CARL VON OSSIETZKY universität OLDENBURG

InformatiCup 2021

Ein intelligenter Agent für das Spiel Spe_ed

Team LightningMcSpe_ed

Denis Groeneveld

Keno Oelrichs García

Max Stargardt

Januar 2021

Inhaltsverzeichnis

1 Einleitung

Wir sind die Gruppe *LightningMcSpe_ed* bestehend aus Studenten der *Carl von Ossietzky Universität* in Oldenburg. Uns wurde die Chance gegeben am InformatiCup2021 teilzunehmen ohne unser Studium dabei zu vernachlässigen, da eine Anrechnung eines adäquaten Informatikmoduls geboten wird. Unser Team besteht aus drei Mitgliedern, welche in der Fakultät II unserer Universität einen Informatik relevanten Studiengang besuchen:

- Denis Groeneveld (Wirtschaftsinformatik B.Sc.)
- Keno Oelrichs García (Informatik B.Sc.)
- Max Stargardt (Informatik B.Sc.)

Der InformatiCup 2021 bietet uns die Möglichkeit, zum ersten Mal in unserem Studium ein Projekt komplett selbstständig durchzuführen, ohne das Rücksprachen mit zuständigem Lehrpersonal gehalten werden müssen.

Unser selbst gesetztes Ziel, unabhängig vom Wettbewerb, ist es mehr über die Implementierung eines schlauen Agenten zu lernen und wie dieser mit möglichst einfachen Mitteln umgesetzt werden kann.

2 Projektanalyse

Die Projektanalyse beschreibt die Definition des Projektes vor Beginn der Umsetzung. Dabei schildern wir zunächst die Aufgabenstellung und gehen anschließend auf die Anforderungen ein.

2.1 Aufgabenstellung und Zielsetzung

Die Aufgabe umfasst die Entwicklung eines Agenten, der in der Lage ist, ohne menschliche Unterstützung das Spiel Spe_ed zu spielen. Dabei soll der Agent möglichst lange am Leben bleiben, indem er weder den Spielfeldrand überschreitet, noch mit bereits vorhandenen Spuren kollidiert. Hierbei ist das Ziel am längsten von allen teilnehmenden Spielern zu überleben.

2.2 Anforderungen

Vor jedem Zug erhalten die teilnehmenden Agenten über einen verschlüsselten Websocket eine Übersicht des aktuellen Spielfelds. Innerhalb von Zehn Sekunden muss nun jeder Agent eine Antwort an den Server zurücksenden. Diese besteht aus einer der vorgegebenen Aktionen: turn_left, turn_right, slow_down, speed_up oder change_nothing. Durch die gegebenen Aktionen ist nur eine horizontale und vertikale

Bewegung möglich. Sollte ein Agent keine oder mehrere Antworten an den Server zurückschicken, so scheidet dieser aus dem Spiel aus.

3 Projektorganisation

3.1 Organisation der Zusammenarbeit

Um unsere Arbeit zu koordinieren und zu synchronisieren müssen Richtlinien für die Zusammenarbeit gesetzt werden. So haben wir uns darauf geeinigt, unser Projekt in wöchentlichen Etappen mit verbundenen Treffen weiterzuentwickeln. Im Zuge der COVID-19 Pandemie ist dieses Treffen leider physisch nicht möglich. Zur Kommunikation wird *Discord* eingesetzt. Discord bietet neben einem Voice- und Text-Chat zusätzlich die Möglichkeit, kleinere Dateien schnell auszutauschen, sowie eine Bildschirmfreigabe von mehreren Teilnehmern gleichzeitig. Dies ersetzt in gewissem Maße die Praktik des Pair Programming.

3.2 Programmiersprache und weitere Tools

Wir sehen *Python* als eine ideale Programmiersprache für das Projekt des Spe_ed Agenten an, da keiner von uns bis zum Beginn des Projektes Berührungspunkte mit dieser Sprache hatte und dies so eine ausgesprochen gute Chance bietet, eine neue Programmiersprache zu lernen. Zudem bietet Python im Vergleich zu anderen Programmiersprachen eine einfach zu erlernende Syntax und leicht benutzbare Packages, welche unter anderem das Arbeiten und den Umgang mit neuralen Netzwerken vereinfachen soll. Auf die Verwendung von neuralen Netzwerken möchten wir in Kapitel 4 noch näher eingehen.

Aufgrund dessen, dass Python die gewählte Projektsprache ist, benutzen wir als Entwicklungsumgebung ausschließlich *PyCharm* und *IntelliJ*, wovon Letzteres durch ein entsprechendes Python Plugin ergänzt wird. Diese Umgebungen erscheinen uns als äußert leistungsstark und angemessen für alle Unternehmungen. Zudem bietet *JetBrains*, der Hersteller beider vorher genannten Entwicklungsumgebungen, jeweils eine vollwertige Lizenz für Studenten an.

Um unsere Fortschritte synchron zu halten und über eine Versionierung zu verfügen, verwenden wir *GitLab*. Das verwendete Repository wird von der Universität Oldenburg auf den Uni-internen Servern gehostet.

4 Projektarchitektur

4.1 Visualisierung des Spielfeldes

Für die Entwicklung des Agenten ist eine durchgängig verfügbare Fassung des Spiels notwendig. Bei der Onlinevariante des Spiels müssen mindestens zwei Teilnehmer anwesend sein, um das Spiel starten zu können. Da dies sehr hinderlich ist, wird das Spiel für eine lokale Nutzung nachgebaut. Hierbei ist es wichtig darauf zu achten, dass alle Regeln des originalen Spiels richtig umgesetzt werden, da dies sonst die gesamte weitere Entwicklung behindern könnte. So muss neben der Kollisionserkennung auch in jedem sechsten Zug bei allen Spielern ein Sprung erfolgen, die eine höhere Geschwindigkeit als „2“ aufweisen. Die Länge der entstehende Lücke entspricht dann der Geschwindigkeit des Spielers minus Zwei.

Da vor jedem Zug neue Variablen vom Server im JSON Format bereit gestellt werden, müssen diese zunächst angefragt, herunterladen, decodiert und auf unser nachgebautes Spiel übertragen werden. Anschließend wird das Spielfeld mittels PyGame wie in Abbildung 1 visualisiert.

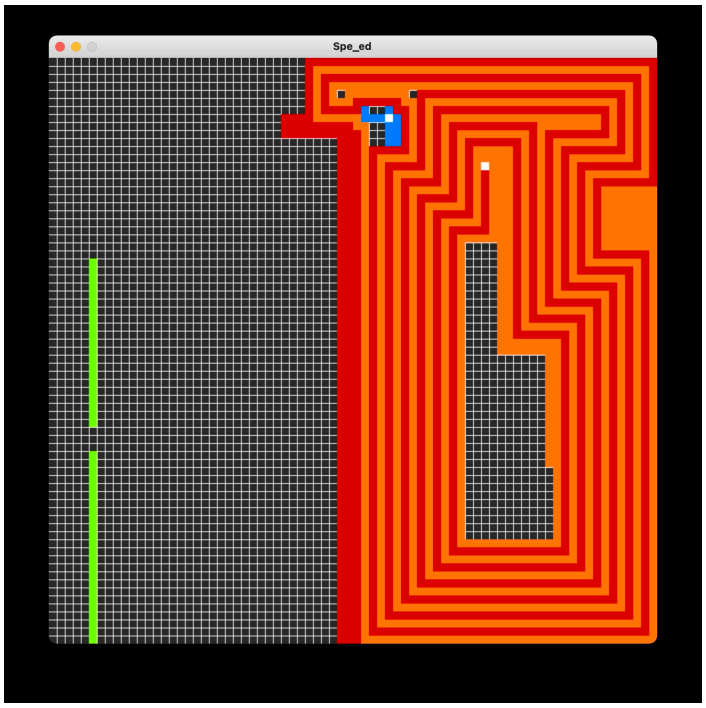


Abbildung 1: Unsere Visualisierung des Spielfeldes mittels PyGame

4.2 Entscheidungsprozess des Agenten

Der Agent durchläuft mehrere Entscheidungsprozesse, um eine optimale Entscheidung für den kommenden Zug zu fällen. Dabei berücksichtigen wir verschiedene Faktoren, auf die wir hier näher eingehen werden. Um diesen Entscheidungsprozess etwas anschaulicher zu gestalten, ist dieser in einem Programmablaufplan in Abbildung 2 dargestellt.

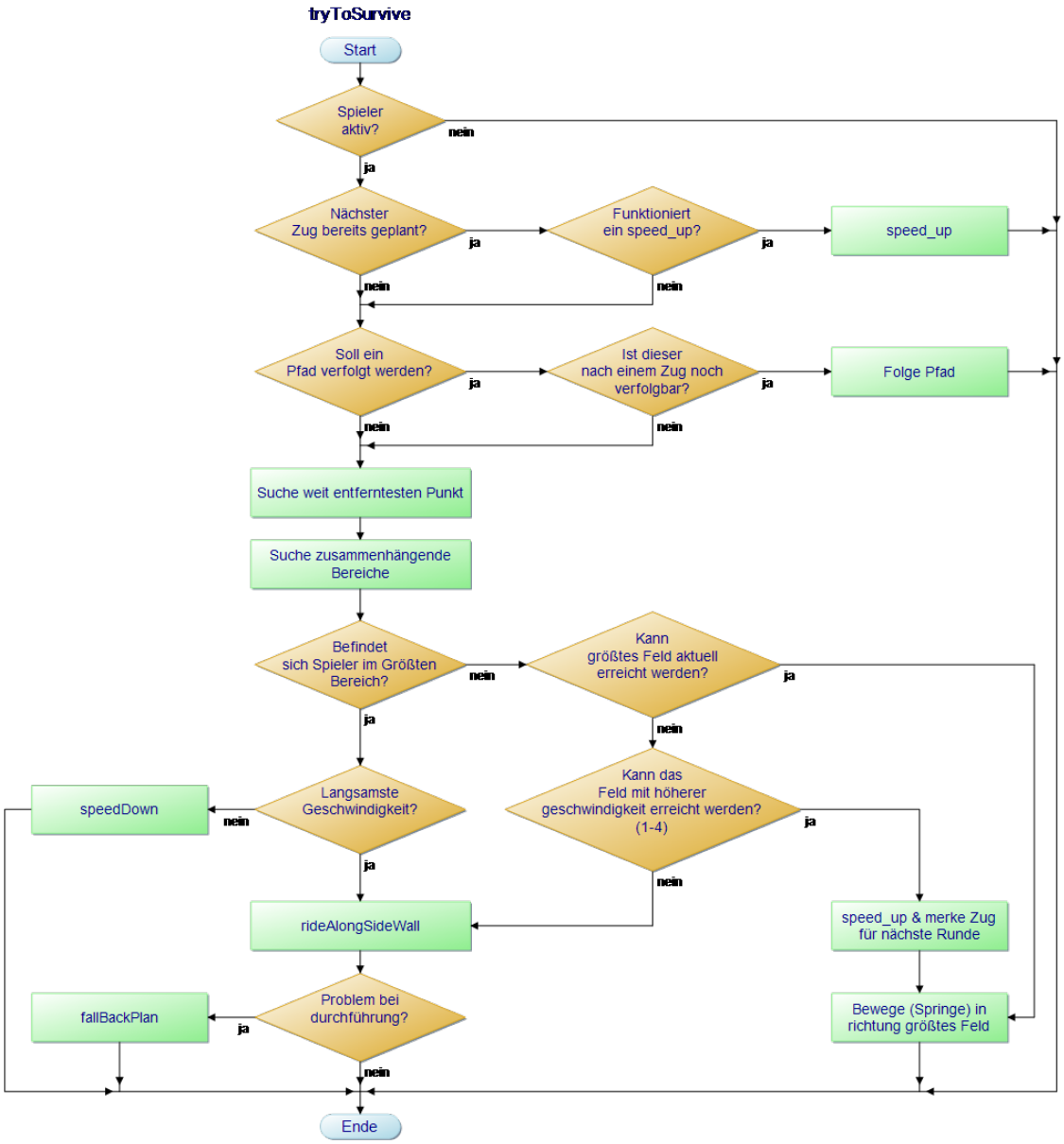


Abbildung 2: Der Entscheidungsprozess des Agenten als Programmablaufplan

Zuerst wird überprüft, ob der Agent noch im Spiel aktiv ist und Züge ausführen kann. Sollte der Agent aktiv sein, überprüft er, ob der nächste Zug bereits geplant ist und ob dieser mit den neuen Spielfeld-Variablen noch durchführbar ist. Bei den vorausgeplanten Zügen handelt es sich entweder um eine Erhöhung der Geschwindigkeit oder dem Folgen eines bestimmten Pfades. Anschließend wird der weit entfernteste erreichbare Punkt ermittelt, indem ähnlich wie beim A* Algorithmus, welcher später noch detaillierter beschrieben wird, ausgehend der aktuellen Position das Spielfeld mit Zählern versehen wird, wie viele Züge benötigt werden um ein Feld zu erreichen. Dabei wird berücksichtigt das alle sechs Züge ein Sprung erfolgt. Ebenfalls werden alle zusammenhängende Bereiche durch den FreeplaceFinder ermittelt.

4.2.1 Der FreeplaceFinder

Um herauszufinden, ob sich noch genügend freier Platz um den Agent herum befindet, bestimmt dieser alle zusammenhängenden Areale und deren Größe.
Dies wird erreicht, indem jeder Block des Spielfeldes auf eine bereits vorhandene Belegung durch ein Areal oder einen Spielers überprüft wird. Ist dies der Fall, so wird mit dem nächsten Block fortgefahren. Ist nun ein Block nicht belegt, so wird dieser von dem aktuellen Index des Areals belegt und alle angrenzenden Blöcke auf Belegung überprüft. Sind alle Blöcke überprüft, so erhält man am Ende ein Koordinatensystem in der Größe des Spielfeldes, in dem jeder freie Punkt einem Areal zugeordnet ist. Über die Anzahl eines Indizes auf diesem Koordinatensystem kann nun bestimmt werden, ob sich der Agent in dem größten Areal befindet oder nicht.

4.2.2 simulateNextTurn

Um in den einzelnen Methoden zu validieren, ob nun ein Zug sinnvoll ist oder nicht, wird dieser simuliert durchgeführt. In dieser Simulation wird als erstes der Zug, den der eigene Spieler durchführen soll, eingetragen. Von diesem Spielfeld wird anschließend eine Kopie erstellt und ermittelt, welche anderen Spieler sich in einem bestimmten Radius zu einem selbst befinden. Nun wird jede mögliche Kombination an Zügen, die diese Spieler durchführen könnten, simuliert durchgeführt. Hierbei wird ermittelt, ob der eigene Spieler durch diesen Zug ausscheiden würde. Alle Simulationen und die durch den Zug überlebten Kombinationen werden gezählt. Sollten mindestens 80 Prozent dieser Simulationen ohne das Ausscheiden des eigenen Spielers beendet werden können, so wird der zu testende Zug als sinnvoll angesehen. Die Methode gibt hierbei ein „True“ zurück. Sollten die 80 Prozent nicht erreicht worden sein, wird ein „False“ zurück gegeben.

4.2.3 Der A* Algorithmus

Bei dem A* Algorithmus handelt es sich um einen informierten Suchalgorithmus, welcher das erste Mal 1968 von Peter Hart, Nils J. Nilsson und Bertram Raphael beschrieben wurde [Luc]. Dieser dient in der Informatik zur Suche des kürzesten Pfades zwischen zwei Knoten innerhalb eines Graphen mit positiven Kantengewichten. Von uns wurde dieser Algorithmus dahingegen erweitert, sodass dieser nicht nur einen gültigen Pfad bei einer Geschwindigkeit von einem Block pro Iteration findet, sondern auch Geschwindigkeiten von bis zu zehn Blöcken pro Sekunde. Hierbei wird berücksichtigt, ob ein Sprung in der aktuellen Iteration möglich ist und welche anderen Blöcke hierdurch erreichbar wären. Als Ergebnis erhält man, wenn gefunden, einen Pfad von der angegebenen Start- zur End-Koordinate.

Der Agent verwendet diesen Algorithmus, um herauszufinden, ob es eine Möglichkeit gibt, von der aktuellen Position in das größte zusammenhängende Feld zu

gelangen, welches der FreePlaceFinder[4.2.1] ermittelt hat. Die Startposition ist hierbei immer die aktuelle Position des Agenten. Die Zielkoordinate wird hier auf den ersten Berührungspunkt mit dem größtmöglichen Zielareal festgelegt, um eine Platzverschwendung im neuen Bereich zu vermeiden. [Sar]

4.2.4 Die *rideAlongsideWall*-Methode

Um als letzter Überlebender auf dem Spielfeld zu stehen, gibt es genau zwei Optionen: Entweder wird aktiv für das Ableben der Kontrahenten gesorgt oder solange überlebt, bis alle Gegner nicht mehr aktiv sind. Diese Methode sorgt für letzteres. Ziel ist es dabei, dass der Agent sich der nächsten Wand nähert und sich entlang dieser in einer spiralförmiger Weise bewegt. Dabei ist es wichtig zu erkennen, ob der Agent mit seiner momentanen Geschwindigkeit überhaupt in der Lage ist entlang der Wand zu fahren, oder ob ein Abbremsen oder gar ein Ausweichen von Nöten ist. Diese beiden Szenarien werden wie folgt unterschieden:

- Ist der Spieler in Richtung nächster Wand unterwegs und hat genügend Platz für seinen nächsten Zug ohne zu sterben, so bremse ab
- Ist der Spieler in Richtung nächster Wand unterwegs und hat *nicht* genügend Platz für seinen nächsten Zug, so wechsle die Richtung zu der mit dem meisten freien Platz.

Sollte der Spieler sich nun entlang der Wand bewegen ist es zunächst entscheidend „Schlaglöcher“ oder eine ein-Zellen-breite Lücke wie in Abbildung 3 zu erkennen und abzuwägen ob es sich rentiert in diese Lücke zu gehen.

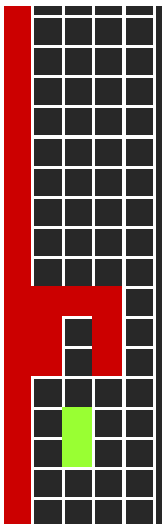
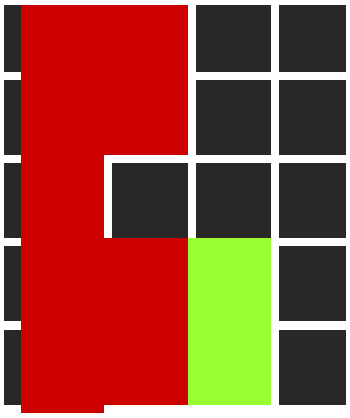


Abbildung 3: Ein „Schlagloch“ oder eine ein Zellen breite Lücke

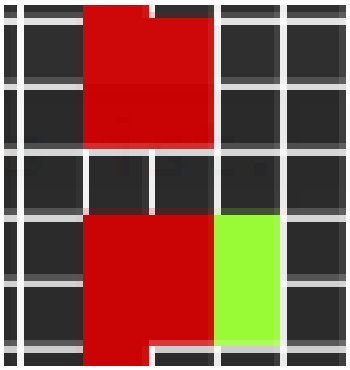
Dies erreicht der Agent, indem er seiner Geschwindigkeit entsprechend das nächste zu belegende Feld betrachtet und prüft, ob dieses von zwei oder mehr Wänden gleichzeitig umgeben ist. Ist dies der Fall, handelt es sich bei dem betrachteten Feld

entweder um eine Ecke, eine Wand oder der gesuchten ein-Zellen-breite Lücke. Um hier gleichzeitig zu prüfen, ob sich ein Befahren dieses Feldes lohnen würde, prüfen wir, ob das Areal hinter dieser potenziellen Lücke gleich groß oder größer ist als das Areal, welches sich vor der Lücke befindet. In dem Fall, dass das betrachtete Feld eine Ecke ist, sind die betrachteten Areale gleich groß und wir bewegen uns zur vermeintlichen Lücke. Somit sind wir nun in der Lage unseren verbleibenden Platz im Areal spiralförmig auszufüllen.

Eine letzte wichtige Gegebenheit, die noch zu beachten ist: Sobald eine Wand abknicken sollte und sich effektiv schräg hinter dem Spieler befindet, ist es nicht mehr möglich durch ein Prüfen der vier Kardinalrichtungen (Oben, Unten, Links, Rechts) zu ermitteln, ob die nächste Wand sich schräg hinter dem Spieler befindet. Daher kontrolliert der Agent, solange er sich nicht direkt neben einer Wand befindet, ob sich eine Wand schräg hinter ihm liegt und ob es sich lohnt in diese Richtung abzubiegen. „Lohnen“ ist in diesem Kontext so zu verstehen, dass auch hier wieder geprüft wird, ob die Lücke eine Sackgasse wie in Abbildung 4a ist, oder Teil des größten Areals wie in Abbildung 4b.



(a) Eine *nicht* lohnenden Lücke



(b) Eine lohnenden Lücke

Abbildung 4: Beispiele einer abknickenden Wand mit einer Lücke

4.2.5 Der Fallbackplan

Sollten alle Stricke reißen, sprich der rideAlongSideWall Algorithmus fehlschlagen, soll einfach nur versucht werden zu überleben. Beim Fallbackplan werden in alle Richtungen die freien Blöcke gezählt und anschließend in die Richtung der meisten freien Blöcke navigiert. Sollten weniger Blöcke frei sein als der Agent schnell ist, so bremst der Agent mit einem speedDown.

4.3 Ansatz eines Neural Networks anhand von NEAT

NEAT ist ein evolutionäres neurales Netzwerk, welches mit geringem Aufwand in Python verwendet werden kann [NEA]. Ziel dieses Netzwerkes ist, dem Bot ein Verhalten beizubringen, welches nicht unbedingt direkt erkannt werden kann, indem gutes Verhalten (überleben) belohnt und schlechtes Verhalten (sterben) bestraft wird. Diese Praktik nennt sich *reinforcement Learning*. Belohnungen und Bestrafungen äußern sich einzig dadurch, dass dem Bot ein gewisser *Score* zugewiesen wird, der sein Überleben darstellt, wobei ein hoher Score einem langen Überleben entspricht. Der Bot benötigt, um seine Entscheidungen zu treffen, gewisse Eingabewerte oder auch Fühler genannt. Zu diesen Eingabewerten zählen beispielsweise die eigene Geschwindigkeit, der Abstand zu Wänden in Kardinal Richtungen oder der Abstand zu anderen Spielern. Sobald der Bot alle vorher bestimmten Eingabewerte hat, kann dieser initial eine beliebige Entscheidung aus dem Entscheidungspool (Richtung oder Geschwindigkeit ändern) treffen. Hierbei sind die ersten Entscheidungen oftmals eher schlecht anzusehen und der Bot erreicht nur einen geringen Score. Jedoch kann der Bot durch hunderte Wiederholungen dieser Simulation aus seinen früheren Generationen lernen, was dementsprechend seine künftigen Entscheidungen beeinflusst. Jedenfalls ist unserer Gruppe klar geworden, dass ein adäquat funktionierendes evolutionäres neurales Netzwerk schwerer als gedacht umzusetzen und daher als Option für unser Projekt fallen zu lassen ist. Unsere Implementierung des NEAT Netzwerkes konnte selbst nach 10.000 Generationen kein brauchbares Ergebnis liefern, weswegen eine hart kodierte Strategie sinnvoller erscheint.

5 Projektabschluss

5.1 Fazit

Insgesamt blicken wir als Gruppe positiv auf das Projekt zurück. Die an uns gestellten Anforderungen für den Agenten wurden zu Projektbeginn klar definiert, was die Umsetzung des Projektes vereinfacht hat.

Eines der Ziele dieses Projektes war für uns die Aneignung der Programmiersprache Python. Durch einfaches Importieren und Nutzen externer Bibliotheken erscheint Python als Sprache sehr attraktiv und erleichtert an einigen Stellen das

Arbeiten. Da die an der Uni Oldenburg gelehrte Sprache Java ist und diese sich in vielerlei Hinsicht stark von Python unterscheidet, galt es anfängliche Hürden zu bewältigen. So besteht zu Projektbeginn neben der schnell erlernbaren Syntax von Python die Herausforderung der Objektorientierten Programmierung. Außerdem verwendet Python für Grafik Anwendungen traditionell nur einen Thread, der Logik und Grafik berechnet. Dies sorgte anfänglich für eine gewisse Verwirrung. Als Fazit lässt sich jedoch bereits jetzt feststellen, dass Python eine solide Programmiersprache für die Aufgabe des InformatiCups 2021 darstellt.

5.2 Auswertung

Die Leistung des Agenten wurde während des Projekts durchgängig getestet und die Strategie auf Basis der Ergebnisse angepasst. Im lokal reproduzierten Spiel sind mehrere Agenten mit derselben Strategie gegeneinander angetreten. Sobald an einer neuen Strategie gearbeitet wurde, musste sie sich gegen der vorherigen Strategie im Spiel behaupten. Ebenfalls wurde die aktuelle Strategie in der Online Variante von Spe_ed gegen andere Agenten getestet, sowie auf einem Uni-Oldenburg Server, der von einer Studentengruppe der Universität entwickelt und gehostet wurde.

Als Qualitätskriterien betrachten wir die ISO 9126 Standards zur Messung von Softwarequalität [Kry]. Dies sind bewehrte Standards zum Messen von Softwarequalität. In Tabelle 1 legen wir die Erfüllung dieser Kriterien dar.

Qualitätskriterien	
Betrachtetes Kriterium	Beschreibung
Funktionalität	Bis auf die Tatsache, das der Agent nicht 100% der Spiele gewinnt, erfüllt er die an ihn gestellten Anforderungen vollständig. Mögliche Fehler, werden abgefangen und verarbeitet, somit kann ein durch den Agenten verursachten Softwareabsturz nahezu ausgeschlossen werden.
Zuverlässigkeit	Eventuelle Fehler, die während der Laufzeit des Programms auftreten können, werden abgefangen. Hierzu zählen Unterbrechungen durch den Benutzer in Form eines Tastatur-Befehls, fehlerhafte Antworten des Servers, als auch Fehler, die innerhalb von Strategien auftreten könnten. Außerdem wird gewährleistet, dass innerhalb eines Zuges auch nur eine Antwort von dem Programm an den Server gesendet wird.

Qualitätskriterien	
Betrachtetes Kriterium	Beschreibung
Benutzbarkeit	Die Software kann, je nach gewählter Installationsmethode, sehr leicht installiert und in Betrieb genommen werden. Durch die Verwendung des bereitgestellten Dockerfiles wird dies noch einmal vereinfacht und ist somit auch für Laien möglich. Da es sich bei dieser Software um einen komplett automatisierten Spieler handelt, welcher somit keinerlei Benutzerinteraktionen erfordert, sind keinerlei Vorkenntnisse über das Spiel nötig. Zudem wird dem Benutzer auch nicht ermöglicht den Computerspieler zu steuern.
Effizienz	Die Durchlaufzeit des Programms, also die Zeit, die der Agent für die Berechnung seines nächsten Zuges benötigt, liegt bei unter einer Sekunde. Dies ist dabei natürlich vom verwendeten Prozessor abhängig. Wir gehen hier von einer durchschnittlichen CPU mit 2,4GHz aus. Aufwendigere Berechnungen und Strategien werden erst berechnet, sobald bestimmte Kriterien erfüllt sind, die diese notwendig machen.
Wartbarkeit	Der Quellcode ist größtenteils modular aufgebaut. Neue Strategien lassen sich einfach implementieren, indem sie zuerst als Funktion angelegt werden und anschließend an der gewünschten Stelle in der Funktion „tryToSurvive“ integriert wird.
Portabilität	Auf den gängigen Betriebssystemen (Linux, Windows, Mac OS) ist Python verfügbar. Es muss nur gegebenenfalls vorher installiert werden. Ebenfalls lässt sich das Programm als Docker Container installieren. Es kann mittels Parameter gewählt werden, ob der Agent lokal oder online spielen soll. Für das online Spielen ist eine Internetverbindung notwendig.

Tabelle 1: Betrachtete wissenschaftliche Qualitätskriterien und deren Erfüllung

5.3 Diskussion

Aktuell besteht das Problem, das nicht weit genug im Voraus geprüft wird, ob der Agent von einem anderen Spieler eingekesselt wird. Bleibt dem Agenten genug Platz, kann dieser beschleunigen und mit einem Sprung wieder in ein größeres Areal gelangen. Genügt der Platz nicht, so versucht der Agent das verbleibende Areal möglichst

effektiv zu nutzen, jedoch final ausscheiden. Dies ist mit der grundsätzlichen Überlebensstrategie unseres Agenten zu begründen. Wie schon in Kapitel 4.2.4 genannt, ist unser vorrangiges Ziel lange auszuweichen, um Zeit zu gewinnen, in der sich potenzielle Gegner selbstständig eliminieren könnten. Dies ist rückblickend weniger ertragreich als zuerst erhofft. So zeigt sich, dass ein Aufbauen auf den Fehlern von Anderen zu keinem kontinuierlichen Sieg führen kann. Daher ist der Sieg unseres Agenten stark von den Strategien anderer Gegenspieler abhängig, weswegen wir nur selten gegen aggressivere Agenten gewinnen.

Um die Überlebenschancen des Agenten zu erhöhen, würde sich ein Vorausrechnen der Züge anderer Spieler anbieten, um eine potenzielle Gefahr früher erkennen zu können und dieser gegebenenfalls auszuweichen. Jedoch wird auch hier wieder auf einem Fehler der Gegenspieler aufgebaut, was unter Umständen eine inhärent schlechte Strategie darstellt. Sollte jedoch kein teilnehmender Spieler eine aggressive Strategie aufweisen, ist unser Bot oftmals siegreich und überlebt seine Kontrahenten.

Insgesamt können wir also behaupten, dass unser Agent eine solide Erfolgsquote erzielt, sich jedoch nur unregelmäßig gegenüber aggressiven Spielern durchsetzt.

6 Benutzerhandbuch

Als Ausgangspunkt unserer Abgabe wird einerseits ein fertiges Dockerfile zur Verfügung gestellt, andererseits besteht auch die Möglichkeit den Quellcode direkt auf einem Betriebssystem der Wahl auszuführen. Beispielsweise wurde das Projekt auf Windows 10, macOS als auch Debian Linux getestet. Die aus den jeweils verwendeten Betriebssystemen resultierenden Vorgehensweisen der Installation von nötiger Software ist nicht völlig deckungsgleich und wird daher nicht näher Erläutert, jedoch wird auf eine Installationsanweisung unter Linux u.Ä. verwiesen.

Hinweis
Folgende Erweiterungen sind **nicht** innerhalb des Docker-Containers verfügbar:

- Anzeige des Live-Spielfeldes

6.1 Installation ohne Docker

Die Installation ohne Docker erfordert folgende, bereits installierte Software. Für die Installation dieser Software sind Installationsanweisungen verlinkt.

- Python 3.8

Nach erfolgreicher Installation der vorausgesetzten Software muss ein Terminal geöffnet werden und mittels des *cd <VERZEICHNIS-PFAD>* Befehls in das heruntergeladene Projekt navigiert werden.

Da die notwendigen Bibliotheken noch installiert werden müssen, muss zu aller erst eine virtuelle Python-Umgebung erstellt werden.

```
python3 -m venv venv
```

Anschließend sollte ein neuer Ordner mit dem Namen „venv“ erstellt worden sein. Nun müssen wir festlegen, dass diese virtuelle Umgebung genutzt wird.

```
source venv/bin/activate
```

Um die notwendigen Bibliotheken zu installieren, kann abschließend folgender Befehl verwendet werden:

```
pip install --upgrade pip && \
pip install --no-cache-dir -r requirements.txt
```

Um das Programm anschließend auszuführen, werden weitere sogenannte Umgebungsvariablen benötigt.
Bei Online-Nutzung:

- Die URL des Servers
- Der zu verwendene API-Key

Bei Offline-Nutzung:

- Das das Programm Offline gestartet werden soll
- Der Pfad zur gültigen Spielfeld-Datei

Die URL des Servers wird beim Starten des Programms mit übergeben. Dies geschieht mittels folgender Parameter-Übergabe:

```
export URL=<URL>
z.B. export URL=wss://msoll.de/spe_ed
```

Außerdem wird der API-Key beim Starten des Programms übergeben. Dies geschieht folgendermaßen:

```
export KEY=<KEY>
z.B. export KEY=72ILGT3YVIW5DV2UR3L5
```

Bei Offline Nutzung muss dies dem Programm mitgeteilt werden. Das geschieht folgendermaßen:

```
export Online=<True/False>
z.B. export Online=False
```

Auch der Pfad zu einer gültigen Spielfeld-Datei muss vorhanden sein. Das Format entspricht hierbei dem vorgegebenen Format des InformatiCups. Einige Beispiele sind außerdem im Hauptverzeichnis dieses Projektes hinterlegt.

```
export Playground=</PFAD/ZUM/SPIELFELD.json>
z.B. export Playground=/Users/ich/Spe_ed/spielfeld.json
```

Zusammengesetzt sieht der Startbefehl des Programms zum Beispiel wie folgt aus:

```
export URL=wss://msoll.de/spe_ed && \
export KEY=72ILGT3YVIW5DV2UR3L5 && \
python3 main/__init__.py
```

Beginnt das Spiel, so wird neben der Ausgabe über die Konsole, auch das gesamte Spielfeld generiert und auf dem Monitor angezeigt. Nach Abschluss eines Spiels wird das gesamte Spielfeld als Bild unter

```
results/(Won/Lost/Draw)/result<Zeitstempel>.jpg
```

abgespeichert.
Unterbrechen lässt sich das Programm mittels der Tastenkombination *CTRL+C*.

6.2 Installation mit Docker

Für die Installation unserer Software mit Docker, wird lediglich folgende, bereits installierte Software vorausgesetzt. Folgen Sie für die Installation dieser Software den jeweils verlinkten Installationsanweisungen.

- Docker Desktop

Nach erfolgreicher Installation der vorausgesetzten Software muss ein Terminal geöffnet und mittels des *cd <VERZEICHNIS-PFAD>* Befehls in das heruntergeladene Projekt navigiert werden. Anschließend lässt sich ein Docker-Container mittels des mitgelieferten Dockerfiles erstellen. Hierfür wird in dem geöffneten Terminal-Fenster folgender Befehl verwendet:

```
docker build -t spe_ed .
```

Dieser sollte nach Abschluss unter anderem den Text „FINISHED“ im Terminal anzeigen. Um den gerade erstellten Docker-Container auszuführen, werden weitere sogenannte Umgebungsvariablen benötigt. Hierzu zählen unter anderem:

- Die Angabe, ob das Projekt mittels Docker ausgeführt wird
- (Optional) Die Angabe, wo Spielergebnisse gespeichert werden sollen

Bei Online-Nutzung außerdem:

- Die URL des Servers
- Der zu verwendene API-Key

Bei Offline-Nutzung außerdem:

- Das das Programm Offline gestartet werden soll
- Der Pfad zur gültigen Spielfeld-Datei

Die URL des Servers wird beim Starten des Programms mit übergeben. Dies geschieht mittels folgender Parameter-Übergabe:

```
export URL=<URL>
z.B. export URL=wss://msoll.de/spe_ed
```

Außerdem wird der API-Key beim Starten des Programms übergeben. Dies geschieht folgendermaßen:

```
export KEY=<KEY>
z.B. export KEY=72ILGT3YVIW5DV2UR3L5
```

Bei Offline Nutzung muss dies dem Programm mitgeteilt werden. Dies geschied folgendermaßen:

```
export Online=<True/False>
z.B. export Online=False
```

Auch der Pfad zu einer gültigen Spielfeld-Datei muss vorhanden sein. Das Format entspricht hierbei dem vorgegebenen Format des InformatiCups. Einige Beispiele sind außerdem im Hauptverzeichnis dieses Projektes hinterlegt.

```
export Playground=</PFAD/ZUM/SPIELFELD.json>
z.B. export Playground=/Users/ich/Spe_ed/spieldfeld.json
```

Ob der Start des Programms über Docker stattfindet, muss dem Programm auch noch übermittelt werden. Dies geschieht mittels folgender Parameter-Übergabe:

```
--env Docker=<True/False>
z.B. --env Docker=True
```

Ist gewünscht, dass die Spielergebnisse nach Abschluss eines Spiels außerhalb des Docker-Containers zur Verfügung stehen, dann muss dies mittels folgenden Parameters mit angegeben werden:

```
-v <PFAD_ZUM_ORDNER>:/usr/src/app/results
z.B. -v /Users/Ich/speed/results:/usr/src/app/results
```

Zusammengesetzt sieht der Startbefehl des Docker-Containers zum Beispiel wie folgt aus:

```
docker run \
-v /Users/Ich/speed/results:/usr/src/app/results \
--env Docker=True \
--env URL=wss://msoll.de/spe_ed \
--env KEY=72ILGT3YVIW5DV2UR3L5 \
--name InformatiCup2021 spe_ed
```

Die Ausgabe der aktuellen Vorgänge erfolgt hierbei über die Konsole. Soll der Docker-Container stattdessen im Hintergrund laufen, so wird beim Starten des Docker-Containers der Befehl

```
docker run
```

in

```
docker run -d
```

abgeändert.

Literatur

[Kry] Dr. Veikko Krypczyk. Softwarequalität – so misst und verbessert man Software.

[Luc] Laurent Luce. A*-Algorithmus.

[NEA] NEAT Overview. https://neat-python.readthedocs.io/en/latest/neat_overview.html. Onlinequelle; Abgerufen am 04.01.2021.

[Sar] Sargeras. Solving mazes using Python: Simple recursivity and A* search.

Glossar

Docker Docker ist eine Möglichkeit, verteilte Anwendungen und gekoppelte Dienste als ein einziges, unveränderliches Objekt zu definieren, zu verpacken, auszuführen und zu verwalten.. 13, 14

Docker-Container Ein Docker-Container-Image ist ein schlankes, eigenständiges, ausführbares Softwarepaket, das alles enthält, was zum Ausführen einer Anwendung erforderlich ist: Code, Laufzeit, Systemtools, Systembibliotheken und Einstellungen.. 11, 13, 14

Dockerfile Ein Dockerfile wird verwendet, um Docker-Images zu erstellen. Es ist eine einfache Textdatei, die aus einer Reihe von Anweisungen oder Befehlen besteht, die von einem automatisierten Build-Prozess in Schritten von oben nach unten ausgeführt werden.. 11, 13

Package Ein Package enthält ein geschlossenes Code-Paket, welches von Dritten genutzt werden kann, um Package spezifische Aufgaben zu erfüllen.. 2

PyGame PyGame ist eine Python Bibliothek zur Spieleprogrammierung. Sie beinhaltet unter anderem verschiedene Module zum Anzeigen und Steuern von Grafik und Ton.. 3

Repository Ein Repository speichert Code und macht ihn für mehrere Personen zur gleichzeitigen Mitarbeit verfügbar. Es bietet zusätzlich eine Versionierung des gespeicherten Codes, sodass man immer auf frühere Variationen zurückgreifen kann.. 2