Vögel – vorgegangen und haben einige Szenarien durchgespielt, z.B. war es hervorragend, wenn der Agent höchstens 3 von 4, 3-4 von 5, 4 von 6, 5 von 7, 6 von 8 etc. Vögel verwendet hat. Mit diesen Vergleichen kamen wir dann auf einen Mittelwert von 75%, d.h. wenn der Agent maximal 75% der Verfügung gestellten Vögel verbraucht, gilt er als gut. Somit gilt: Je kleiner der Prozentsatz an verwendeten Vögeln ist, desto besser ist die Gesamtbewertung. Daher muss mit dem Wert $Wert2 = 1 - ratio$ gerechnet werden, um eine proportionale Gewichtung zu ermöglichen.

Um diese beiden Werte dann in einem gemeinsamen Grenzwert zusammenfassen zu können, muss das Gesamtbild in Betracht gezogen werden. Abbildung 4 veranschaulicht alle Werte in einer Tabelle mit der Formel $0.7 * Wert1 + 0.3 * Wert2$, wobei $Wert1$ das Verhältnis der Punktzahl des einzelnen Schusses (erster Aspekt) und $Wert2$ das Verhältnis für die Anzahl der Vögel (zweiter Aspekt) darstellt :

		0,3										
		0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
	0	0	0,03	0,06	0,09	0,12	0,15	0,18	0,21	0,24	0,27	0,3
	0,1	0,07	0,1	0,13	0,16	0,19	0,22	0,25	0,28	0,31	0,34	0,37
	0,2	0,14	0,17	0,2	0,23	0,26	0,29	0,32	0,35	0,38	0,41	0,44
	0,3	0,21	0,24	0,27	0,3	0,33	0,36	0,39	0,42	0,45	0,48	0,51
0,7	0,4	0,28	0,31	0,34	0,37	0,4	0,43	0,46	0,49	0,52	0,55	0,58
	0,5	0,35	0,38	0,41	0,44	0,47	0,5	0,53	0,56	0,59	0,62	0,65
	0,6	0,42	0,45	0,48	0,51	0,54	0,57	0,6	0,63	0,66	0,69	0,72
	0,7	0,49	0,52	0,55	0,58	0,61	0,64	0,67	0,7	0,73	0,76	0,79
	0,8	0,56	0,59	0,62	0,65	0,68	0,71	0,74	0,77	0,8	0,83	0,86
	0,9	0,63	0,66	0,69	0,72	0,75	0,78	0,81	0,84	0,87	0,9	0,93
	1	0,7	0,73	0,76	0,79	0,82	0,85	0,88	0,91	0,94	0,97	1

Abbildung 4: Evaluationswerte

Die erste Bedingung ist, dass $Wert1 > 0.65$ sein muss, damit die Gesamtbewertung gut ist, auch wenn alle Vögel verwendet wurden. Daraus folgt: $0.7 * 0.65 + 0.3 * 0 = 0.455$. Diese Grenze lässt sich auch gut auf andere Fälle anwenden, was auch anhand der Tabelle 4 zu sehen ist. Beispielsweise wenn der Agent 80% aller Vögel verwendet hat, muss der betrachtete Schuss in der vorigen Runde mind. 60% von der maximal zu erreichenden Durchschnittspunktzahl abgedeckt haben, um noch als `GOOD` markiert zu werden – $0.7 * 0.6 + 0.3 * 0.2 = 0.48$. Andernfalls gilt er als `BAD`.
Ein weiteres Beispiel zeigt, wenn der Agent bei dem aktuellen Level zuvor nur 2 von 5 Vögeln benötigt hat, d.h. $Wert2 = 1 - 2/5 = 0.6$, muss die erreichte Punktzahl des Schusses nur 40% von dem maximalen Durchschnittswert betragen, um die Grenze zum guten Schuss zu überschreiten.
Mit etlichen Szenarien waren wir mit dem Gesamtbild zufrieden, weshalb wir die Grenze bei 0.45 bestehen ließen, da es ebenfalls die besten Resultate hervorbrachte.

5 Schluss

5.1 Auswertung des Angry Birds AI Wettbewerbs 2017

Der IJCAI 2017 Angry Birds AI Wettbewerb, an dem das Team BamBird dieses Jahr als Titelverteidiger teilnahm, wurde bereits am 24.08.2017 ausgetragen. Zu unserer Enttäuschung mussten wir den Titel leider weitergeben, da der Agent in einem Level hängen geblieben ist. Dies hat ihm leider so viel Zeit gekostet, dass es ihm nicht gelungen ist, die anderen Agenten wieder einzuholen.[4] Ob es daran lag, dass keine passende Strategie gefunden wurde oder ob es immer wieder das gleiche Level ausgewählt hat, konnten wir leider nicht in Erfahrung bringen.

5.2 Vergleich zu letztem Jahr

Der signifikanteste Unterschied zwischen unserer Vorgehensweise und der aus dem letzten Jahr ist die Verwendung der Datenbank. Während das Team 2016 keinen Speicher verwendet hat und ihre Taktik auf Zufall basierte, bauen unsere Ideen zur Verbesserung und zum Lernen der Spieltaktik auf der Level-Storage auf. Dementsprechend fehlte dem Team aus letztem Jahr auch eine Schussauswahl.
Am meisten ähnelt sich jedoch die Levelauswahl, wobei wir ihre Idee für die Grenze, die wir für die Anzahl an Wiederholungsversuchen zulassen, auch in unsere Gedankengänge eingebaut haben (siehe Kapitel 8.3 - „A case study on the level selection functionality“ aus dem Projektbericht 2016).

5.3 Verbesserungsmöglichkeiten

5.3.1 Datenbank

Derzeit verwendet die `LevelStorage` eine Map und eine separate Liste, die die Level-IDs umfasst. Es wäre übersichtlicher eine einzige Liste oder Map zu haben und diese als Einheit zu verwenden.
Außerdem könnte der Speicher auch erweitert werden. In jedem Fall verbessert werden sollte, dass mehrere Listen bzw. eine große Liste von allen Schüssen angelegt wird. In der jetzigen Implementierung wird die Schussliste eines Levels nach dem Wiederholen desselben Levels lediglich überschrieben, d.h. man verliert die Information der Schüsse aus dem ersten Durchgang.
Zusätzlich könnten noch andere Merkmale gespeichert werden, abhängig von den anderen Gruppen, z.B. Ergebnisse der Physiksimulation. Eine sinnvolle Idee wäre es auch,

[4] `https://aibirds.org/angry-birds-ai-competition/competition-results.html` (zuletzt abgerufen: 13.09.2017)

komplette Levelszenen zu speichern, also die Strukturen und ihre Reihenfolge, Anzahl der Vögel und Anzahl der Schweine. Somit könnten die einzelnen Level genauer analysiert und evaluiert werden.

5.3.2 Levelauswahl

Bei der Levelauswahl könnte man eine andere Formel erarbeiten, um die Wahrscheinlichkeit für ein Level zu berechnen, sowie andere Komponenten mit einbeziehen. Ebenfalls könnten andere Grenzen gesetzt werden, wie oft ein Level gespielt werden darf und die Reihenfolge nicht nur abhängig von der erreichten Punktzahl machen, sondern auch auf die verbliebene Zeit und das Ausmaß des Levelumfangs, sodass man ab einer bestimmten Zeit beispielsweise nur noch kleinere Level spielt, um noch so viele Level spielen zu können wie möglich.

5.3.3 Schussauswahl und Evaluation

Für die Schussauswahl könnte das Meta-Team, falls die Physiksimulation hinzukommt, eigene Konfidenzen für die übergebenen Pläne errechnen, anstatt sich allein auf das Strategie-Team zu verlassen.
Derzeit erfolgt die Schussevaluation über den Vergleich von Schüssen, die im selben Level an der gleichen Stelle ausgewählt werden. Der erste Durchgang aller Level erfolgt somit ohne Evaluation. Es wäre sinnvoll einen Weg zu finden Schüssel über Level hinaus zu vergleichen. Ein Ansatz hierfür wäre die Erweiterung der Datenbank um das Speichern von ganzen Welten. Dadurch können die Schüsse bereits beim ersten Durchgang verglichen werden, z.B. besitzt Level 1 eine ähnliche Struktur wie Level 4, könnte man sich den Schuss auf diese Struktur aus Level 1 zur Hilfe für Level 4 nehmen.

5.4 Fazit

Letztendlich sind wir dennoch zum dem Entschluss gekommen, dass der Agent, trotz des Versagens im Wettbewerb, gelungen ist und nach der Umstrukturierung eine gute Basis zu einem hervorragenden Agenten besitzt. In den nächsten Semestern kann der Fokus dann mehr auf maschinelles Lernen gelegt werden und die angesprochenen Probleme noch hinreichender berarbeitet werden.

Literatur

[1] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-hex: an artificial player for angry birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):128–139, 2016.

[2] Ruofei Du, Zebao Gao, and Zheng Xu. Deliberately planning and acting for angry birds with refinement methods.

[3] Leonardo Anjoletto Ferreira, Guilherme Alberto Wachs Lopes, and Paulo Eduardo Santos. Combining qualitative spatial representation utility function and decision making under uncertainty on the angry birds domain. In *Proceeding of the International Joint Conference on Artificial Intelligence. Beijing, China*, 2013.

[4] Jochen Renz et al. Aibirds: The angry birds artificial intelligence competition. In *AAAI*, pages 4326–4327, 2015.

[5] Jochen Renz, Xiaoyu Ge, Stephen Gould, and Peng Zhang. The angry birds ai competition. *AI Magazine*, 36(2):85–87, 2015.