Informaticup 2021 Dokumentation

Grundei Moritz, Schmuhalek Martin

January 2021

Abgabetermin: 17.01.2021

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Problembeschreibung

Die Aufgabenstellung des Informaticups 2021 ist es, für das Spiel `Spe_ed` ein Programm zu entwickeln, welches möglichst viele Spiele gegen die Lösungen der anderen Teilnehmer gewinnt. Aufgrund der hohen Anzahl an möglichen Spielverläufen ist es dabei nicht möglich, komplette Spielbäume auszurechnen und damit den optimalen Zug für jede Spielsituation herauszufinden.

2 Lösungsansätze

Für die Lösung des beschriebenen Problems sind verschiedene Ansätze denkbar. Im Folgenden werden die verwendeten und implementierten Ansätze kurz beschrieben.

2.1 Zufälliger Algorithmus

Die Idee dieses Algorithmus ist es, einfach zufällig immer einen der verfügbaren Züge zu wählen. Dieser Ansatz dient eher dem Testen als der Darstellung eines ernsthaften Lösungsansatzes. Jedoch sollte jeder der weiteren Ansätze zumindest besser sein als der zufällige Algorithmus.

2.2 "Passiver" Algorithmus

Dieser Ansatz stellt eine Weiterentwicklung des zufälligen Algorithmus dar. Vor jedem Zug werden hierbei alle möglichen Züge - ohne Beachtung der Züge der anderen Spieler - simuliert und jeder Zug, der zum direkten Ausscheiden führen würde, wird ausgeschlossen. Daraufhin wird zufällig einer der verbleibenden Züge ausgewählt.

2.3 Heuristischer Algorithmus

Bei diesem Ansatz handelt es sich um eine Weiterentwicklung des "Passiven" Algorithmus. Dafür wird weiterhin jeder mögliche Zug simuliert und alle zum direkten Ausscheiden führenden Züge entfernt. Daraufhin wird für jeden Zug eine Bewertung erstellt. Für die Bewertung werden verschiedene Kriterien ausgewählt. Als beste Bewertung hat sich dabei eine Kombination der Anzahl der freien Felder in der Umgebung des Zielfeldes sowie der Entfernung zu den anderen Spielern ergeben.

2.4 Reinforcement Learning Algorithmus

Dieser Ansatz basiert auf einem Algorithmus der Reinforcement Learning Kategorie. Dabei handelt es sich um eine Unterklasse der Machine Learning Algorithmen. Dieser soll durch selbständiges Training möglichst gute Ergebnisse erzielen. Die Grundidee ist hierbei, dass auf Basis der Beobachtungen, die nach einem Zug gemacht werden, mithilfe einer Rewardfunktion eine Bewertung durchgeführt wird. Der Algorithmus wird dann daraufhin trainiert diesen Reward zu maximieren.

2.5 Graphen Algorithmus

Die Idee hinter diesem Ansatz ist es, das Spielfeld als Graphen darzustellen. Auf diesem Graphen soll dann mithilfe des Dijkstra Algorithmus der kürzeste Weg zu einem vor jedem Zug bestimmten Punkt auf dem Graphen gefunden werden. Um Punkte, die nicht erreichbar sind auszuschließen, wird vor der Bewertung die Zusammenhangskomponente berechnet, in der sich der Spieler aktuell befindet. Die Bewertung der übrigen Felder erfolgt dann ähnlich wie bei dem Heuristischen Algorithmus.

2.6 "Hybride" Algorithmen

Während der Implementierung der oben genannten Ansätze kamen verschiedenen Ideen auf, die Ansätze zu kombinieren. So wurde entstanden in der Hauptsache 2 "Hybride" Algorithmen.

2.6.1 Reinforcement + Heuristik

Hierfür wurde der Reinforcement Algorithmus mit den Heuristischen Algorithmus kombiniert. Die Idee hierbei ist es, dass der Reinforcement Algorithmus grundsätzlich die Auswahl der Züge übernimmt. Kommt es hierbei zu dem Fall, dass ein Zug gewählt wird, der zum Ausscheiden führt, wird der gewählte Zug von dem Zug des Heuristischen Algorithmus überschrieben.

2.6.2 Heuristik + Graph

Bei dieser Kombination übernimmt grundsätzlich der heuristsiche Algorithmus die Kontrolle über die Züge. Es werden jedoch vor jedem Zug die Zusammenhangskomponenten der Zielposition ermittelt. Sollten sich hierbei gravierende Unterschiede in der Größe der Zusammenhangskomponenten feststellen lassen, wird der Zug gewählt, welcher zur größtmöglichen Zusammenhangskomponente führt.

2.7 "Simpler" Algorithmus

Dieser Algorithmus entstand durch einen Programmierfehler bei der Implementierung des Hybriden Algorithmus "Heuristik + Graph". Durch einen Fehler wurden die Züge, die ausgewählt wurden, verändert. Dies führte dazu, dass ein Algorithmus entstand, welcher solange möglich, immer abwechselnd sich nach links und rechts dreht. Sollte das nicht mehr möglich sein, wird einer der übrigen Züge gewählt. Außerdem wählt auch dieser Algorithmus bei großen Unterschieden in den Zusammenhangskomponenten den Zug, welcher zur größeren Zusammenhangskomponente führt.

3 Implementierung

Zu Beginn wurde das Spiel `Spe_ed` so gut wie möglich nachgestellt. Dies ermöglicht das effizientere Testen der verschiedenen Ansätze. Außerdem wäre ohne eine lokale Version des Spiels das Trainieren des Reinforcement Algorithmus so gut wie unmöglich gewesen, da hierfür mehrere tausend Spiele trainiert werden muss.

3.1 Zufälliger Algorithmus

Die Implementierung des zufälligen Algorithmus ist denkbar simpel. Es wird lediglich aus einer Liste mit allen möglichen Zügen einer ausgewählt und durchgeführt. Wie bereits erwähnt, galt dieser Ansatz nur dem Testen des Spiels so wie der anderen Algorithmen. Aus Gründen der Vollständigkeit wird er hier trotzdem genannt.

3.2 "Passiver" Algorithmus

Für die Implementierung dieses Algorithmus ist es nötig, jeden der fünf möglichen Züge darauf zu überprüfen, ob dieser zum direkten Ausscheiden führt. Dafür wurde die aktuelle Position im Spielfeld je nach Zug angepasst und anschließend überprüft, ob die neue Position bereits belegt ist oder sich außerhalb des Spielfeldes befindet. Alle Züge, die aufgrund dieser Überprüfung als nicht möglich erkannt wurden, werden aus der Liste der Züge entfernt. Aus dieser Liste wird anschließend einer der übrigen Züge zufällig ausgewählt.

3.3 Heuristischer Algorithmus

Die Implementierung dieses Algorithmus verwendet für das Ausschließen von Zügen, die zum direkten Ausscheiden führen, denselben Mechanismus wie der vorhergehende "passive" Algorithmus. Erweitert wird dies dann durch Methoden, die für die neue Position eine Bewertung erstellen.

3.3.1 Bewertung der Zielfelder

Die Bewertung der Zielfelder wird anhand von 2 Metriken vorgenommen.
Die erste Metrik ist die Anzahl der freien Felder im Zielumfeld. Dafür werden die Felder in einem Umfeld von 5 * 5 um das Zielfeld ausgewertet. Die Summe der freien Felder wird dann normiert und als Bewertung für das Zielfeld verwendet. Hierbei gilt: je größer die Anzahl der freien Felder umso besser.
Die zweite Metrik zur Bewertung ist die durchschnittliche Distanz zu den anderen Spielern. Die Distanz entspricht der Standardnorm des Vektors, der zwischen den Positionen des eigenen Spielers und des betrachteten Gegners erstellt werden kann. Anschließend wird diese Distanz dann normiert. Zuletzt wird die Summe der normierten Distanzen gebildet und durch die Anzahl der noch aktiven Spieler mit - Ausnahme des eigenen - geteilt, um den Durchschnitt zu erhalten. Auch bei dieser Bewertungsmetrik gilt: je größer der durchschnittliche Abstand, umso

besser. Des weiteren wird der geringste Abstand zum Rand des Spielfeldes in die Bewertung eingerechnet.

3.3.2 Auswahl des Zuges

Die vorhandenen Metriken werden anschließend gewichtet und summiert. Der Zug, welcher die größte Gesamtbewertung hat und nicht zum direkten Ausscheiden führt, wird dann durchgeführt. Für die Gewichtung wurden verschiedene Werte ausprobiert und mittels Optimierungsverfahren versucht, eine möglichst gute Kombination zu finden. Dabei stellte sich jedoch heraus, dass verschiedene Gewichtungen wenig Einfluss auf die Ergebnisse hatten und somit die Gewichtung beider Metriken mit jeweils 1 ausreichend ist.

3.4 Reinforcement Learning Algorithmus

Der Reinforcment Learning Algorithmus basiert auf der Implementierung des Proximal Policy Optimization Algorithmus von OpenAI. Der Quellcode hierfür ist im Internet unter der MIT Lizenz verfügbar. Die Aufgabe war es dabei, den Algorithmus anzupassen und für die gewünschte Umgebung nutzbar zu machen. Um dies zu erreichen musste die Trainingsumgebung, welche in der Implementierung vorhanden war, durch die Spielumgebung ausgetauscht werden. Dafür wurde eine Schnittstelle erstellt, welche die Beobachtungen aus dem aktuellen Spielstand aufbereitet, so dass der Algorithmus diese verwenden kann. Des Weiteren wurde eine Reihe von verschiedenen Rewardfunktionen erstellt, welche sich voneinander unterscheiden, um verschiedene Rewards geben zu können.
Für das Training des Algorithmus wurde eine Umgebung erstellt, welche eine variable Anzahl an Spielen durchführt, in denen der Reinforcement Learning Algorithmus gegen die anderen Algorithmen antritt. Am Ende der durchgeführten Spiele wird dann die Anzahl der gewonnenen Spiele ausgegeben.

3.5 Graphen Algorithmus

Der Graphen Algorithmus basiert auf den Dijkstra Algorithmus zur Wegfindung. Hierfür wird zunächst das Spielfeld als Graph erstellt. Dabei wird für jedes Feld in einem vorher festgelegtem Abstand zum Ausgangsfeld des Spielers ein Knoten erzeugt. Als Nächstes werden die Kanten zwischen den Knoten erzeugt. Ist ein Knoten entweder durch den eigenen Spieler, einen anderen Spieler oder weil er sich außerhalb des Spielfeldes befindet nicht zu überqueren, wird keine Kante zu ihm oder von ihm erzeugt. Innerhalb dieses Graphs wird dann die Zusammenhangskomponente, in der sich der Spieler aktuell befindet, gebildet. In dieser Zusammenhangskomponente wird dann auf Basis der freien Felder im Umfeld des Ziels entschieden, welches Feld das "optimale" Feld ist. Zu diesem Feld wird dann der kürzeste Weg mittels des Dijkstra Algorithmus gefunden. Von dem dabei entstehenden Pfad wird dann der erste Zug durchgeführt. Die gesamte Prozedur wird bei jedem Zug wiederholt, um auf Veränderungen im Spielfeld reagieren zu können.

3.6 Hybride Algorithmen

3.6.1 Reinforcement + Heuristik

Für diesen Ansatz wurden der Reinforcement Ansatz und der heuristische Ansatz verbunden. Es wurde dabei auf die bereits vorhandenen Implementierungen zurückgegriffen. Um dies zu realisieren wird bei jedem Zug, der durch den Reinforcement Algorithmus gewählt wird, mithilfe des heuristischen Algorithmus überprüft, ob dieser zum direkten Ausscheiden führt oder nicht. Sollte der Fall eintreten, dass der gewählte Zug dabei vom heuristischen Algorithmus als nicht möglich bewertet wird, ersetzt der Zug des heuristischen Algorithmus den des Reinforcement Algorithmus. Dies wird im Reward mit einer negativen Wertung für den Reinforcement Algorithmus bestraft.

3.6.2 Heuristik + Graph

Dieser Ansatz verbindet die Auswahl der Züge durch den heuristischen Algorithmus mit der Erkennung von Zusammenhangskomponenten aus dem reinen Graphen Ansatz. Dabei soll der heuristische Algorithmus grundsätzlich die Züge auswählen. Für jeden möglichen Zug werden dabei die daraus resultierenden Zusammenhangskomponenten berechnet. Sollte dabei die Größe der Zusammenhangskomponenten von verschiedenen Zügen signifikant voneinander abweichen, wird der Zug gewählt, welcher den Spieler in die größte Zusammenhangskomponente bringt.

3.7 "Simpler" Algorithmus

Wie in der Beschreibung des Ansatzes bereits erwähnt, basiert dieser Algorithmus auf einem Fehler in der Implementierung des Hybriden Algorithmus "Heuristik + Graph". Dieser Fehler führte dazu, dass die eigentlich ausgewählte Aktion überschrieben wurde. Dabei wird aus einer Liste der möglichen Züge immer mit der Reihenfolge "`change_nothing`", "`turn_left`" und "`turn_right`" der letzte Zug gewählt, welcher noch möglich ist. Eine Ausnahme wird dann gemacht, wenn einer der Züge dazu führt, dass eine deutlich kleinere Zusammenhangskomponente betreten wird. Dann wird der Zug gewählt, welcher in die größte Zusammenhangskomponente führt.

4 Bewertung der Ansätze

Im Folgenden wird eine Einschätzung der verschiedenen Ansätze abgegeben. Dabei war die für die Bewertung ausschlaggebende Metrik die Anzahl der Züge im Spiel gegen sich selbst, die anderen Ansätze und im Test auf dem zur Verfügung stehenden Server gegen andere Teams. Außerdem war beim Testen festzustellen, dass die Performance eigentlich aller Ansätze dadurch verbessert wird, wenn die Möglichkeit, die Geschwindigkeit zu ändern, ignoriert wurde.

4.1 Zufälliger Algorithmus

Wie zu erwarten ist die Performance dieses Ansatzes am schlechtesten. Jedoch variiert die Performance in einzelnen Runden teilweise sehr stark. Insgesamt ist das mit recht deutlichem Abstand der schlechteste Ansatz, was jedoch so auch zu erwarten war.

4.2 "Passiver" Algorithmus

Dieser Algorithmus ist im Vergleich mit dem zufälligen Ansatz schon deutlich besser und überlebt einige Runden länger. Jedoch ist die Performance auch bei diesem Ansatz weit davon entfernt wünschenswert zu sein. Eine häufige Ursache für das Ausscheiden war hierbei das Einschließen in selbsterstellten Teilen des Spielfeldes.

4.3 Heuristischer Algorithmus

Dieser Algorithmus ist der erste Algorithmus, der es im Online-Spiel gegen andere Teams geschafft hat in einigen Fällen zu gewinnen. Des Weiteren ist die Gewinnrate dieses Algorithmus gegen die vorher genannten Ansätze nahezu hundert Prozent. Nach einigem Ausprobieren und Testen mit verschiedenen Bewertungen und Gewichtungen, konnten in einem Spielfeld mit Größe 70 x 70 teilweise auch Spiellängen im Bereich von 300 - 500 Zügen erreicht werden, was eine Verbesserung um mindestens den Faktor 10 gegenüber der vorherigen Ansätze darstellt. Somit wurde dieser Ansatz als erste einigermaßen zufriedenstellende Lösung akzeptiert.

4.4 Reinforcement Algorithmus

Die Bewertung dieses Algorithmus erfolgte ausschließlich auf Spielen gegen die anderen Algorithmen und nicht auf Online-Spielen, da für das Training eine sehr hohe Anzahl an Spielen nötig ist. Die Ergebnisse, die während des Trainings erzielt wurden, waren oft sehr unzufriedenstellend. Die erste Implementierung verwendete dabei einen Algorithmus der im Vergleich zu dem am Ende verwendeten PPO-Algorithmus deutlich simpler war. Die Umstellung zum PPO Algorithmus verbesserte das Ergebnis deutlich. Davor erlernte der Algorithmus oft einfach nur "geradeaus zu laufen" oder sich solange zu drehen, dass er einen Kreis schloss und somit verloren hatte. Dies konnte mir der Umstellung auf den PPO Algorithmus verringert werden. Jedoch konnten auch damit keine Ergebnisse erzielt werden, die eine gute Siegesrate gegen den heuristischen Algorithmus aufweisen konnten.

Hierbei sollte allerdings auch gesagt werden, dass diese enttäuschenden Ergebnisse auch daran lagen, dass einiges wichtiges Wissen über die Funktionsweise solcher Algorithmen fehlt und weniger an der Eignung solcher Algorithmen für die zu lösende Aufgabe. So gelang es zum Beispiel nicht, dem Algorithmus aufgrund der verwendeten Implementierung, das gesamte Spielfeld als Input zu geben, sondern nur einzelne Metriken, welche mit denen des heuristischen Algorithmus vergleichbar sind.

4.5 Graphen Algorithmus

Die Performance dieses Ansatzes kann sehr stark variieren. Im Durchschnitt bleibt sie jedoch hinter dem heuristischen Ansatz zurück. Oft passiert es dabei, dass andere Spieler sich so bewegen, dass dieser Algorithmus keine Möglichkeit mehr hat, eine große Anzahl an Zügen durchzuführen. Jedoch wird der vorhandene Platz meist noch sehr effizient genutzt.

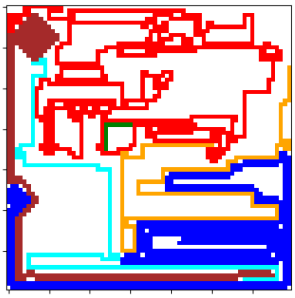

Figure 1: Typisches Muster des Graphen Players (Gelb und Hellblau)

4.6 Hybride Ansätze

4.6.1 Reinforcement + Heuristik

Die Verbindung des Reinforcement Algorithmus mit dem Heuristischen Algorithmus führte dazu, dass viele Züge vom Heuristischen Algorithmus bestimmt wurden. Jedoch war die Performance hier etwas schlechter als beim "alleinig" heuristischen Algorithmus. Ein Problem war die Bewertung von Zügen, welche durch den heuristischen Algorithmus gewählt wurden, in der Berechnung für den Reward des Reinforcement Algorithmus. Somit ist auch bei diesem Ansatz davon auszugehen, dass der Ansatz, bei besserer Umsetzung, zu besseren Ergebnissen kommen könnte.

4.6.2 Heuristik + Graph

Bei diesem Hybriden Ansatz konnte eine Verbesserung des heuristischen Algorithmus mithilfe von Teilen des auf Graphen basierenden Algorithmus erzielt werden. Dabei war festzustellen dass die durchschnittliche Spieldauer leicht, sowie die Gewinnrate in Onlinespielen gestiegen war.

Verantwortlich dafür dürfte die Tatsache sein, dass der Algorithmus die größeren Zusammenhangskomponenten wählt. Dies führt dazu, dass die Zeit bevor der Algorithmus ein Feld wählt auf dem bereits ein anderer Spieler oder er selbst war, gestiegen ist. Die Platzeffizienz dieses Algorithmus lässt jedoch zu wünschen übrig. Außerdem kann auch bei diesem Algorithmus die Performance sehr stark schwanken. Im Durchschnitt ist sie jedoch als gut zu beurteilen. Dieser Algorithmus konnte regelmäßig im Test gegen die anderen Algorithmen sowie im Onlinespiel gewinnen.

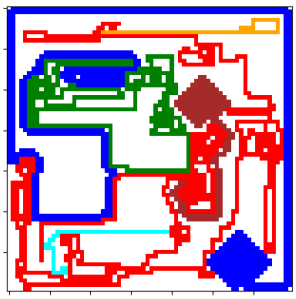

Figure 2: Typisches Muster des Heuristik + Graph Players (Rot und Grün)

4.7 "Simpler" Algorithmus

Dieser Algorithmus performt trotz des recht simplen Verhaltensmusters besonders im Spiel gegen sich selbst gut. Dabei wird fast das gesamte Spielfeld ausgefüllt. Etwas anders sieht das beim Spiel gegen andere Algorithmen aus. Hierbei passiert es oft, dass der Algorithmus sich in engen Stellen des Spielfelds einsperrt. Jedoch ist auch hier die Performance die dieser Algorithmus erzielt insgesamt gut. So gewinnt er oft gegen die anderen hier vorgestellten Ansätze und auch im Onlinespiel.

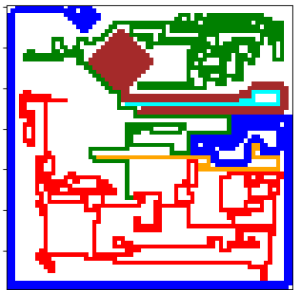

Figure 3: Typisches Muster des "Simple" Players (Braun und Dunkelblau)

4.8 Finale Bewertung und Auswahl der Lösung

Für die finale Bewertung wurden 3 Ansätze gewählt. In diese Auswahl kamen der "Graphen" Algorithmus, der "Heuristik + Graphen" Algorithmus und der "Simple" Algorithmus. Die finale Auswahl erfolgte auf Grundlage von ca. 200 Spielen der 3 Algorithmen gegeneinander. Dabei wurden jeweils 2 Algorithmen jeder Art verwendet und die jeweiligen Parameter leicht verändert. Hierbei zeigte sich dass der "Simple" Algorithmus am häufigsten gewinnt. Deshalb wurde dieser als Lösung ausgewählt.

5 Allgemeines Vorgehen bei der Entwicklung

Bevor die eigentliche Entwicklung der Lösungen begonnen wurde, wurde versucht, das Spiel auf dem Online-Server so exakt wie möglich nachzustellen. Großer Wert wurde dabei darauf gelegt, die vom Server bereitgestellten JSON Logs so exakt wie möglich darzustellen, um den Aufwand für die Umstellung vom lokalen Testspiel auf den Online-Server so einfach wie möglich zu gestalten. Dies hatte zum Ergebnis, dass eigentlich gar keine Anpassungen mehr nötig sind, um die Umstellung zu realisieren.

```
INFO:root:0
INFO:root:ID:1 x:1 y:7 speed:1 direction:up active:True
INFO:root:ID:2 x:37 y:48 speed:1 direction:up active:True
INFO:root:ID:3 x:31 y:19 speed:1 direction:left active:True
INFO:root:ID:4 x:39 y:46 speed:1 direction:left active:True
INFO:root:ID:5 x:45 y:19 speed:1 direction:left active:True
INFO:root:ID:6 x:24 y:21 speed:1 direction:right active:True
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
INFO:root:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 4: Ausschnitt einer Log Datei

Des Weiteren wurde darauf geachtet, die Koppelung des Spiels mit den Lösungs-Algorithmen so lose wie möglich zu gestalten, um das einfache Austauschen der verschiedenen Algorithmen zu erleichtern. Dafür wurde eine Schnittstelle entwickelt, welche dafür zuständig ist, die gewählte Aktion eines Spielers entgegenzunehmen und dem Spieler den aktuellen Log im JSON Format zur Verfügung zu stellen.

Danach wurden die verschiedenen Lösungsansätze implementiert. Diese erfolgte hauptsächlich in Form von "Pair Programming". Dabei geschieht die Programmierung zu zweit. Einer der Entwickler schreibt den Programmcode, während der Andere diesen gleichzeitig überprüft und mögliche Verbesserungen vorschlägt. Hierbei wechseln die Rollen der Entwickler oft.

Die verschiedenen Ansätze wurden dabei in der Reihenfolge entwickelt, in welcher sie im vorherigen Teil genannt werden. Diese Reihenfolge ergab sich zu Beginn vor allem daraus, dass die ersten Ansätze aufeinander aufbauen. Daher wurde auch von Anfang an darauf geachtet eine gute Wiederverwendbarkeit zu gewährleisten. Der erste Bruch dieses Musters ergab sich mit dem Reinforcement Algorithmus. Dieser baut nur zu sehr geringen Teilen auf den vorherigen Ansätzen auf. Insgesamt wurde sehr viel Zeit mit diesem Ansatz verbracht.

Die beiden hybriden Ansätze entstanden durch Überlegungen und Feststellungen, welche während der Entwicklung der zugrundeliegenden Algorithmen gemacht wurden. Die Verbindung der Al-

gorithmen war dann aufgrund der zuvor festgelegten Strukturen relativ simpel.
Während der gesamten Entwicklung wurde darauf geachtet, den Code möglichst lesbar zu halten und an den nötigen Stellen mit Kommentaren zu ergänzen. Außerdem wurde auch darauf geachtet, besonders nach dem Abschließen größerer Teile des Projekts, den Code nochmals zu refactoren um eine gute Lesbarkeit zu gewährleisten.

6 Fazit

Insgesamt konnten während der Entwicklung viele neue Kenntnisse gesammelt werden. Besonders hervorzuheben sind hierbei Kenntnisse in der Entwicklung mit Python sowie im Bereich des Reinforcement Learnings. Trotz der etwas enttäuschenden Ergebnisse des Reinforcement Ansatzes wurde viel Zeit damit verbracht, diesen zu verbessern und die Performance zu steigern. Im Vorfeld waren dafür nur sehr grundsätzliche Kenntnisse vorhanden. Jedoch konnten im Laufe des Projekts "Hands-On" Erfahrungen mit dem auf den ersten Blick sehr komplexen Thema gesammelt werden.
Außerdem konnten Kenntnisse in der eigenständigen Durchführung von Projekten gesammelt werden. Hierfür hat es sehr geholfen, ein Projekt mit einer konkreten Abgabefrist zu haben. Dies motivierte sehr stark dazu, an dem Projekt zu arbeiten, um dieses auch rechtzeitig abgeben zu können.

7 Weitere Überlegungen

Es wurden einige Überlegungen getätigt, die nicht umgesetzt werden konnten. Diese sollen hier jedoch zumindest kurz erwähnt werden.
Während der Implementierung des Reinforcement Algorithmus war eine Überlegung, eine Art Turnier zu erstellen, in dem jeweils leicht abgeänderte Versionen des Algorithmus gegeneinander antreten sollten um die beste Variante zu finden. Hierbei wäre sehr interessant zu beobachten gewesen, ob die verschiedenen Varianten nach öfterem Spiel gegen- bzw. miteinander Strategien entwickeln, die einem gegenseitigen "Unterstützen" ähnlich sind um den gesamten erhaltenen Reward zu verbessern. Dies konnte nicht durchgeführt werden, da die Implementierung des Algorithmus keine wünschenswerte Ergebnisse lieferte und somit die Zeit für die anderen Ansätze verwendet wurde.
Eine weitere Überlegung im Bezug auf den Graphen Algorithmus war es, einen "aggressiven" Algorithmus zu erstellen, welcher den Graphen des Spielfeldes möglichst in immer kleiner werdende Zusammenhangskomponenten teilt. Dies sollte die Bewegungen der anderen Spieler so einschränken, dass diese früher ausscheiden. Leider kam dieser Gedanke erst gegen Ende der Abgabefrist auf, so dass er nicht realisiert wurde. Außerdem ist das Erstellen von Trennern in einem Graphen recht komplex, womit auch Bedenken im Bezug auf die Laufzeit aufkamen.

8 Quellen

Sämtliche Bilder stammen aus eigenen Quellen

Die Implementierung des PPO Algorithmus stammt von : `https://github.com/openai/spinningup`
Die Implementierung steht unter der MIT License zur Verfügung