InformatiCup 2021

Lösung von Team Lehnurr

Verfasser:	Team Lehnurr (Felix Lehnerer & Silas Schnurr)
Hochschule:	Duale Hochschule Baden-Württemberg Karlsruhe
Fachrichtung:	Informatik
Datum:	17.01.2021

Inhaltsverzeichnis

Glossar

API	Programmierschnittstelle (engl. Application Programming Interface)
Black-Box	Ein System, welches es nicht ermöglicht, die inneren Abläufe nachzuverfolgen.
Bot	Ein Programm, welches das Spiel spe_ed spielt, aber nicht von informatiCup Teilnehmern entwickelt wurde. (von engl. robot = Roboter)
Erweiterter, gerichteter Graph	Ein gerichteter Graph im Sinne der Graphentheorie, dessen Kanten zusätzliche Informationen halten.
Experimentelles Prototyping	Das Erstellen eines Prototyps, welcher der Konzepterstellung und Anforderungsdefinition dient, aber vollständig verworfen wird.
Game-Solver	Ein Computerprogramm, welches mit dem Ziel entwickelt wurde, ein bestimmtes Spiel zu Spielen und möglichst oft zu gewinnen.
KI	Künstliche Intelligenz
Pfad	Für das Siel spe_ed: Eine Folge von passierten Zellen, welche durch eine definierte Abfolge von Aktionen entsteht. Ein Pfad ist nicht an die maximale Rundenanzahl eines Spiels gebunden und kann diese über- oder unterschreiten.
Prototyp	Ein funktionsfähiges System, welches die Grundlage für ein Endprodukt darstellt.
RAD	Konzept für die schnelle Anwendungsentwicklung mithilfe von Prototyping. (engl. Rapid Application Development)
Solver	*Siehe Game-Solver*
Strategie	*Siehe Pfad*
Tool	Eine Software, welche den Entwicklungs-, Fehlerbeseitigungs- oder Test-Prozess unterstützt.

Abbildungs- und Tabellenverzeichnis

1 Einleitung

Der Wettbewerb InformatiCup 2021 fordert Studierende heraus, einen Game-Solver für das Spiel spe_ed zu entwickeln. Bei diesem Spiel müssen die Spieler ähnlich wie in den Klassikern „Tron“, „Snake“ oder „Achtung Kurve!“ einen Pfad auf einem begrenzten Spielfeld ziehen, dabei Kollisionen vermeiden und somit länger als die anderen Spieler überleben. Diese Ausarbeitung beschäftigt sich mit dem theoretischen Hintergrund der von dem Team Lehnurr entwickelten Lösung.

Der entwickelte Solver basiert auf einem heuristischen Lösungskonzept, welches das Spiel spe_ed als erweiterten, gerichteten Graphen auffasst. Erweiterter Graph bedeutet in diesem Kontext, dass die Kanten mit zusätzlichen Informationen angereichert werden. Das Spielfeld wird unter verschiedenen Gesichtspunkten bewertet. Diese Bewertungen werden mit unterschiedlicher Gewichtung miteinander kombiniert. Durch dieses Verfahren ist es möglich, für alle Situationen eine einheitliche Entscheidungsfindung zu verfolgen, sodass keine spezifischen Sonderfälle betrachtet werden müssen.

Mit einer Gewinnrate von 85% bei 35 Spielen gegen andere Wettbewerbsteilnehmer im Zeitraum von einer Woche vor der Abgabefrist für Lösungen, kann die Zielvorgabe „möglichst oft zu gewinnen“ als erreicht deklariert werden.

In der vorliegenden Arbeit werden die theoretischen Grundlagen erörtert, verschiedene Lösungskonzepte aufgezeigt und die Entscheidungsfindung des Game-Solvers betrachtet. Des Weiteren wird auch die Architektur des Systems erläutert, spezifische Implementierungsdetails sind jedoch nicht enthalten. Konkrete Fragen zu der Implementierung werden durch die Dokumentation des verwendeten GitHub Repositories lehnurr/spe-ed-solver und die Dokumentation des Quellcodes unter spe-ed-docs.lehnurr.de beantwortet.

2 Geplantes Vorgehen

Um einen ordnungsgemäßen Ablauf des Entwicklungsprozesses zu gewährleisten gilt es zunächst eine Vorgehensweise zu erarbeiten, die auf den Kontext des Projekts angepasst ist.

2.1 Methodik

Für ein zielgerichtetes Vorgehen innerhalb des Projektes wird das *Rapid Application Development (RAD)* Konzept angewandt. Das Vorgehen der *RAD*-Methode erfordert es zunächst, die Problemstellung des Wettbewerbs zu betrachten und konkrete Ziele zu definieren. Neben der Analyse der Problemstellung muss auch die Bewertungsgrundlage für die Feststellung der Qualität einer möglichen Lösung betrachtet und erörtert werden. [1]

Die konkreten Ziele für die Lösung des Problems sind im Folgenden aufgeführt:

Z1. Erfüllung der Wettbewerb-Anforderungen

Die technologischen (Verwendung von *WebSockets* und *Docker*) und funktionalen (möglichst oft gewinnen und automatisches beenden) Vorgaben müssen erfüllt sein. [2]

Z2. Transparentes Lösungsverfahren

Jede Entscheidung des *Solvers* muss nachvollziehbar sein. Es darf sich nicht um eine Black-Box-Lösung handeln.

Z3. Einheitliches Lösungsverfahren

Die Lösung muss für alle Spielsituationen nach dem gleichen Vorgehen entscheiden und nicht für bestimmte Sonderfäll separate Implementierungen verwenden.

2.2 Projektverlauf

Im Anschluss an die Analyse und dem Festlegen einer Bewertungsgrundlage gilt es theoretische Lösungsansätze zu erörtern, als Prototyp zu implementieren, wenn nötig zu überarbeiten und die verschiedenen Ansätze miteinander zu vergleichen. Bei dieser Art des Prototypings handelt es sich um ein experimentelles Prototyping. Das bedeutet, dass die Bestandteile des Prototyps nicht für die Implementierung des eigentlichen Systems verwendet werden. Der Prototyp dient in diesem Fall der Festlegung des Lösungsansatzes und Definition von Architekturanforderungen. Die Ergebnisse des experimentellen Prototypings bilden somit die Grundlage für die Implementierung des Systems.

Der zeitliche Verlauf des Projekts ist durch einen Meilensteinplan (siehe *Abbildung 1*) vorgegeben. Da das Team Lehnurr aus zwei Personen besteht, ist diesbezüglich kein umfangreicher Projektstrukturplan notwendig, um einen kontrollierten Projektverlauf zu gewährleisten. Die Phase „Prototyping" kann dabei in drei Arbeitsbereiche unterteilt werden:

- Analyse und Konzeptentwicklung (30.09 – 11.10.2020)
- Entwicklung und Vergleich von Lösungsansätzen (11.10. – 16.11.2020)
- Konzeptentwicklung für die Implementierung (02.11. – 16.11.2020)

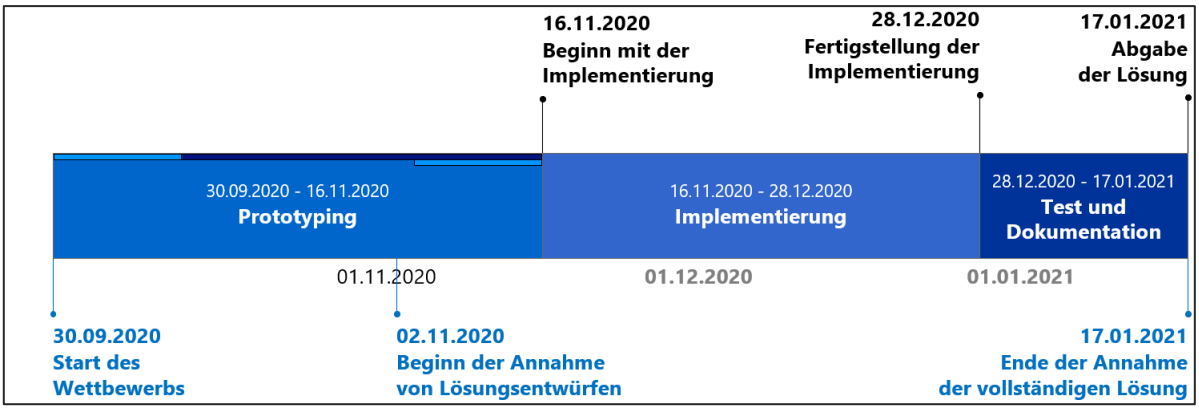

Abbildung 1: Meilensteinplan

Die Kommunikation wird durch einen wöchentlichen Jour fixe sichergestellt. Die Verwendung von geteilten Dokumenten und einer gemeinsamen Quellcodeverwaltung ermöglicht das effiziente gemeinsame Arbeiten auf dem aktuellen Entwicklungsstand.

2.3 Entwicklungstools

Zur besseren Unterstützung des Entwicklungsprozesses und der Überwachung der Ergebnisse sind Erweiterungen des Prototyps und der finalen Software angedacht.

Aufgrund von hohen Wartezeiten der bereitgestellten API des Wettbewerbs, ist die Integration einer Simulation angedacht. Dort kann ohne Wartezeiten eine kontrollierte Umgebung gebildet werden, in welcher ein entwickeltes Verhalten auch gegen sich selbst getestet werden kann.

Zusätzlich soll eine visuelle Oberfläche bereitgestellt werden. Somit wird ermöglicht, dass der Stand des Spiels direkt eingesehen, und Entscheidungsgrundlagen nachvollzogen werden können. Ziel der Visualisierung ist es, unvorhergesehene Fehler in der Strategie der Lösung und Programmierfehler schnell identifizieren zu können.

Des Weiteren ist es wichtig, eine geeignete Eingabemöglichkeit für spielbezogene Paramater bereitzustellen, sodass ohne hohen Aufwand verschiedene Spielstrategien ausgewertet werden können. Dies ist unter anderem durch eine Konfigurationsdatei oder die Übergabe von Kommandozeilenargumenten möglich.

3 Problemstellung

Das zu lösende Problem wird im Folgenden formal formuliert, um darauf basierend Analysen durchzuführen, eine Bewertungsgrundlage festzulegen und verschiedene Lösungsmöglichkeiten aufzuzeigen.

3.1 Modellierung

Um die Vergleichbarkeit der Lösungskonzepte zu gewährleisten, ist eine einheitliche Modellierung des Spiels spe_ed erforderlich. Die Modellierung erfolgt in der Normalform, da das Spiel der nicht-kooperativen Spieltheorie zuordenbar ist und die Aktionen der Spieler innerhalb einer Runde gleichzeitig erfolgen. [3, pp. 14,15]

Menge der Spieler

$$P \subseteq \{1, 2, 3, 4, 5, 6\} \mid 2 \leq |P|$$

Strategieraum

Die Strategiemenge S in einer Runde i für einen Spieler j setzt sich aus den fünf möglichen Aktionen zusammen.

$$S_{i,j} = \{turn\_left, turn\_right, slow\_down, speed\_up, change\_nothing\}$$

Der gesamte Strategieraum einer Runde i ist somit die Produktmenge S_i der Strategiemengen aller Spieler.

$$S_i = \prod_{j \in P} S_{i,j}$$

Die Betrachtung von mehreren Runden ist entsprechend der Modellierung durch das Bilden der Produktmenge der jeweiligen Runden möglich. Hierbei gilt es jedoch zu beachten, dass die Menge der Spieler P durch das Ausscheiden eines Spielers j verändert wird.

$$P \coloneqq P/j$$

Nutzenfunktion

Der Spieler j muss in jeder Runde i aus der Strategiemenge $S_{i,j}$ die beste Möglichkeit x auswählen. Hierfür ist ein Zusammenhang zwischen den Entscheidungen und dem Resultat einer Runde herzustellen, um festzulegen was die beste Möglichkeit ist. Als Resultat einer Runde werden WIN, BETTER, SAME, WORSE und LOSE betrachtet[1], wobei als gewünschtes Resultat

$$WIN > BETTER > SAME > WORSE > LOSE$$

gilt. Diese ordinale Ordnung ergibt sich aus der Gewinnwahrscheinlichkeit $W_{i,j,x} \in [0,1]$ in Runde i von Spieler j für Entscheidung x und der Gewinnwahrscheinlichkeit $W_{i-1,j,y} \in [0,1]$ der vorherigen Runde $i-1$ für die bereits getroffene Entscheidung y.

$$Resultat(i,j,x,y) = \begin{cases} WIN, & W_{i,j,x} = 1 \\ BETTER, & W_{i,j,x} - W_{i-1,j,y} > 0 \wedge W_{i,j,x} \neq 1 \\ SAME, & W_{i,j,x} = W_{i-1,j,y} \\ WORSE, & W_{i,j,x} - W_{i-1,j,y} < 0 \wedge W_{i,j,x} \neq 0 \\ LOSE, & W_{i,j,x} = 0 \end{cases}$$

Der Spieler muss dementsprechend durch die Entscheidung für Möglichkeit x die Nutzenfunktion N maximieren. [4]

$$N = Resultat(i,j,x,y)$$

Bei dieser Modellierung gilt es zu beachten, dass strategische Entscheidungen über mehrere Runden hinweg möglich sind, indem diese in die Berechnung der Gewinnwahrscheinlichkeit W mit einbezogen werden.

[1] Laut den Regeln für spe_ed gibt es kein Unentschieden. [4] [12]

3.2 Spielparameter

Die spe_ed Regeln sind bereits durch die Aufgabenstellung des Wettbewerbs [2] gegeben und werden deshalb hier nicht erneut erläutert.

In jedem spe_ed-Spiel gibt es variable Eingangsparameter, die den Spielfluss beeinflussen. Die Analyse der Eingangsparameter kann im späteren Verlauf auf die Simulation übertragen werden. Somit kann die Simulation genauer das echte Spielverhalten darstellen.

Zunächst wird für jedes Spiel ein Spielfeld benötigt, dass eine, über den Spielverlauf hinweg, statische Größe hat. Diese unterscheidet sich jedoch zwischen einzelnen Spielen. In *Abbildung 2* ist der Box-Plot der Breite und Höhe des Spielfelds von gesammelten Daten über 130 Spiele dargestellt. Aus der Darstellung geht hervor, dass die Höhe und Breite des Spielfelds gleichverteilt ist. Der minimal auftretende Wert liegt bei 41 Zellen, der maximale Wert bei 80 Zellen. In *Abbildung 3* ist die resultierende Flächenverteilung als Histogramm dargestellt. Daraus geht hervor, dass keine offensichtlichen Abhängigkeiten zwischen der Breite und Höhe des Spielbretts bestehen.

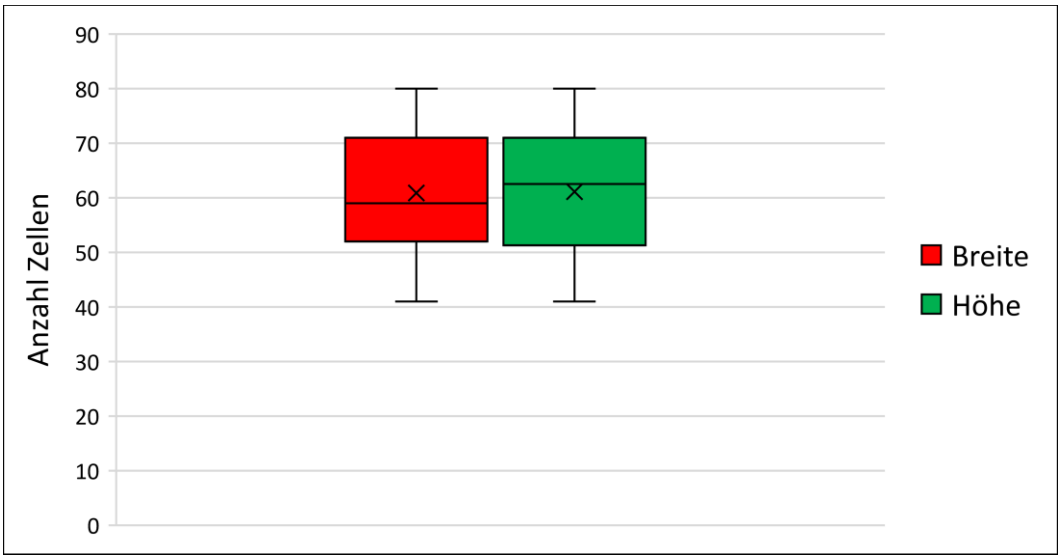

Abbildung 2: Box-Plot Breite & Höhe des Spielfeldes

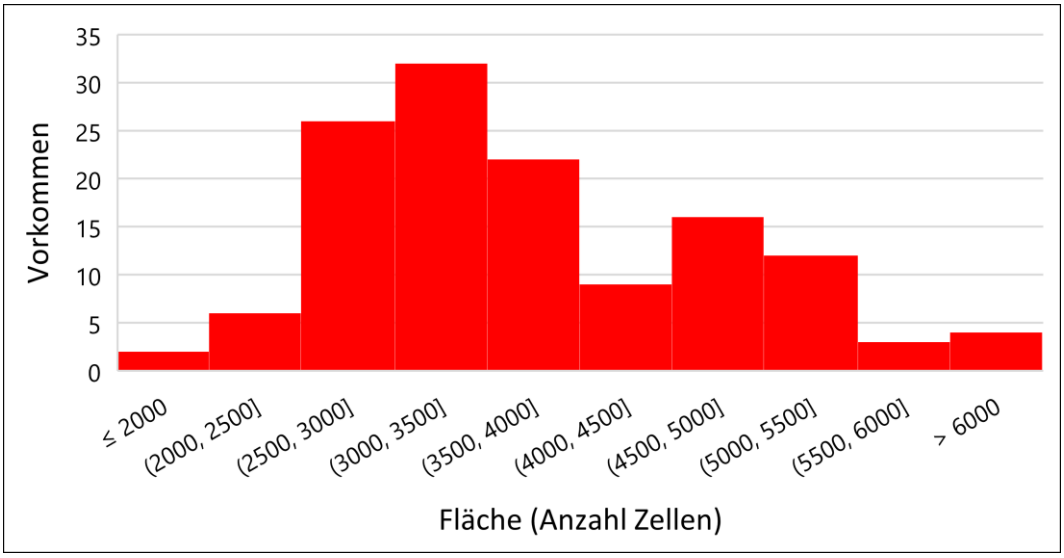

Abbildung 3: Histogramm der Flächenverteilung

Neben der Spielfeldgröße ändert sich in jeder Runde die Zeit, die dem Spieler zum Berechnen seiner Antwort zur Verfügung steht. In *Abbildung 4* sind die verfügbaren Zeiten von insgesamt 5.000 Runden (von verschiedenen Spielen) in einem Box-Plot dargestellt. Die gemessenen verfügbaren Zeiten liegen größtenteils zwischen 4 und 15 Sekunden. Zu beachten ist, dass die gemessene Deadline aufgrund der durchzuführenden Synchronisation der Client- und Serverzeit leicht verschoben sein kann.

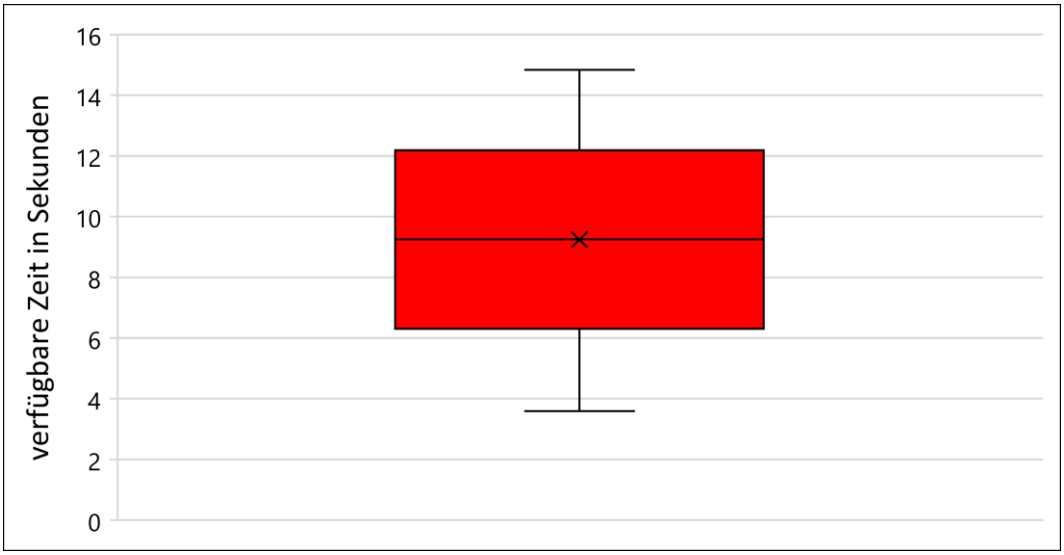

Abbildung 4: Box-Plot verfügbarer Zeit pro Runde

Zu beachten ist, dass alle gemessenen Daten lediglich als Orientierung, nicht aber als Garantie zu betrachten sind. Als wichtigster Parameter eines Spiels sind die gegnerischen Spieler zu betrachten. Diese sind aufgrund ihres komplexen Verhaltens und der komplexen Interaktion mit dem eigenen Spieler nicht ohne Weiteres analysierbar.

Die vollständigen Daten der Auswertungen, sowie die verwendeten Rechenvorschriften zum Erstellen der Diagramme sind in *Anhang 1* und *Anhang 2* einsehbar.

3.3 Bewertungsgrundlage

Als bekanntes Kriterium zur Messung der Qualität einer Lösung für den Wettbewerb wird die Gewinnrate gegenüber anderen Spielern genannt [2, p. 5]. Die Messung dieser ist jedoch nicht trivial.

Die größte Herausforderung der Bewertung ist die fehlende Transitivität bei der Gewinnrate. Gewinnt ein Spieler „A“ zumeist gegen Spieler „B“ und Spieler „B“ gewinnt in den meisten Fällen gegen Spieler „C“, ist nicht garantiert, dass Spieler „A“ auch in der Mehrheit der Spiele gegen Spieler „C“ gewinnt. Durch dieses Verhalten ist nicht direkt feststellbar, wie gut oder schlecht eine Lösung ist. Definitionen, die andere Spieler miteinschließen setzen voraus, dass man deren Verhalten kennt.

Nach Tests der bereitgestellten Online API zeigt sich, dass nicht immer echte gegnerische Spieler auftreten. Auch ist nicht (in einem längeren Zeitraum) feststellbar in wie vielen Fällen man gegen welches Team gespielt hat. Eine statistische Auswertung auf Basis von online durchgeführten Spielen stellt somit keine akkurate Messung der eigenen Gewinnrate dar, ist jedoch eine mögliche Annährung, welche nicht auf Annahmen zu gegnerischem Verhalten basiert. Messungen, die auf Simulationen basieren, zeigen die Gefahr, durch die eigene Lösungsidee beeinflusst zu werden und sind eventuell ungültig für kreative Lösungsansätze anderer Teams.

3.4 Lösungsmöglichkeiten

Die möglichen Alternativen liegen den verschiedenen Lösungskonzepten in der nicht-kooperativen Spieltheorie zugrunde. [5]

Der naive Ansatz eine dominante Strategie zu wählen, welche gegenüber den Strategien der Gegner immer ein stark dominierendes Pareto-Optimal darstellt ist wünschenswert. [3, pp. 15-20] Durch die Modellierung ist jedoch erkennbar, dass aufgrund des exponentiellen Wachstums des Strategieraums in jeder Runde, die Anzahl der Strategien (eine Abfolge von Entscheidungen) exponentiellen wächst. Für ein Spiel mit n Runden kann daher mit einem Aufwand von $O(n)$ eine Strategie bestimmt werden. Für die Bestimmung einer dominanten Strategie müssen jedoch alle Strategien bekannt sein. Der Aufwand für diese Ermittlung liegt für m Spieler mit

$$\left|S_{i,j}\right| = 5 \qquad \text{bei} \qquad O(m^n)$$

und ist durch „moderne Mehrkernprozessor-Computer mit einigen GB RAM" [6] innerhalb der vorgegebenen Deadlines[2] nicht berechenbar. Zusätzlich gibt es die Problemstellung, dass für die Wahl einer dominanten Strategie davon ausgegangen werden muss, dass auch die Gegner eine dominante Strategie verfolgen. Dies ist in dem Fall von spe_ed nicht gegeben.

Das Bilden eines Nash-Gleichgewichts ist ebenfalls für einen einzelnen Spieler nicht möglich. Dies ist dadurch begründet, dass die Strategie nicht zu Beginn des Spiels festgelegt wird, sondern von den Spielern während des Spiels an die Aktionen der Gegner angepasst wird. [3, pp. 20-26]

[2] Siehe *3.2 Spielparameter*

Außerhalb der Spieltheorie gibt es für dieses Suchproblem die Möglichkeit eine künstliche Intelligenz auf Basis eines neuronalen Netzwerks einzusetzen oder einen heuristischen Lösungsweg zu wählen.

Der Einsatz einer künstlichen Intelligenz (KI) erfordert ausreichend Trainingsdaten, um gute Ergebnisse zu erzielen. Das Sammeln von Spieldaten über die zur Verfügung gestellte Plattform ist möglich, das Problem hierbei ist jedoch, dass auf dieser Plattform zum einen Computerspieler mit naiven Strategien das Ergebnis verfälschen und zum anderen die Lösungen von anderen Wettbewerb-Teilnehmern nicht dem Verhalten ihrer abgegebenen Lösung entsprechen müssen. Daraus resultiert, dass die trainierte KI sehr gut mit den vom Spiel bereitgestellten Computerspielern und den Vorversionen einiger abgegebenen Lösungen umgesehen kann, gegen die abgegebenen Lösungen jedoch möglicherweise versagt. Dies kann verhindert werden, indem weitere künstliche Trainingsdaten generiert werden. Da die Teilnahme am Wettbewerb nicht mit dem Ziel eine Künstliche Intelligenz zu modellieren und zu trainieren erfolgt, sondern als Herausforderung im Bereich Programmieren betrachtet wird, wird keine künstliche Intelligenz eingesetzt.

Als letzte Lösungsmöglichkeit wird ein heuristischer Ansatz betrachtet. Hierbei wird versucht das Pareto-Optimal zu erreichen, aber im Gegensatz zu einer dominanten Strategie werden nicht alle möglichen Strategien betrachtet. Dieser Ansatz lässt Spielraum für mögliche heuristische Betrachtungen, die auf Basis einer reduzierten Sicht auf das Spiel Entscheidungen treffen. Aufgrund des umfangreichen Lösungsraums bietet sich die heuristische Herangehensweise an, da nicht alle Lösungen des Lösungsraum zeitgerecht bearbeitet werden können.

4 Pfadsuche

Für die spätere Entscheidungsfindung ist die Suche nach Pfaden unverzichtbar. Durch die Modellierung ist zu sehen, dass der Strategieraum mit jeder Runde exponentiell wächst, sodass die Kombination aus allen möglichen Aktionen nur bis zu einem bestimmten Grad möglich ist. Wird nur der eigene Strategieraum betrachtet, können deutlich mehr Strategien berechnet werden, für die jedoch nicht garantiert ist, dass sie kollisionsfrei verlaufen. Im Folgenden gilt es verschiedene Ansätze der Pfadsuche aufzuzeigen. Ein Pfad ist hierbei eine Strategie eines Spielers, also eine Abfolge von Aktionen. Diese Pfade sind nicht präfixfrei, sodass der Pfad $[\,turn\_left,\ turn\_left\,]$ und der daraus resultierende Pfad $[\,turn\_left,\ turn\_left,\ turn\_right]$ als zwei verschiedene Pfade aufgefasst werden. Die Ansätze der Pfadsuche können unter gleichen Bedingungen anhand der folgenden drei Kriterien miteinander verglichen werden.

K1. Anzahl der gefundenen Pfade

K2. Anzahl der erreichbaren Zellen

K3. Verteilung der erreichbaren Zellen auf dem Spielfeld

4.1 Vorwärtssuche

Die Vorwärtssuche stellt die einfachste Form der Suche dar. Ohne Verbesserungen müssen zur Erweiterung eines Pfades alle fünf Aktionen durchgeführt werden. Ist die Aktion gültig, kann das entstandene Kind als neuer Pfad aufgefasst werden. Basierend auf der Ausgangssituation eines Spielers in einer gegebenen Runde, lassen sich weitere mögliche Pfade finden, in den die Kinder eines gefundenen Pfades einer Bearbeitungsschlange angehängt werden.

Durch das beschriebene Verfahren sind in kurzer Zeit viele Pfade auffindbar, die allerdings keine bestimmte Richtung aufweisen. Durch die Erweiterung der Suchaufgabe durch einen definierten Zielpunkt ist die Priorisierung von Pfaden möglich (z.B. A*-Algorithmus).

Während die Suche ohne ein definiertes Ziel sehr effizient viele Pfade findet, sind unter diesen viele Pfade, die sich den gleichen Zielpunkt teilen. Falls nicht genügend Rechenleistung besteht, um das ganze Spielbrett abzudecken, besteht ebenso das Problem, dass viele Punkte um den Ursprung herum gefunden werden, aber nur wenige am äußeren Rand des Spielbretts. Durch eine priorisierte Vorwärtssuche in Richtung von zufälligen Punkten kann, insofern genügend Punkte gesucht werden, eine repräsentative Aussage über das ganze Spielbrett getroffen werden. Die priorisierte Vorwärtssuche hat allerdings den erheblichen Nachteil, dass nicht der kürzeste Pfad, sondern ein möglichst kurzer Pfad gefunden wird. Es kann dementsprechend nicht garantiert werden, dass es für einen gefundenen Zielpunkt keinen besseren Pfad gibt.

Somit ist die priorisierte Vorwärtssuche eine mögliche Alternative, die zum Einsatz kommen kann, wenn für die gewöhnliche, ungerichtete Suche nicht genügend Berechnungszeit zur Verfügung steht.

4.2 Rückwärtssuche

Neben der Vorwärtssuche kann ebenfalls die Rückwärtssuche betrachtet werden. Durch das komplexe Fortbewegungsverhalten von spe_ed Spielern ist allerdings zu beachten, dass gefundene Pfade, die Sprünge (nur jede sechste Runde möglich) enthalten, zur Runde und Position der Ursprungssituation passen.

Im Gegensatz zur Vorwärtssuche ist bei der Rückwärtssuche nur die priorisierte Vorgehensweise sinnvoll, da es nur einen einzigen sinnvollen Zielpunkt (Ausgangssituation des Spielers) gibt. Der Vorteil der Rückwärtssuche ist, dass bereits gefundene Pfade zur Ausgangsposition in einer Map gespeichert und wiederverwendet werden können. Somit kann die Rückwärtssuche mit bereits gefundenen Pfaden optimiert und beschleunigt werden.

4.3 Bidirektionale Pfadsuche

Bereits aus der Rückwärtssuche geht hervor, dass die Wiederverwendung zuvor gefundener Teilergebnisse vorteilhaft für die Performance der Suche ist. Die Idee der bidirektionalen Suche im Kontext von spe_ed beschreibt das initiale Suchen nach Teillösungen durch eine nicht priorisierte Vorwärtssuche. Diese ist jedoch in der Tiefe (Anzahl an zukünftigen Runden) beschränkt.

So wird ein Fangnetz für die Rückwärtssuche aufgebaut. Findet die priorisierte Rückwärtssuche ein Element dieses Fangnetzes, wurde eine gültige Lösung für einen Zielpunkt entdeckt. Alle Teilpfade werden ebenfalls dem Fangnetz an bereits gefundenen Lösungen hinzugefügt.

Wie bereits bei der priorisierten Vorwärtssuche beschrieben, kann auch durch die bidirektionale Suche nicht sichergestellt werden, dass ideale Pfade gefunden werden. Somit ist auch dieses Verfahren nur dann sinnvoll, wenn die Abdeckung, die durch eine gewöhnliche, nicht priorisierte Vorwärtssuche erreicht wird, nicht ausreichend für ein Zielvorhaben ist.

4.4 Optimierte Pfadsuche

Der Nachteil der Pfadsuche ist, dass sehr viele Berechnungen nötig sind und die Diversifikation der gefundenen Pfade nicht implizit gegeben ist. Die optimierte Pfadsuche versucht die betrachteten Lösungsansätze durch Modifikationen zu verbessern, sodass mehr Pfade mit einer höheren Diversifikation gefunden werden können.

4.4.1 Graph-basierte Optimierung

Durch die bereits betrachtete Pfadsuche werden die möglichen Pfade in jeder Spielrunde erneut berechnet, obwohl diese teilweise in bereits berechneten Pfaden enthalten sein können. Um die Pfade aus der vorherigen Runde wiederzuverwenden, muss ermittelt werden, welche Pfade, oder Teile von Pfaden, trotz der Veränderungen auf dem Spielfeld weiterhin gültig sind.

Der naive Ansatz diesbezüglich ist das Ermitteln von weiterhin gültigen Pfaden aus der Vorrunde. Dieser Ansatz liefert einerseits wenige Ergebnisse, da nur die Pfade verwendet werden können, welche der gewählten Aktion entspringen, andererseits muss für jeden möglichen Pfad überprüft werden, ob er eine Zelle verwendet, welche nicht mehr verfügbar ist. Durch diese Nachteile ist eine Neuberechnung der Pfade effizienter.

Der Ansatz der Graph-basierten Pfadsuche ist es nicht, einen gesamt berechneten Pfad wiederzuverwenden, sondern bereits berechnete Teile von Pfaden neu zusammenzusetzen. Die Zellen werden in diesem Kontext als Knoten und jede mögliche Bewegung als gerichtete Kante aufgefasst. Jede mögliche Bewegung bedeutet, dass für jeden Knoten bei jeder Geschwindigkeit, Richtung und *Art der Runde*[3] eine Kante auf den Zielknoten gibt. Bei 4 Richtungen, 10

[3] eine Runde erlaubt Sprünge oder erlaubt diese nicht.

Geschwindigkeitsstufen und 2 Arten von Runden hat jeder Knoten $4 * 10 * 2 = 80$ ausgehende Kanten[4]. Jede dieser Kanten ist im Startzustand abstrakt, sodass nur der Start-Knoten festlegt ist, jedoch nicht der Verlauf oder das Ziel zur Verfügung stehen. Sobald für die Pfadberechnung ein Schritt berechnet werden muss, wird die korrekte Kante über einen Index ermittelt. Für den Fall, dass die benötigte Kante abstrakt ist, wird diese berechnet und durch die konkrete berechnete Kante ersetzt, sodass dieser Teil-Pfad für Berechnungen in den folgenden Runden zur Verfügung steht. Die Berechnung der Kante bedeutet, dass die passierten Knoten bestimmt werden.

Wenn ein Knoten als nicht mehr verwendbar markiert wird, weil ein Spieler diesen belegt, werden alle eingehenden und ausgehenden Kanten entfernt. Zusätzlich müssen alle Kanten entfernt werden, welche durch diesen Knoten verlaufen. Die Ermittlung dieser Kanten hat einen geringen, vernachlässigbaren Aufwand, da die betroffenen Kanten über eine Look-Up-Tabelle abgefragt werden können. Ein Eintrag dieser Look-Up Tabelle ist beispielsweise

$$\Delta x = -10, \Delta y = 0 \Rightarrow RIGHT, NO\_JUMP, 10 \wedge RIGHT, DO\_JUMP, 10$$

Dieser Eintrag bedeutet, dass die Zelle, welche sich 10 Einheiten links und 0 Einheiten über der betroffenen Zelle befindet, zwei Kanten hat, welche nichtmehr gültig sind. Für beide betroffenen Kanten muss der Spieler mit der Geschwindigkeit 10 nach rechts fahren. Zum einen ist die Kante betroffen, welche für Runden mit Sprung steht, da die betroffene Zelle der Endpunkt des Sprungs ist, zum anderen ist die Kante betroffen, welche keinen Sprung erlaubt, da ebenfalls der Endpunkt passiert wird.

[4] Da die Art der Runde bei $Geschwindigkeit < 3$ irrelevant ist, ist eine Reduzierung auf $4 * 8 * 2 + 4 * 2 * 1 = 72$ möglich.

Für die Berechnung eines Pfads muss dementsprechend lediglich die Kante für den aktuellen Status des Spielers abgefragt werden. Diese Kante ist entweder schon vorberechnet und der Ziel-Knoten für die Bewegung steht ohne weitere Berechnungen fest, oder die Kante ist noch nicht berechnet und muss einmalig berechnet werden. Die dritte Möglichkeit ist, dass die Kante nicht mehr verfügbar ist, da der zugrundeliegende Pfad eine Kollision verursacht.

Zusätzlich zu der Kollisionsbestimmung mit persistierten Zellen auf dem Spielbrett, muss sichergestellt werden, dass ein berechneter Pfad nicht mit sich selbst kollidiert. Dies ist möglich, indem jeder berechnete Pfad Referenzen auf die verwendeten Kanten hält. Wenn dem Pfad eine neue Kante hinzugefügt werden soll, muss geprüft werden, dass diese neue Kante mit keiner referenzierten, bereits passierten Kante einen Schnittpunkt hat. Dies kann durch eine vereinfachte Schnittpunktberechnung[5] erfolgen, sodass nicht jede passierte Zelle überprüft werden muss.

Der Vorteil dieser Optimierung ist, dass seltener geprüft werden muss, ob eine mögliche Aktion eine Kollision mit dem Spielfeld verursacht und dass die Berechnung der Selbstkollisionen mit zunehmender Geschwindigkeit des Spielers weniger Rechenleistung erfordert (denn je schneller der Spieler sich bewegt, desto effizienter ist die Schnittpunktberechnung im Vergleich zu dem überprüfen aller passierter Zellen).

[5] Eine vereinfachte Schnittpunktberechnung bedeutet, dass keine Diagonalen möglich sind und im Fall einer Kante, welche einen Sprung beinhaltet lediglich der Start- und Endpunkt für die Schnittpunktberechnung betrachtet werden.

4.4.2 Heuristische Optimierung

Die heuristisch optimierte Pfadsuche arbeitet nach dem Ansatz der klassischen Vorwärtssuche, ignoriert jedoch einige auftretende Pfade für die Weiterberechnung, falls es bereits ähnliche Pfade gibt. Um die Ähnlichkeit festzustellen, müssen Ähnlichkeitsklassen definiert werden. Wenn zwei Pfade die gleiche Zelle passieren, wird über diese Klassifizierung festgestellt, ob beide Pfade weiterverfolgt werden müssen, oder ob ein Pfad ab dieser Stelle für zukünftige Berechnungen ignoriert wird. Die folgenden Ähnlichkeitsklassen betrachten lediglich die Position, Blickrichtung und Geschwindigkeit des Spielers, es ist jedoch auch möglich diese Klassen genauer zu spezifizieren, indem auch betrachtet wird, ob ein Sprung erfolgt oder wie viele Runden benötigt werden, um diese Zelle zu erreichen.

Ähnlichkeitsklasse 1: Unabhängig von der Geschwindigkeit und der Richtung werden die Pfade als ähnlich bewertet.
Ähnlichkeitsklasse 2: Die Geschwindigkeit darf um maximal 2 abweichen und die Richtung darf nicht um 180° gedreht sein.
Ähnlichkeitsklasse 3: Die Geschwindigkeit und Richtung müssen exakt übereinstimmen.

In dem Fall, dass zwei Pfade einer Ähnlichkeitsklasse zugeordnet werden, muss festgelegt werden, welcher Pfad abgebrochen wird. Diese Entscheidung kann durch pseudo-Zufall getroffen werden oder anhand der bereits resultierenden Pfade des Pfades, welcher zuerst die gemeinsame Zelle passiert.

4.4.3 Multithreading

Multithreading kann für die Pfadsuche eine erhebliche Performancesteigerung hervorrufen, da somit mehr Rechenleistung zur Verfügung steht. Dennoch gibt es hierbei Punkte, die beachtet werden müssen, um ein zuverlässiges Verhalten zu gewährleisten. Neben den allgemeinen Bedingungen, wie der korrekten Verwendung von Threadsafe-Datentypen, gibt es weitere, spezifische Gefahren.
Soll die Pfadsuche für mehrere Ausgangssituationen benutzt werden, um deren Ergebnis zu vergleichen, muss sichergestellt werden, dass die Threads separater Rechnungen in etwa die gleiche Rechenzeit erhalten.
Die höchste Performance beim Multithreading kann erwartet werden, wenn alle Kerne des Prozessors arbeiten. So ist zu beachten, dass Threads möglichst nicht an eine bestimmte und lange Aufgabe gebunden sind. Damit können Rechenaufgaben besser auf alle Kerne verteilt werden, wenn alle Aufgaben in Pakete eingeteilt werden, die gleichmäßig auf alle Threads zu verteilen sind. Dieses Konzept bedeutet im Kontext der Pfadsuche, dass eine Grundmenge von Pfaden berechnet und anschließend auf die Threads verteilt wird. Hierdurch ist gewährleistet, dass jeder Thread Pfade an verschiedenen Stellen des Spielfeldes berechnet, sodass bei einer Sackgasse in einem Bereich noch andere Endpunkte in anderen Bereichen für die Berechnung zur Verfügung stehen.

4.5 Vergleich der Ansätze

Die verschiedenen Ansätze können wie bereits formuliert durch die Kriterien K1 – K3 miteinander verglichen werden. Da der Vorteil der Rückwärtssuche und der bidirektionalen Pfadsuche gegenüber der Vorwärtssuche die Wiederverwendung von Pfaden ist, gilt es in einem Vergleich festzustellen, ob die Vorwärtssuche in Kombination mit der Graph-basierten Optimierung ein besseres Ergebnis erzielt.

Die heuristische Optimierung hat einen sehr negativen Einfluss auf die Anzahl der gefundenen Zellen, da durch die Ähnlichkeitsklassen oft Pfade nicht weiterverfolgt werden, welche exklusiv eine Zelle erreichen würden. Dieser Nachteil wiegt besonders schwer, wenn ein Sprung in einen abgeschlossenen Bereich auf dem Spielfeld übersehen wird, weswegen die heuristische Optimierung nicht als Alternative in Erwägung gezogen wird.

Das Ergebnis des Vergleichs der Ansätze ist, dass die Graph-basierte Optimierung im Durchschnitt ca. 50% mehr berechnete Pfade ermöglicht. Durch diese Steigerung fällt auch indirekt die Bewertung der Kriterien K2 und K3 besser aus, sodass die Vorwärtssuche in Kombination mit der Graph-basierten Optimierung verwendet wird. Zusätzlich wird Multithreading eingesetzt, da diese Optimierung für alle Ansätze den gleichen Verbesserungsfaktor ermöglicht.

5 Entscheidungsfindung

Wie zuvor beschrieben soll eine Entscheidungsfindung entwickelt werden, die transparente Ergebnisse berechnet. Diese sind möglichst durch ein implizites Vorgehen zu lösen. Die Verwendung von explizit formuliertem Verhalten in spezifischen Situationen ist zu vermieden.

Die Entscheidungsfindung der Lösung besteht aus mehreren Bewertungsschritten, die zum Ziel haben, eine möglichst passende Aktion für die aktuelle Spielrunde zu wählen.

5.1 Spielbrettbewertung

Um die Attraktivität von Punkten auf dem Spielbrett messen zu können, wird das Spielbrett zunächst bewertet. Hierbei zeigte sich, nach der Implementierung im Prototyp, ein Verfahren zur Einschätzung von gegnerischen Aufenthaltswahrscheinlichkeiten als vorteilhaft.

Das Verhalten von gegnerischen Spielern ist, aufgrund der Unkenntnis über Strategien anderer Teams, nur beschränkt vorhersehbar. Die gegnerische Aufenthaltswahrscheinlichkeit wird also approximiert. Generell ist davon auszugehen, dass alle Teams einen gültigen Spielzug ausführen, insofern ein solcher möglich ist, da diese Prüfung sehr leicht durchführbar ist.

Die Bestimmung von gegnerischen Wahrscheinlichkeiten erfolgt über eine rekursive, nicht priorisierte Vorwärtssuche, die Pfade bis zu einer festgelegten Suchtiefe ermittelt. Die Suchtiefe gibt dabei die maximale Tiefe der Rekursion an. Auf jeder Ebene ergibt sich die Wahrscheinlichkeit für einen Pfad aus der Division von „1“ durch die Anzahl der gültigen Pfade. Dieses Zwischenergebnis muss daraufhin mit der Wahrscheinlichkeit des Eltern-Pfades multipliziert werden. So ergibt sich die neue Wahrscheinlichkeit, die für jeden (gültigen) Kind-Pfad identisch ist.

Die Wahrscheinlichkeit für den gegnerischen Aufenthalt für einen bestimmten Punkt ergibt sich aus der Summe der Wahrscheinlichkeiten aller Pfade, die diesen Punkt beinhalten. So entsteht eine Bewertung des Spielbretts als Matrix, die für jede Zelle eine Wahrscheinlichkeit für einen bestimmten Gegner enthält.
Durch dieses Vorgehen geht allerdings verloren, in welcher Runde diese Wahrscheinlichkeit gilt. Erreicht man selbst einen Punkt früher als der Gegner, so ist die Wahrscheinlichkeit seines Auftretens für den eigenen Erfolg irrelevant. Deshalb wird eine zweite Matrix berechnet, welche die minimale Anzahl an Schritten enthält, die der Gegner durchführen muss, um diesen Punkt zu erreichen.
Die beschriebenen zwei Matrizen stellen eine vereinfachte Sicht auf die möglichen Züge eines gegnerischen Spielers dar. Da allerdings bis zu sechs Spieler in einem Spiel teilnehmen können, gilt es die Ergebnismatrizen zu kombinieren. Die Wahrscheinlichkeit wird hierbei durch eine elementweise Feststellung des maximalen Werts zusammengefasst. Für die minimale Anzahl an Schritten wird die elementweise Feststellung des minimalen Werts verwendet. Als Ergebnis resultieren somit zwei Bewertungen für das Spielbrett, die eine stark vereinfachte Sicht auf die Zukunft aller gegnerischen Spieler darstellen, aber effizient ausgewertet werden können.
Um weitere Informationen über das Spielfeld bereitzustellen, wird eine heuristische Erweiterung der gegnerischen Wahrscheinlichkeit ergänzt. Für alle Elemente der Wahrscheinlichkeitsmatrix, der noch kein Wert zugewiesen wurde, wird der Wert $1/5^{n+1}$ zugewiesen, wobei 5 die Anzahl an verfügbaren Aktionen und n die eingestellte Suchtiefe für die zuvor durchgeführte Vorwärtssuche ist.

Die Matrix der minimalen Schritte wird durch ein Floodfill ähnliches Verfahren ergänzt. Als initiale Punkte des Verfahrens werden die Endpunkte der zuvor durchgeführten Suche verwendet. Die abzuarbeitenden Punkte werden iterativ durch die Punkte ergänzt, die in der Nachbarschaft der bisherigen Punkte liegen und noch kein Wert in der Matrix zugewiesen ist.

In *Abbildung 5* ist ein Beispiel für eine Matrix der gegnerischen Wahrscheinlichkeit dargestellt, wie sie aus dem entwickelten Viewer visualisiert wird. Die dazugehörige Matrix der minimalen Schritte ist in *Abbildung 6* dargestellt. In dem linken Bild ist jeweils die Ausgangssituation des roten Spielers dargestellt. Die rechten Bilder stellen jeweils die Visualisierung der Matrix dar.

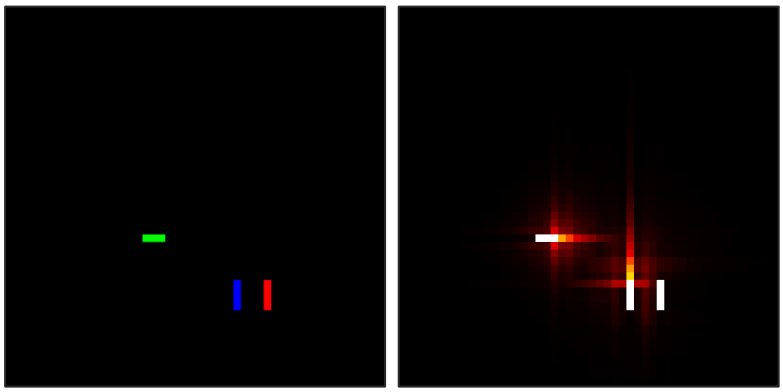

Abbildung 5: Beispiel-Matrix gegnerische Wahrscheinlichkeiten

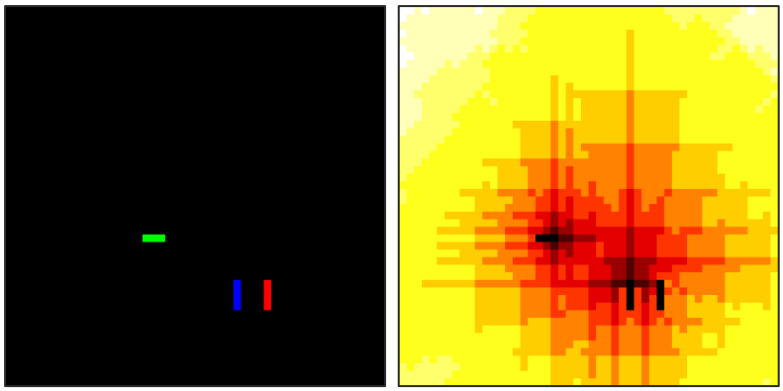

Abbildung 6: Beispiel-Matrix minimale gegnerische Schritte

5.2 Pfadbewertung

Im Mittelpunkt der zu treffenden Bewertungen steht die Bewertung der einzelnen Pfade. Die Ermittlung von Pfaden basiert, wie erörtert, auf der Vorwärtssuche in Kombination mit der Graph-basierten Optimierung und Multithreading. Für den vorliegenden Lösungsansatz werden die Pfade hinsichtlich zweier Kriterien bewertet:

- Erfolg
- Abschneiden von Gegnerpfaden

Ziel ist es, mit der Erfolgs-Bewertung Pfade schlechter zu bewerten, die von Gegnern mit einer höheren Wahrscheinlichkeit abgeschnitten werden. Die Abschneide-Bewertung ist dafür zuständig, Pfade gut zu bewerten, die gegnerische Pfade unterbrechen.

Der Erfolg eines Pfades wird definiert durch die Multiplikation der Erfolgswahrscheinlichkeiten der einzelnen Pfadelemente. Jedes Pfadelement repräsentiert den Spielzug einer Runde. Die Erfolgswahrscheinlichkeit des Pfadelements ergibt sich aus der inversen maximalen gegnerischen Aufenthaltswahrscheinlichkeit an den Punkten des Spielbretts, die in dem Pfadelement enthalten sind. Dabei muss die Anzahl der Schritte beachtet werden, die benötigt werden, bis das jeweilige Pfadelement erreicht werden kann. Da die gegnerische Wahrscheinlichkeit bereits nach wenigen Runden einen sehr geringen Wert erreicht, wird ein zusätzlicher Exponent eingeführt, der vorrangig geringe Werte der Wahrscheinlichkeit anhebt (Exponent muss kleiner als eins sein). In *Algorithmus 1* ist die beschriebene Berechnungsvorschrift dargestellt.

```
ergebnis := 1
für jedes element in pfad:
  lokal := 0
  für jeden punkt in element:
    wenn minSchritte[punkt] <= element.schritte:
      lokal := max(lokal, wahrscheinlichkeiten[punkt])
  ergebnis := ergebnis * (1-lokal^EXPONENT)
Rückgabe ergebnis
```

Algorithmus 1: Bestimmung des Erfolgs-Wertes

Eine ähnliche Berechnung ist ebenfalls für die Bewertung des Abschneidens von gegnerischen Pfaden angedacht. Hierbei wird die maximale Wahrscheinlichkeit gesucht, deren Position vor dem Gegner erreicht und blockiert werden kann. Die zugehörige Rechenvorschrift ist in *Algorithmus 2* dargestellt.

```
ergebnis := 0
für jedes element in pfad:
  für jeden punkt in element:
     wenn minSchritte[punkt] > element.schritte:
       ergebnis:= max(ergebnis, wahrscheinlichkeiten[punkt])
Rückgabe ergebnis
```

Algorithmus 2: Bestimmung des Abschneide-Wertes

5.3 Aktionsbewertung

Die Aktionsbewertung hat zum Ziel, jede der fünf möglichen Aktionen in jeder Runde zu bewerten. Jede Aktion wird dabei mit einem Wert $\in [0,1]$ bewertet. Im Folgenden werden die verwendeten Verfahren zur Bildung und Kombination von Aktionsbewertungen der vorliegenden Lösung beschrieben.

5.3.1 Erreichbare Zellen

Um Pfadbewertungen korrekt auf eine Aktionsbewertung abzubilden, muss die Pfadsuche für jede mögliche Aktion in der nächsten Runde durchgeführt werden. Hierbei ist zu beachten, dass die Kind-Pfade jeder Aktion eine ähnliche Rechenzeit erhalten. Nur so kann sichergestellt werden, dass die Aktionsbewertung nicht durch unbeabsichtigte Nebeneffekte beeinflusst wird. Dies ist insbesondere bei der Verwendung von Multithreading zu beachten.

Die Endpunkte jedes Pfades werden mit ihrer Erfolgs- und ihrer Abschneide-Bewertung in jeweils eine Matrix eingetragen. Die Matrizen werden jeweils mit 0 initialisiert und haben die Größe des Spielfelds. Jedes Element kann nur überschrieben werden, wenn der neue Wert höher als der aktuelle Wert des Elements ist. Durch dieses Vorgehen entstehen für alle fünf Aktionen jeweils zwei Matrizen:

- maximale Erfolgs-Bewertung für jeden Pfad-Endpunkt
- maximale Abschneide-Bewertung für jeden Pfad-Endpunkt

In *Abbildung 7* ist ein Beispiel für eine Erfolgsmatrix (rechts) dargestellt. Dabei wird das Spiel aus der Sicht des grünen Spielers betrachtet. Das linke Bild zeigt die Spielsituation, das mittlere Bild die gegnerische Wahrscheinlichkeit.

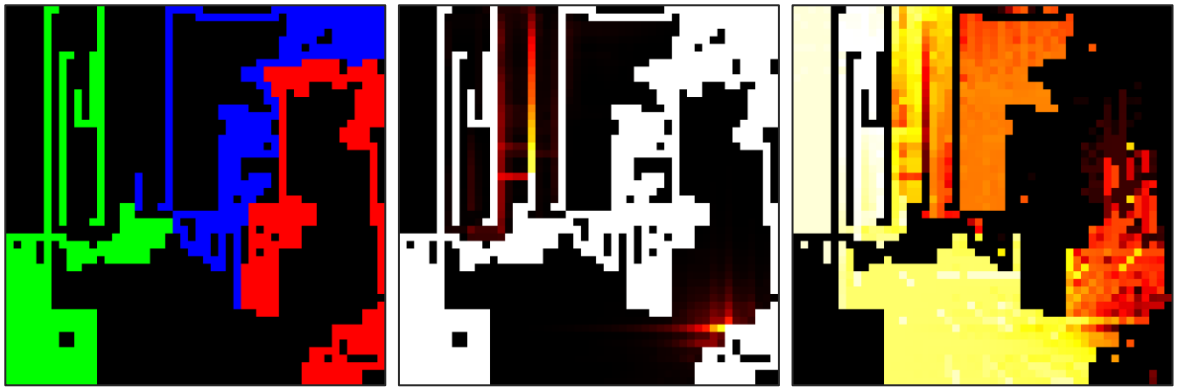

Abbildung 7: Beispiel-Matrix für die Erfolgs-Bewertung

In *Abbildung 8* ist ein Beispiel für die Abschneide-Bewertung (rechts) dargestellt. Die Ausgangssituation ist im linken Bild zu sehen. Das Spiel wird aus der Sicht des blauen Spielers betrachtet, der sich aktuell oben links befindet. Die verwendeten Wahrscheinlichkeiten sind im mittleren Bild dargestellt.

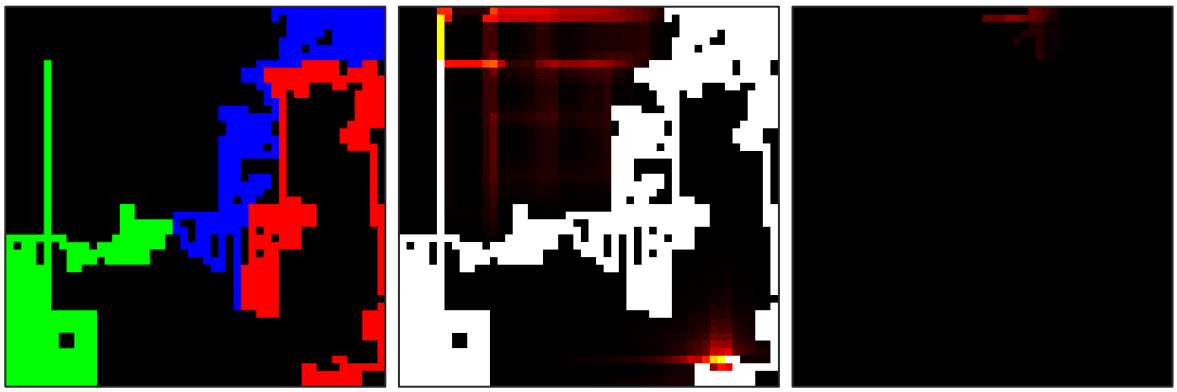

Abbildung 8: Beispiel-Matrix für die Abschneide-Bewertung

Diese Matrizen werden im nächsten Schritt auf zwei Aktionsbewertungen abgebildet. Die Erfolgs-Aktionsbewertung wird durch die Summe aller Matrixelemente der Matrix für die Erfolgs-Bewertung gebildet. Bei der Abschneide-Aktionsbewertung wird der maximale Wert der Matrix verwendet.
Für die Abschneide-Aktionsbewertung ist bereits garantiert, dass die einzelnen Bewertungen für jede Aktion zwischen 0 und 1 liegen. Um Gleiches für die Erfolgs-Bewertung zu garantieren, muss diese angepasst werden. Dafür wird jede Einzelbewertung für jede Aktion mit der höchsten auftretenden Einzelbewertung dividiert. Zur Kontrolle der Funktionsfähigkeiten können die Matrizen in der Visualisierung der Software dargestellt werden.

5.3.2 Relevanz von Zellen

Für den Fall, dass die Erfolgs-Bewertung und die Abschneide-Bewertung für alle möglichen Aktionen nur eine geringe Differenz aufweisen, wird ein weiterer Faktor betrachtet. Das Ziel dieses Faktors ist es, Zellen zu meiden, welche in der Zukunft mit hoher Wahrscheinlichkeit benötigt werden. Diese sogenannte Relevanz einer Zelle ist die Anzahl der berechneten Pfade, welche diese Zelle verwenden. Da jeder Pfad durch das Ausführen einer zusätzlichen Aktion in einem neuen Pfad resultiert, sind Zellen umso wichtiger, je früher sie verwendet werden. Diese Relevanz wird für alle Zellen bestimmt, welche innerhalb einer Runde erreichbar sind. Im Anschluss wird die Relevanz der möglichen Aktionen bestimmt, indem die Summe der Relevanz, der an diesem Zug beteiligten Zellen, berechnet wird. Dieses Ergebnis muss ebenfalls in Form einer Aktionsbewertung bereitgestellt werden, um die Ergebnisse der Berechnungen kombinieren zu können. Um die Vergleichbarkeit sicherzustellen, muss die Aktionsbewertung auf den Wertebereich der Erfolg- und die Abschneide-Bewertung normalisiert werden. Hierfür werden alle Relevanzen durch die höchste ermittelte Relevanz

dividiert. Im Anschluss muss die Relevanz invertiert werden, sodass für die Aktionsbewertung unwichtige Zellen einen höheren Wert als wichtige Zellen aufweisen. Da alle Werte auf den Wertebereich $[0, 1]$ abgebildet sind, wird für die Invertierung jeder Wert von 1 subtrahiert.

5.3.3 Kombination

Jede Bewertung kann mit einem bestimmten Gewicht mit einer anderen Bewertung kombiniert werden, indem alle einzelnen Bewertungen addiert werden. Die Kombination von Aktionsbewertungen ist eine Rechenvorschrift, bei der die Aktionsbewertungen a und b hinsichtlich der Aktion i kombiniert werden. Die Gewichtung w legt dabei fest mit welchem Faktor die Bewertungen kombiniert werden. Die resultierende Gewichtung c ergibt sich aus $c_i = a_i + w * b_i$.
Zuvor wurden bereits die drei relevanten Aktionsbewertungen beschrieben, die es passend zu gewichten gilt:

- Erfolg
- Abschneiden
- Invertierte Relevanz

Die Erfolgs-Aktionsbewertung stellt für das Spiel spe_ed die wichtigste Bewertung dar, da sie angibt, welche Punkte des Spielbretts mit welcher Erfolgsbewertung erreichbar sind. Wird hier von der optimalen Bewertung abgewichen, kann der Spieler schnell das Spiel verlieren.
Die Gewichtungen für das Abschneiden von gegnerischen Pfaden und die Gewichtung der invertierten Relevanz dürfen wichtige Unterschiede in der Erfolgs-Aktionsbewertung nicht überschreiben. Können Pfade abgeschnitten werden, sind das oft auch wichtige Pfade. Somit muss die invertierte Relevanz geringer gewichtet werden als das Abschneiden, um zu verhindern, dass das Abschneiden nicht durchgeführt werden kann.

Aus der finalen kombinierten Aktionsbewertung geht die gewählte Aktion der Runde hervor. Dafür wird die Aktion mit der höchsten Einzelbewertung gewählt. So kann sichergestellt werden, dass alle Aktionswertungen in die getroffene Entscheidung einfließen.

5.4 Deadline Management

Ein essenzieller Bestandteil der Entscheidungsfindung ist das Zeitmanagement innerhalb einer Runde. Somit ist es erforderlich, zu jedem Zeitpunkt die verbleibende Berechnungszeit feststellen zu können. Das bedeutet, die Clientzeit muss mit der Serverzeit synchronisiert werden. Dafür ist eine separate API [2, p. 5] verfügbar.

Beim Start der Anwendung wird die Synchronisation mit der Serverzeit durchgeführt. Aufgrund von mehrfach auftretenden Serverproblemen der API, die in hohen Antwortzeiten resultiert, wird der Server mehrfach für seine Zeit angefragt. Das Ergebnis mit der kürzesten Antwortzeit wird zur weiteren Synchronisation verwendet.

Die relative Verschiebung ergibt sich aus der Clientzeit t_c, der Serverzeit t_s und der Antwortzeit des Servers a durch $\Delta t = t_s - t_c + a$. Die Clientzeit wird dabei vor der Anfrage an die API erfasst.

Während des Spiels muss die Information der zeitlichen Verschiebung für das Transformieren von Deadlines verwendet werden. Für jede Deadline d_s, die vom Server empfangen wird, wird eine neue Deadline generiert, die für den Client gültig ist. Diese ergibt sich durch $d_c = d_s - \Delta t$.

Nach wenigen Tests stellt sich heraus, dass die Probleme der API des Öfteren Antwortzeiten von mehreren Sekunden ergeben. So entstehen auf dem Client ungültige (negative) Deadlines. Deshalb wird nachfolgend die nachträgliche Synchronisation mit dem Server beschrieben.

Aus der Analyse der Spielparameter geht hervor, dass mit keinen Bearbeitungszeiten von unter drei Sekunden zu rechnen ist. Da dies nur als Richtwert, nicht aber als gegeben betrachtet werden kann, wird im Folgenden von einer minimalen verfügbaren Bearbeitungszeit von einer Sekunde ausgegangen. Erhält der Client eine Deadline, die weniger als eine Sekunde Rechenzeit bietet, so ergibt sich die neue Zeitverschiebung $\Delta t'$ aus der bisherigen Zeitverschiebung Δt, der aktuellen Clientzeit t_c' und der (auf dem Client gültigen) Deadline d_c' durch

$$\Delta t' = \Delta t - ((t_c' + 1s) - d_c').$$

5.5 Ablauf

Im Folgenden wird der Ablauf der Entscheidungsfindung aufgezeigt, der sich durch das Prototyping ergibt. Dieser besteht aus den zuvor beschriebenen Elementen der Entscheidungsfindung.

Zunächst werden mit einer nicht priorisierten rekursiven Vorwärtssuche die gegnerischen Aufenthaltsorte für die nächsten Runden je Gegner bestimmt. Die Ergebnisse aller Gegner werden im Anschluss zusammengefasst. Als Suchtiefe wird 6 festgelegt, da somit maximal $5^6 = 15.625$ Pfade pro Gegner gefunden werden. Diese Anzahl ist so gering, dass hierbei auf das Sicherstellen des Einhaltens der Deadline verzichtet werden kann.

Im Anschluss erfolgt die Suche konkreter Pfade mithilfe der Graph-basierten Pfadsuche für jede mögliche Aktion in der aktuellen Runde. Parallel dazu wird die Erfolgs- und die Abschneide-Matrix für jede mögliche Aktion gebildet. Zudem wird während der Pfadsuche die Relevanz der unmittelbar erreichbaren Zellen festgestellt und im Anschluss die dazugehörige Aktionsbewertung für die invertierte Relevanz erstellt. Zur Performancesteigerung wird zur Graph-basierten Pfadsuche Multithreading verwendet.

Die Pfadsuche stellt den rechenintensivsten Arbeitsschritt der Entscheidungsfindung dar. Um sicherzustellen, dass dieser rechtzeitig fertiggestellt ist, wird solange gerechnet, bis nur noch 0,5 Sekunden der Deadline übrig sind. Dieser Puffer stellt sicher, dass nachfolgende, nicht aufwendige, Rechenschritte zeitgerecht durchgeführt werden können.

Als letzter Schritt werden die Aktionsbewertungen aus den Matrizen gewonnen. Zur Kombination der Aktionsbewertungen werden die in *Tabelle 1* dargestellten Gewichtungen verwendet. Nach der Kombination wird die Aktion mit der höchsten Einzelbewertung gewählt und als Antwort an den spe_ed Webservice gesendet.

Aktionsbewertung	Gewichtung
Erfolg	1,00
Abschneiden	0,40
Invertierte Relevanz	0,15

Tabelle 1: Gewichtungen für Aktionsbewertungen

Die dargestellten Aktionsbewertungen beruhen dabei auf begründeten Schätzungen. Die Erfolg-Aktionsbewertung stellt mit der Gewichtung von 1,0 die Grundlage der kombinierten Bewertung dar. Die Abscheide-Bewertung wird auf 0,4 gesetzt. Somit wird erlaubt, dass ein gegnerischer Pfad abgeschnitten wird, falls es die Möglichkeit dazu gibt. Mit einer Gewichtung von 0,15 soll erreicht werden, dass die invertierte Relevanz nur dann eine Rolle spielt, wenn mit jeder Aktion eine ähnliche Erfolgs- und Abschneide-Aktionsbewertung erreicht wird. Somit erhält die invertierte Relevanz die geringste Priorität ist jedoch erforderlich, um sicherzustellen, dass wichtige Pfade nicht ohne Grund abgeschnitten werden.

5.6 Erwartetes Verhalten

Der zuvor beschriebene Ablauf dient dem Einbezug von gegnerischem Verhalten, sowie der Vorhersage des eigenen Verhaltens über mehrere Runden hinweg. Im Nachfolgenden wird das erwartete Verhalten beschrieben, das der Begründung der zuvor aufgeführten Entscheidungsfindung dient. Dafür werden Situationen beschrieben, die im Verlauf eines Spieles auftreten können. Durch den Umgang wird beschrieben, wie das erhoffte Verhalten der Entscheidungsfindung zu einer angebrachten Konsequenz führt.

Szenario 1	**Beschreibung**
Ausgangssituation	Der Spieler ist in einem kleinen Bereich des Spielfeldes, ein kritischer Teil des Spielfeldes ist bedroht von einem anderen Spieler abgeschnitten zu werden, sodass dieser im späteren Verlauf unerreichbar bleibt.
Umgang	1. die Wahrscheinlichkeit der gegnerischen Pfade ist an der kritischen Stelle erhöht 2. Pfade, die durch die kritische Stelle verlaufen, werden geringer im Erfolg bewertet, wenn der Gegner vor dem Spieler dort ist 3. Aktionen, die mehr Pfade enthalten, die die kritische Stelle vor dem Gegner erreichen, werden besser bewertet
Konsequenz	Spieler bewegt sich in Richtung der betroffenen Stelle

Szenario 2	Beschreibung
Ausgangssituation	Der Spieler befindet sich in der Nähe eines gegnerischen Spielers. Der Gegner wird in den nächsten Runden eine bestimmte Stelle passieren müssen.
Umgang	1. die Wahrscheinlichkeit der gegnerischen Pfade ist in an der kritischen Stelle erhöht 2. Pfade, die kritische Punkte der gegnerischen Wahrscheinlichkeit vor einem Gegner berühren, erhalten eine höhere Abschneide-Bewertung 3. Aktionen, die den besten Pfad zum Abschneiden des Gegners enthalten, werden höher bewertet.
Konsequenz	Spieler bewegt sich in Richtung der abschneidbaren Stelle

Szenario 3	Beschreibung
Ausgangssituation	Der Spieler befindet sich nicht in der Nähe von gegnerischen Spielern in einem bereits abgeschnittenen Bereich.
Umgang	1. die Wahrscheinlichkeit der gegnerischen Pfade ist in den relevanten Bereichen mit „0“ belegt 2. für jede Aktion werden ähnliche viele und ähnlich gute Pfade gefunden 3. Aktionsbewertung des Erfolges für die meisten Aktionen nahezu gleich 4. Aktionen, die viele relevante Pfade abschneiden, erhalten eine schlechte invertierte Relevanz
Konsequenz	Spieler bewegt sich langsam, schneidet keine wichtigen Pfade ab → längeres Überleben

Szenario 4	Beschreibung
Ausgangssituation	Der Spieler steht vor einer Abzweigung: Er kann entweder in einen großen oder einen sehr kleinen Bereich navigieren. Navigiert er in den großen Bereich, riskiert er eine Kollision mit einem Gegner.
Umgang	1. die Wahrscheinlichkeit der gegnerischen Pfade ist in Richtung des großen Bereichs hoch 2. Pfade, die in den großen Bereich gehen werden, geringer gewertet in ihrem Erfolg, aber es werden mehr Pfade im großen Bereich gefunden
Konsequenz	Kollision wird nicht prinzipiell ausgeschlossen, sondern das Risiko wird bewusst eingegangen, wenn so deutlich mehr Pfade erreicht werden können

Die beschriebenen Situationen und den Umgang mit diesen bilden eine Grundlage für die Analyse der Funktionsfähigkeit und für die spätere Evaluation des theoretischen Ansatzes.

6 Softwarearchitektur und -qualität

Die verwendete Softwarearchitektur ist ein Ergebnis, welches auf den Vorgaben des InformatiCup-Wettbewerbs und den Erkenntnissen des bereits beschriebenen *Prototypings* basiert. Das Nachfolgende gilt der Beschreibung der Architektur auf makroskopischer Ebene. Die feinere Dokumentation der Architektur erfolgt zur besseren Lesbarkeit im Quellcode selbst in Form von dokumentierenden Kommentaren und als separate, automatisiert generierte HTML Quellcodedokumentation.

6.1 Architekturanforderungen

Durch die Prototyp-Entwicklung mit der Programmiersprache *Python 3* sind konkrete und vielversprechende Lösungsansätze hervorgegangen und es können Anforderungen an die Architektur und Funktionalität des Ergebnisses spezifiziert werden. Zum einen gibt es Funktionalitäten, welche durch die Entwicklung mit *Python 3* möglich sind und in der zu verwendenden Architektur auch unterstütz werden müssen, zum anderen gibt es Funktionalitäten, welche nicht durch *Python 3* unterstützt werden, jedoch für die festzulegende Architektur erforderlich sind. Im Folgenden sind die Architektur-Anforderungen AA1 – AA7 aufgeführt. Die Wichtigkeit der Anforderungen ist durch die Ein-Kriteriums-Klassifikation (↑ Obligatory, – Mandatory, ↓ nice-to-have) gekennzeichnet. [7, p. 128]

Aus der Aufgabenstellung resultierende Architektur-Anforderungen

AA1: Kommunikation über WebSockets	↑
Durch die Aufgabenstellung ist eine Anforderung an die Architektur die Möglichkeit der verschlüsselten Kommunikation über WebSockets mit dem Spielserver.	

AA2: Über Docker bereitstellbar	↑
Durch die Aufgabenstellung ist eine Anforderung an die Technologieauswahl, dass das Ergebnis als *Docker Image* in einem *Docker Container* lauffähig sein muss. Alle verwendeten Technologien der Gesamtarchitektur müssen dieser Anforderung nachkommen.	

Aus dem Prototyping resultierende Anforderungen

AA3: Multithreading ermöglichen	↓
Da das Lösen der Aufgabe im Sinn der Komplexitätstheorie als Optimierungsproblem angesehen werden kann, hat die verfügbare Rechenleistung Einfluss auf das Ergebnis. Durch die Zusicherung des Wettbewerb-Teams, dass die Referenzsysteme „moderne Mehrkernprozessor-Computer" sind [6], muss die Architektur ermöglichen die vorhandenen Kerne des Prozessors auszunutzen.	

AA4: Einfache Anbindung externer APIs	–
Die Integration externer Programmbibliotheken soll durch die Architektur ermöglicht werden, sodass Abhängigkeiten und deren Lizenzen strukturiert und einfach verwaltet werden können.	

AA5: Hohe und Stabile Performance	↑
Das Prototyping mit *Python 3* zeigt eine Schwäche bei der Perfomance während der Ausführung auf, da die Skriptsprache langsamer als Programmiersprachen mit JIT-Compiler ist. Aus diesem Grund muss eine etablierte Technologie mit möglichst hoher Performance verwendet werden.	

AA6: Vererbung und Polymorphie	**–**
Die stark dynamische Typisierung, Mehrfachvererbung und fehlenden expliziten Zugriffsmodifikatoren in *Python 3* sind im Kontext des Prototypings sinnvoll und beschleunigen das Produzieren von Quellcode, für eine stabile und wartbare Software wird jedoch auf eine klassische objektorientierte Technologie gesetzt. Das Ziel hierbei ist eine modulare, strukturierte Software zu erstellen, welche es durch Datenkapselung und Wiederverwendung ermöglicht verschiedene Implementierungen zu vergleichen.	

AA7: Trennbarkeit von Bestandteilen	**↓**
Die Trennung und Austauschbarkeit von Bestandteilen sind wichtig, um das System ohne großen Aufwand ändern zu können. Da das System kontinuierlich getestet und verbessert wird, ist es von Bedeutung, dass die einzelnen Komponenten klar getrennt und austauschbar sind. Hierfür bieten sich verschiedene Konzepte von objektorientierten Programmiersprachen an.	

6.2 Technologie Auswahl

Als Alternativen für die Technologieauswahl werden *Java 11*, *C#9 im .NET 5.0 Framework* und *C++* betrachtet. Trotz der Popularität von Python 3 wird diese Technologie nicht betrachtet, da sie als Ergebnis des Prototypings als unzureichend eingestuft wird. Dies beruht auf dem in Python verwendeten Global Interpreter Lock (GIL), das Multithreading erheblich erschwert [8]. Für das Multiprocessing zeigten sich im Prototyping bereits Performanceprobleme. Die Performance von Python in einem einzigen Thread wurde als nicht ausreichend wahrgenommen.

Eine Vorauswahl der Technologie erfolgt durch die Spezifikation von K. O. Kriterien, welche erfüllt werden müssen. Diese Kriterien sind *AA1: Kommunikation über WebSockets* und *AA2: Über Docker bereitstellbar* aufgrund der Wettbewerbsregeln.

Da die Technologien sehr ähnlich sind, erfüllen diese die Architekturanforderungen auch gleichermaßen. Lediglich C++ erlaubt eine höhere Performance, jedoch auch die Notwendigkeit einer manuellen Speicherverwaltung. Der resultierende, höhere Aufwand bei der Entwicklung, ist ein Nachteil von C++, sodass die Wahl auf Java oder C# fällt.

Aufgrund von persönlichen Kenntnissen der Teammitglieder wird Java als Programmiersprache gewählt. Java wurde in einer Vorlesung behandelt, womit eine gemeinsame Grundlage für die Programmierkenntnisse gesichert ist. Des Weiteren bietet Java eine Plattformunabhängigkeit, womit auch eine weitläufige Verwendung des Programms außerhalb des Docker Containers sichergestellt ist.

6.3 Technologie Stack

Neben der Kerntechnologie sind weitere Tools erforderlich. In *Tabelle 2* sind die verwendeten Tools und externen Abhängigkeiten aufgeführt. Für alle externen Abhängigkeiten ist sichergestellt, dass diese über Lizenzen verfügen, die die Verwendung im Rahmen des Vorhabens erlauben.

Aufgabe	Auswahl
Versionsverwaltung	Git über GitHub
Build-Management-Tool	Apache Maven
Unit Testing	JUnit4
Kommandozeilenargumente	picocli
JSON Parsing	gson
Websocket	Jetty Websocket

Tabelle 2: Technologie Stack

Die Auswahl des Technologies Stacks erfolgte nach dem Kriterium der Vertrautheit der Teammitglieder mit den entsprechenden Tools. Zur Auswahl wurden keine weiteren Kriterien betrachtet, da die Verwendung der Tools bereits ausreichende Ergebnisse ermöglicht.

6.4 Modulare Architektur

Die Software wird in mehrere Module untergliedert, die jeweils einen zusammenhängenden Funktionalitätsbereich darstellen. Somit wird eine höhere Wartbarkeit gewährleistet, da einzelne Module bei Bedarf ausgetauscht werden können.

Die entwickelten Module sind in Tabelle 3: Aufteilung der Software in Module*Tabelle 3* dargestellt. Durch die vorliegende Aufteilung der Software in die einzelnen Module ergibt sich eine weitgehende gegenseitige Unabhängigkeit der Module. Lediglich das unterstützende „utility" Modul wird von allen operativen Modulen verwendet, da es wichtige Austauschformate beinhaltet und somit die Kommunikation zwischen den Modulen ermöglicht. Das „core" Modul steuert die Anwendung und ist deshalb direkt von den operativen Modulen abhängig.

Modul	Aufgaben
core	• Parsen von Umgebungsvariablen • Parsen von Kommandozeilenargumenten • Starten der Anwendung mit richtigen Parametern
solver	• Entscheidungsfindung
visualisation	• Verwaltung der grafischen Benutzeroberfläche
simulation	• Offline-Simulation des Spiels
web-communication	• Online-Durchführung des Spiels • Synchronisation mit Serverzeit • Kommunikation mit spe_ed Webserver
utility	• geteilte Datentypen und Funktionalitäten aller Module • Bereitstellen von Austauschformaten

Tabelle 3: Aufteilung der Software in Module

In *Abbildung 9* sind die Abhängigkeiten der Module in UML-Notation dargestellt. Durch die Architektur in der gezeigten Schichten-Form sind die Module „solver“, „visualisation“, „simulation“ und „web-communication“ leicht austauschbar, da jeweils nur das „core“ Modul von ihnen abhängig ist. Das „core“ Modul ist somit für das Initialisieren des Kontrollflusses und für die Kooperation der Module in der Anwendung zuständig.

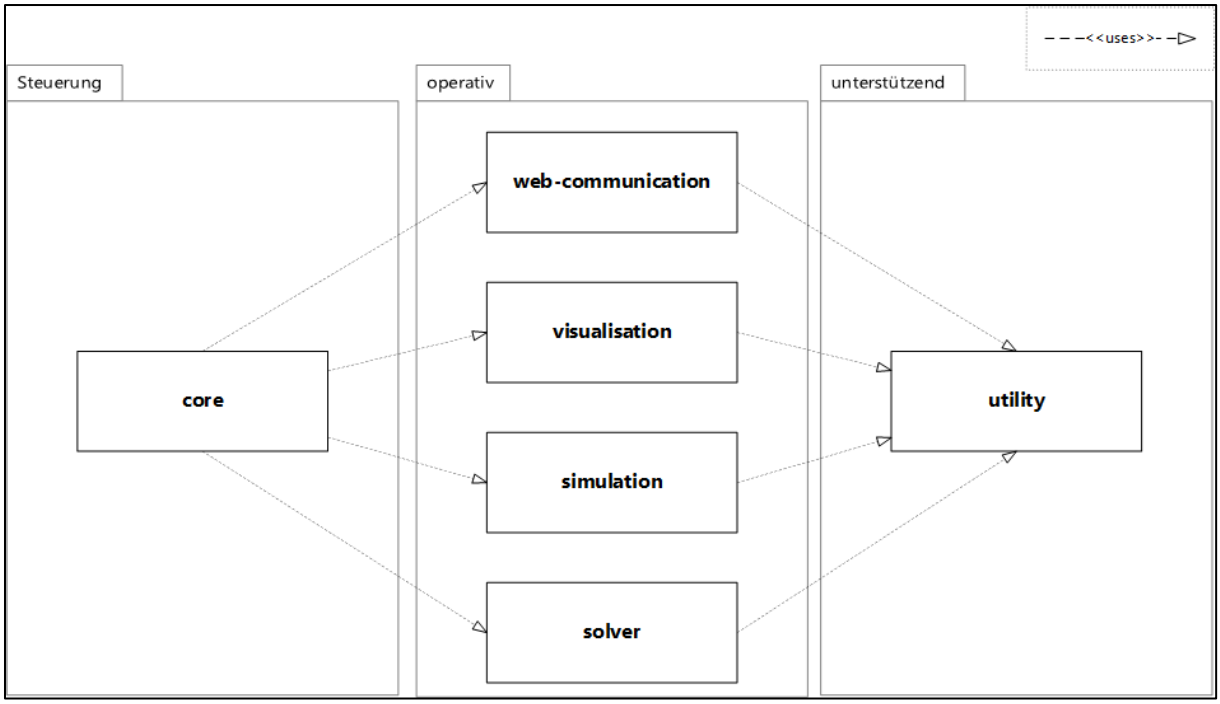

Abbildung 9: Abhängigkeiten der Module

6.5 Software Testing

Das Testen der Software stellt einen essenziellen Kontrollprozess der Entwicklung der Software dar. Nur durch ausreichende Tests kann die Funktionalität der Anwendung sichergestellt werden. Für das vorzustellende Testkonzept werden Erfahrungen aus dem Prototyp verwendet.

Funktionen des „utility" Moduls werden in allen weiteren Modulen verwendet. Während der Entwicklung des Prototyps zeigt sich, dass gefundene Fehler in dem Aufgabenbereich des Moduls in einem sehr hohen Suchaufwand resultieren. Zudem wird das „utiltity" Modul häufig verändert und Funktionalitäten geändert. Aufgrund der besonderen Wichtigkeit des Moduls und dem hohen Behebungsaufwand von Fehlern werden für alle Klassen des Moduls automatisierte Tests geschrieben.

Das „core" Modul ist vor allem für das Starten der Anwendung und den Programmfluss in Verwendung. Bereits im Prototyp zeigt sich, dass hier auftretende Fehler schnell erkannt und behoben werden können. Zudem werden nur minimale Änderungen des Moduls während der Entwicklung durchgeführt. Manuelle Tests, z.B. für falsche Benutzereingaben sind aufgrund der Verwendung der externen Abhängigkeit für die Kommandozeile ohnehin durchzuführen. Eine automatisierte Testabdeckung würde hier also nur bereits getestetes Verhalten validieren und einen erheblichen Mehraufwand für die Entwicklungszeit bedeuten, weshalb für das „core" Modul manuelle Tests angedacht sind. Um den Prozess zu beschleunigen werden in der verwendeten IDE *run configurations* mit bestimmten Umgebungsvariablen und Kommandozeilenargumenten angelegt.

Das „visualisation" Modul dient lediglich der Beschleunigung des Entwicklungsprozesses. Es ist davon auszugehen, dass Visualisierungsfehler direkt erkannt werden und leicht zu beheben sind. Dies wird bestätigt durch die entwickelte Visualisierung des Prototyps. Separate Tests sind hier ausdrücklich nicht

angedacht, da das Modul selbst während des gesamten Entwicklungsprozesses in einem einzigen Modus getestet wird und im Endergebnis (Docker Container) deaktiviert ist.

In der Simulation ist vorrangig darauf zu achten, dass das Spielerverhalten mit den spe_ed Spielregeln kompatibel ist. Deshalb ist dieses durch automatisierte Tests abzudecken. Weitere Verhaltensweisen sind oft über Zufälle gesteuert, wodurch eine automatisierte Testung hier nicht zielführend ist. Da jedes Simulationsspiel ohnehin ausführlich analysiert wird, ist davon auszugehen, dass solche Fehler dort gefunden werden. Dies bestätigte sich ebenfalls durch die Prototypentwicklung. Gleiches gilt für die unabdingbare Notwendigkeit, das Spielerverhalten automatisiert zu testen.

Tests in der Kommunikation mit dem Webserver müssen bereits manuell durchgeführt werden, um die Funktonalität des fertigen Docker Images sicherzustellen. Zudem wird die Kommunikation bei jeder Auswertung des Spielverhaltens auf der Online API hinsichtlich der Funktionalität getestet. Eine automatisierte Testabdeckung ist hier vor allem für Randfälle sinnvoll, die seitens des Servers nicht zu erwarten sind. Das gilt z.B. für falsch formatierte oder ungültige JSON Objekte.

Das „solver" Modul beinhaltet die eigentliche Lösung des spe_ed Problems und kontrolliert die Entscheidungsfindung. Bereits kleine Änderungen an der Programmierung können eine erhebliche Änderung am Spielverhalten verursachen. Diese Veränderungen sind nicht automatisiert erfassbar, sondern müssen manuell, mit Hilfe des Viewers und der Logs nachvollzogen und evaluiert werden. Dennoch ist es wichtig, die korrekte Funktion von oft benutztem Kernverhalten durch automatisierte Tests sicherzustellen.

6.6 Coding Conventions

Die Verständlichkeit des Quellcodes sowie dessen Wartbarkeit kann durch den sinnvollen Einsatz von Coding Conventions erhöht werden.

Neben der Einhaltung der Empfehlungen der *Java language specification* [9] und dem *Java Style Guide* von Google [10] werden für das Projekt weitere Konventionen definiert.

Die Umsetzung dieser Konventionen wird einerseits durch Code-Reviews nach dem 4-Augen-Prinzip sichergestellt, andererseits wird das Tool *SonarLint* zur Statischen-Codeanalyse eingesetzt, welches die Einhaltung von unterstützten Konventionen überprüft.

Naming Conventions

- Die gesamte Code-Basis (inklusive Kommentare) ist ausschließlich in englischer Sprache verfasst
- Keine Verwendung von Abkürzungen in Kommentaren oder als variablen Namen
- Interfaces: Um Interfaces von Elementen mit implementierter Logik unterscheiden zu können, beginnen diese mit dem Großbuchstaben „`I`".
- Generics: Für generische Typen auf Klassen-Ebene wird die Konvention für Klassennamen verwendet. Generics auf Methoden-Ebene dürfen zusätzlich ein einzelner Großbuchstabe sein.
- Unit Tests: Der Name von Test-Klassen endet mit „Test". Die enthaltenen Test-Funktionen beginnen mit „test" gefolgt von dem Namen der zu testenden Funktion.

Dokumentation

- Jede Funktion, welche außerhalb der umgebenden Klasse zugreifbar ist, muss durch einen JavaDoc-Kommentar beschrieben werden. Das schließt auch Argumente, potenziell geworfene Exceptions, sowie die Rückgabe ein.

Clean Code

Damit der Quellcode intuitiv verständlich ist, werden diverse Regeln befolgt:

- Das Schlüsselwort final muss für alle Variablen, welche nicht verändert werden sollen, verwendet werden.
- Die Anzahl der Zeichen in jeder Zeile darf 120 nicht überschreiten.
- Die Komplexität von Methoden und die Anzahl der Argumente soll bei maximal 15 liegen.
- Inline Kommentare sind, wenn möglich, zu vermeiden und durch leicht verständlichen Quellcode zu ersetzen.

Exceptionhandling

Um Fehler zu vermeiden und wenn nötig nachvollziehen zu können, ist es erforderlich den Umgang mit (unerwarteten) Fehlern festzulegen. Hierfür wird ein Logging-Konzept verwendet, welches es erlaubt, die Ausgaben in der Konsole einzuschränken, erweiterte Debug-Informationen anzuzeigen, Ausgaben in einer Log-Datei zu persistieren und unerwartete Fehler abzufangen, sodass benutzerfreundliche Meldungen ausgegeben werden und technische Fehlerbeschreibungen nur in der Log-Datei verfügbar sind.[6]

[6] Die Verwendung der Logging-Komponente ist im Handbuch näher beschrieben.

6.7 Wartbarkeit

Die Anpassung und Erweiterung des Systems sind aufgrund der verwendeten Technologien gegeben. Die modulare Struktur des Maven-Projekts ermöglicht es, einzelne Komponenten separat zu betrachten, anzupassen und auszutauschen. Die Verfügbarkeit und Aktualität von externen Komponenten wird über die Plattform *GitHub* mithilfe des sogenannten *Dependabot* sichergestellt.
Durch die vorgestellten Coding Conventions ist die Dokumentation des Quellcodes sichergestellt und durch die Verwendung des Versionsverwaltung-Tools *Git* können einzelne Änderungen und deren Begründung nachverfolgt werden.
Durch Tests in den zuvor beschriebenen Bereichen ist sichergestellt, dass zukünftige Änderungen am System keine neuen Fehler verursachen.
Eine nachträgliche Anpassung des Systems wird durch diese Technologien unterstütz und kann aufgrund der Dokumentation und den Coding Conventions auch von projektfremden Personen durchgeführt werden.

6.8 Continuous Integration

Um die Qualität des Systems sicherzustellen, werden die folgenden Aspekte von Continuous Integration eingesetzt. [11]

- Verwendung eines einzelnen Repositories und geteilter Dokumente, um gemeinsam am Quellcode zu arbeiten.
- Mithilfe von GitHub Actions wird bei einer Änderung des Quellcodes
    a) die Software kompiliert
    b) alle Unit-Tests ausgeführt
    c) ein DockerImage auf der Plattform DockerHub veröffentlicht
    d) die Dokumentation des Quellcodes aktualisiert

7 Auswertung

Im nachfolgenden wird die Evaluation des theoretischen Ansatzes durchgeführt. Dafür wird zunächst die generelle Funktionsfähigkeit geprüft. Im Anschluss werden Spiele auf der Online API des Wettbewerbs zum Vergleich gegen andere Teams analysiert.

7.1 Szenarien

In *Kapitel 5.6* wurden bereits bestimmte Szenarien beschrieben und der erhoffte Umgang mit diesen geschildert. Im Rahmen der Evaluation sollen diese Szenarien hinsichtlich ihres Auftretens in real durchgeführten Spielen auf der bereitgestellten Online API, geprüft werden.

Die Szenarien *„Szenario 1“*, *„Szenario 2“*, *„Szenario 3“* und *„Szenario 4“* können in den Spielen nachgewiesen werden. Videos von deren Auftreten sind über *Anhang 3* einzusehen. Durch die Visualisierung wird auch die Funktionsfähigkeit der Entscheidungsfindung nachgewiesen.

7.2 Vergleich mit anderen Teams

Über die bereitgestellte Online API des Wettbewerbs ist es möglich, gegen andere Spieler anzutreten. Am Ende des Spiels wird der Name aller beteiligten Teams angezeigt, so sind Spieler über einen gewissen Zeitraum hinweg nachverfolgbar. [2, p. 3] Aus mehreren Spielen ging hervor, dass ein Spiel der Online API oft mit Bots aufgefüllt wird. Diese Bots zeigen meist ein primitives Verhalten, haben aber ebenfalls einen Namen.

Zur Auswertung werden Spiele über einen längeren Zeitraum hinweg dokumentiert. Dabei wird vor vermerkt, wie oft ein anderes Team früher oder später ausscheidet im Vergleich zu Team Lehnurr. Zudem erfolgt für jedes Spiel und jeden Spieler eine Einschätzung, ob es sich bei dem Spieler um einen Bot der API handelt. Ziel der Auswertung ist es, die Gewinnrate gegenüber anderen Teams[7] zu vergleichen und die allgemeine Gewinnrate[8] festzustellen.

In *Abbildung 10* sind die Platzierungen in den durchgeführten Testspielen dargestellt. In Summe wurden 35 Testspiele durchgeführt, von denen 30 Mal der erste Platz belegt werden konnte. In fünf Fällen wurde der zweite Platz belegt. Nahezu alle Spiele beinhalteten mindestens einen echten Gegner.

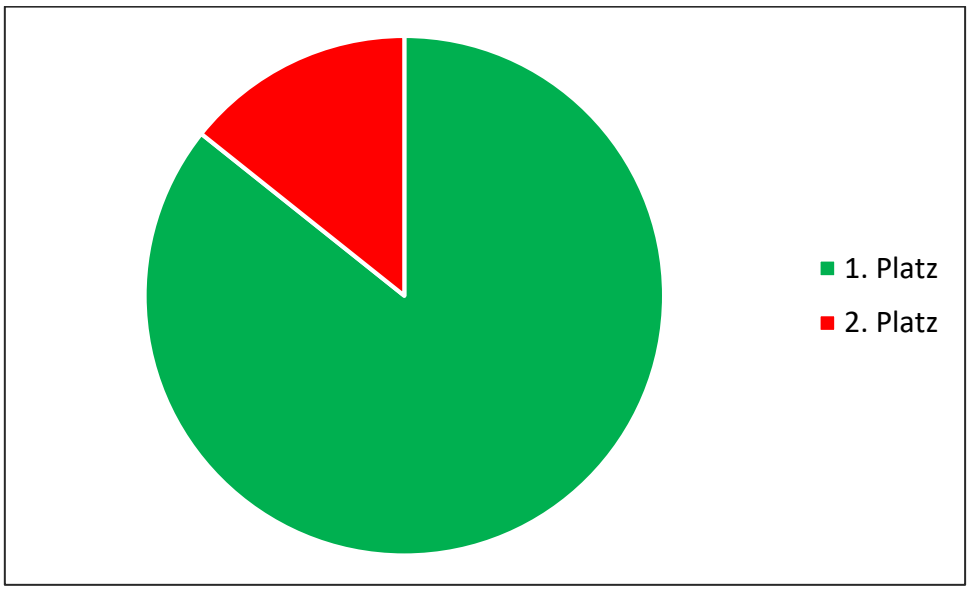

Abbildung 10: Platzierungen in den Testspielen

[7] Anzahl der Spiele, in denen das eigene Team besser war als das gegnerische Team, im Vergleich zur Anzahl der Spiele, die gegen das gegnerische Team gespielt wurden

[8] Anzahl gewonnener Spiele im Vergleich zur Anzahl der gespielten Spiele

In *Abbildung 11* ist die Gewinnrate im Vergleich zu den besten gegnerischen Teams dargestellt. Dies beinhaltet alle Gegner, die ein Spiel gewonnen haben und Gegner, die nicht als API-Bot eingeschätzt werden. Die schlechteste Gewinnrate liegt in zwei Fällen bei 50% für jeweils vier Spiele, die gegen die Teams durchgeführt wurden.

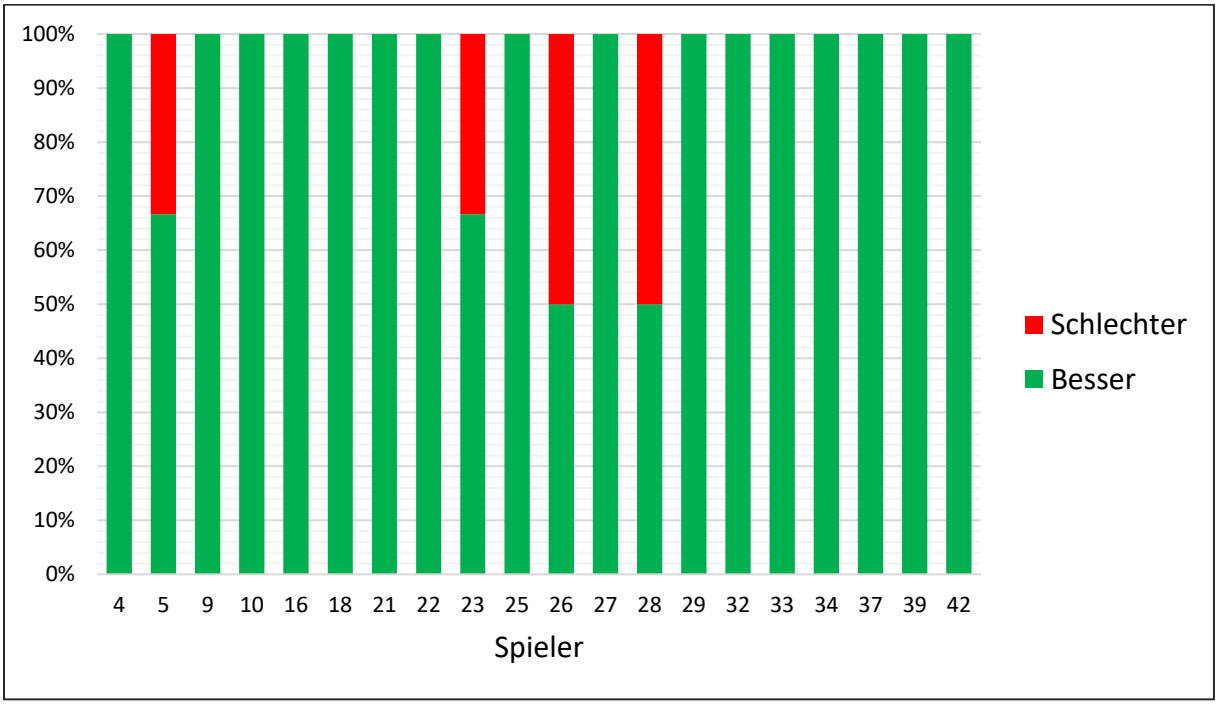

Abbildung 11: Vergleich der Lösung zu gegnerischen Spielern

Timeouts[9] wurden aus den Berechnungen ausgeschlossen, da davon auszugehen ist, dass diese nicht das echte Spielerverhalten der gegnerischen Teams darstellen.

[9] Fälle, in denen ein gegnerischer Spieler keine Aktion vornimmt, obwohl er gültige Aktionen durchführen könnte

Es ist darauf hinzuweisen, dass die Gewinnraten gegen verschiedene Spieler, sowie die Qualität der Lösung im Allgemeinen, in ihrer Genauigkeit begrenzt sind. Störungsfaktoren sind dabei vor allem die mehrfach auftretenden Timeouts, die API-Bots und die dadurch resultierende geringe Anzahl an echten Spielern in einem Spiel. Des Weiteren ist nicht ausgeschlossen, dass Teams, die eine Taktik verwenden, welche leicht gekontert werden kann, die Online API nicht nutzen, um ihre Lösung geheim zu halten.
Die gesammelten Daten der gesamten Auswertung und die verwendeten Rechenvorschriften sind in *Anhang 4* enthalten und können unter dem dort angegebenen Link eingesehen werden.

8 Ausblick

Die erhaltenen Ergebnisse der vorausgegangenen Auswertung sind nur als vorläufiger Zwischenstand anzusehen, da die Lösungen anderer Teams nicht bekannt sind. Der nächste Schritt einer möglichen Erweiterung besteht also in der Analyse der Lösungen anderer Teams. Dennoch gibt es auch in der vorliegenden Lösung Erweiterungspotenzial.

Die Gewichtungen der Aktionsbewertungen bestehen derzeit auf begründeten Schätzungen und einigen Tests im Prototyp. Durch evolutionäre Verfahren können hierfür bessere Gewichtungen gesucht werden. Dafür ist zwingend erforderlich, dass Gewichtungen auf realistisch-handelnden gegnerischen Verhaltensweisen aufgebaut werden.

Auch durch die Optimierung bestimmter Algorithmen (z.B. zur Pfadsuche oder zur Berechnung der Wahrscheinlichkeiten) kann eine Performancesteigerung und somit auch eine Verbesserung der Lösungsqualität erreicht werden.

Durch den modularen Aufbau der Softwarearchitektur, ist die Erweiterung der Software leicht möglich. Auch können leicht neue Spielertypen mit neuen Verhaltensweisen in die Software integriert werden. Dies wird unterstützt durch die JavaDocs, die den Quellcode dokumentieren.

9 Fazit

In dieser Ausarbeitung wurde die Entwicklung eines Lösungsverfahrens für das Spiel spe_ed im Rahmen des InformatiCup 2021 entwickelt. Nach einer initialen Modellierung und Analyse des Spiels und seiner Parameter wurde ein theoretischer Ansatz zur Lösung des Spiels erarbeitet.

Um die schnelle Evaluation von Lösungsideen auszuwerten wurde ein Prototyp entwickelt, der über eine Visualisierung des Spiels verfügt. In Kombination mit einer Nachbildung des Spielverhaltens in einer Simulation kann so der Entwicklungsprozess erheblich beschleunigt werden.

Aufgrund der gewünschten Transparenz und Nachvollziehbarkeit der Lösung wurde eine heuristische Entscheidungsfindung entwickelt, die mögliche Spieleraktionen hinsichtlich ihres Erfolgs und ihrer Chance gegnerische Pfade abzuschneiden bewertet. Kern der Bewertung ist die Graph-basierte Pfadsuche, die das effiziente Suchen nach möglichen zukünftigen Pfaden ermöglicht.

Der theoretische Ansatz wurde in einem Java Programm umgesetzt. Durch einen modularen Aufbau wird die Erweiterbarkeit der Software erleichtert und der Entwicklungsprozess erheblich beschleunigt. Mit Hilfe von GitHub Actions wird bei jeder Änderung automatisiert die Software getestet, ein Docker Image erstellt und eine Dokumentation des Projekts erstellt.

Anhand von durchgeführten Spielen auf der bereitgestellten Online API des Wettbewerbs, kann die Eignung der entwickelten Lösung gegen andere Teams aufgezeigt und evaluiert werden. Das erhoffte Verhalten der Entscheidungsfindung ist ebenfalls validiert und anhand von Spielen gegen andere Teams nachgewiesen.

10 Handbuch

In diesem Handbuch werden der Installationsprozess und die Benutzerschnittstellen des spe-ed-solvers von Team Lehnurr für den InformatiCup:2021 beschrieben. Theoretische und technische Hintergründe sind nicht enthalten und können über die folgenden Wege in Erfahrung gebracht werden:

Quellcode	GitHub Repository: lehnurr/spe-ed-solver
Dokumentation des Quellcodes	https://spe-ed-docs.lehnurr.de
Theoretische Ausarbeitung und Architektur	In GitHub repository lehnurr/spe-ed-solver/elaboration
Informationen über das Team	https://team.lehnurr.de
Individuelle Anfragen	Per E-Mail an team@lehnurr.de

Inbetriebnahme

Der spe-ed-solver kann in einem Docker-Container oder in einer Java-11 Umgebung ausgeführt werden.

Docker

> ! Für das Erstellen und Ausführen eines Docker Images muss Docker auf Ihrem System installiert sein.

Über die Plattform DockerHub kann ein Docker Image des aktuellen Entwicklungsstands mit

```
docker pull teamlehnurr/spe-ed-solver:latest
```

abgerufen werden.

Es besteht auch die Möglichkeit, ein neues Docker Image zu erstellen. Hierfür wird das Repository des spe-ed-solvers benötigt. In dem Verzeichnis „.docker“ sind zwei Dockerfile’s enthalten:

- „Dockerfile“: Setzt 4 verfügbare Threads voraus
- „DockerfileOneThread“: Erlaubt das Ausführen des spe-ed-solvers mit einem einzelnen Thread.

Um ein Docker Image zu erstellen, muss darauf geachtet werden, dass mit dem Befehl „docker build“ der korrekte PATH-Kontext zum Stammverzeichnis des Repositories und mit FILE das gewünschte Dockerfile definiert wird[10]. Um dies zu erleichtern, steht die Datei „imageBuild.bat“ bereit, welche den Befehl

```
docker build --pull --rm -f "Dockerfile" -t teamlehnurr/spe-ed-solver:latest ".."
```

im Verzeichnis „.docker“ ausführt. Das Vorgehen, um ein DockerImage für die Ausführung mit einem Thread zu erstellen, ist analog hierzu.

Beim Ausführen eines DockerImages muss beachtet werden, dass die Umgebungsvariablen URL und API_KEY mit korrekten Werten definiert sein müssen. Das zusätzliche Angeben einer TIME_URL wird empfohlen, ist jedoch nicht zwingend erforderlich.

Der einfachste Weg das erstellte DockerImage auszuführen ist:

1. Eine Kopie „variables.local.env“ der Datei variables.repo.env erzeugen
2. In „variables.local.env“ die Werte der Umgebungsvariablen anpassen
3. imageRun.bat ausführen

   ```
   docker run --env-file="variables.local.env" teamlehnurr/spe-ed-solver
   ```

Alternativ können die Umgebungsvariablen beim Starten manuell über die Kommandozeile angegeben werden.

[10] Der Standardwert für FILE ist „DockerFile“ und der Standardwert für den PATH-Kontext ist das Verzeichnis, in welchem der Befehl ausgeführt wird.

Java 11

Das Erstellen der Software mit Java erfordert eine Java Entwicklungsumgebung der Version 11 oder höher und Apache Maven. Weitere Abhängigkeiten werden automatisch von Maven bereitgestellt, sodass mit dem Befehl

```
mvn package -f=spe-ed-solver
```

im Verzeichnis des Repositories die Software erstellt werden kann. Hierdurch wird im Verzeichnis `spe-ed-solver/core/target` die ausführbare Datei `core-<VERSION>-jar-with-dependencies.jar` erstellt.

Auf eine Installation von Maven kann verzichtet werden, wenn die bereits kompilierte `core-1.0-jar-with-dependencies.jar` Datei von GitHub verwendet wird.

Um die Anwendung zu starten muss eine Java runtime environment für die Version Java 11 oder das Java Development Kit für Java 11 auf dem System verfügbar sein. Unter dieser Voraussetzung kann die `*.jar`-Datei mit dem Befehl

```
java -jar core-1.0-jar-with-dependencies.jar
```

gestartet werden. Die notwendigen Umgebungsvariablen URL und API_KEY können entweder beim Starten über die Kommandozeile angegeben werden, oder als tatsächliche Umgebungsvariable des Systems bereitstehen. Die optionale TIME_URL kann ebenfalls über beide Wege angegeben werden.

Abhängigkeiten

Durch das Build-Management-Tool Maven sowie die Technologie Docker ist es für das Erstellen der Software nicht erforderlich, sich mit weiteren Abhängigkeiten auseinanderzusetzen. Die Lizenzbestimmungen und Spezifikationen der verwendeten Bibliotheken in Version 1.0 sind in *Tabelle 4* aufgeführt, die der automatisch generierten Dokumentation entnommen worden.

Unter spe-ed-docs.lehnurr.de sind immer die verwendeten Abhängigkeiten des neusten Entwicklungsstandes und deren Lizenzbestimmungen aufgeführt.

Externe Abhängigkeit	Lizenzbestimmung
JUnit4	Eclipse Public License 1.0
picocli	Apache 2.0
gson	Apache 2.0
Jetty Websocket	Apache 2.0, Eclipse Public License 1.0

Tabelle 4: Lizenzbestimmungen externer Abhängigkeiten

Benutzeranleitung

Wird Docker für das Ausführen der Software verwendet, sind keine weiteren Interaktionen möglich. Die Software verbindet sich automatisch mit dem konfigurierten spe_ed Webservice, spielt ein Spiel und beendet sich. Über die Ausgaben in der Konsole werden Informationen über den Verbindungsstatus der Anwendung mit dem spe_ed Webservice, über den Spielverlauf und die Entscheidungsfindung des Solvers bereitgestellt. Die Informationen bezüglich der Entscheidungsfindung sind auf den theoretischen Ansatz zurückzuführen, weswegen deren Bedeutung nicht in diesem Handbuch erläutert wird.

Wenn eine Java-Umgebung für das Ausführen verwendet wird, können beim Starten über die Kommandozeile Funktionen aktiviert oder deaktiviert werden. Hierfür gibt es zwei verschiedene Modi:

- LIVE: die Software verbindet sich mit dem spe_ed Webservice, spielt genau ein Spiel und beendet sich.
- SIMULATED: die Software simuliert den spe_ed Webservice, verwaltet hierfür die verschiedenen Spieler und beendet sich.

Funktionen im LIVE-Modus

Die Verwendung des Livemodus erfordert die folgende Eingabe über die Kommandozeile. Werte in eckigen Klammern sind optional und weißen einen Standardwert auf.

Das Kommandozeilenargument, um ein Spiel auf dem angegebenen Webservice mit einer Solver-Instanz zu spielen, ist folgendermaßen aufgebaut:

```
start live [-d] [-v] [-c=<level>] [-l=<logFilePath>]
           [-m=<maxThreadCount>] [-s=<solverType>]
```

In *Tabelle 5* ist für die verfügbaren Optionen eine Beschreibung aufgeführt. Bei Falscheingaben wird eine Fehlermeldung mit Korrekturhinweis ausgegeben.

Argument	Beschreibung
`-c` `--consoleLoggingLevel=<level>`	Ermöglicht es, die Ausgaben in der Konsole zu begrenzen. Der Standardwert ist 3 und zeigt alle Ausgaben an. Mögliche Werte: `ERROR = 0`, `WARNING = 1`, `GAME_INFO = 2`, `INFO = 3`
`-d` `--debug`	Erlaubt es, erweiterte Debug-Informationen auf der Konsole auszugeben.
`-l` `--logFileDirecotry=<logFilePath>`	Ändert das Verzeichnis, in welchem die LOG-Dateien abgelegt werden. Der Standardwert ist das Verzeichnis „log“ im Ausführungsverzeichnis. Der Wert “/” deaktiviert diese Funktion.
`-m` `--max-thread-count=<maxThreadCount>`	Legt die maximale Anzahl der zu verwendenden Threads fest. Der Standardwert ist 4. Der Maximalwert richtet sich nach den Hardware-Spezifikationen.
`-s` `--solver=<solverType>`	Ändert den Algorithmus, welcher zum Spielen des Spiels verwendet wird. Durch `-s=?` können möglichen Werte eingesehen werden.
`-v` `--viewer`	Visualisiert das Spiel in einem separaten Fenster, welches die verschiedenen Runden zeigt und den Lösungsweg darstellt. Diese Option verhindert das automatische Schließen der Anwendung.

Tabelle 5: Kommandozeilenargumente für den LIVE-Modus

Funktionen im SIMULATED-Modus

Das Kommandozeilenargument, um ein Spiel in einer lokalen Simulation mit einer Menge von Solvern zu spielen, ist folgendermaßen aufgebaut:

```
start simulated [-d] [-v] [-s=<solverTypes>] [-l=<logFilePath>]
                [-c=<level>] [-l=<logFilePath>] [-m=<maxThreadCount>]
                [-h=<boardHeight>] [-w=<boardWidth>]
                [--lower-deadline=<limit>] [--upper-deadline=<limit>]
```

In *Tabelle 6* ist für die verfügbaren Optionen eine Beschreibung aufgeführt. Bei Falscheingaben wird eine Fehlermeldung mit Korrekturhinweis ausgegeben.

Argument	Beschreibung
`-c` `--consoleLoggingLevel=<level>`	Ermöglicht es, die Ausgaben in der Konsole zu begrenzen. Der Standardwert ist 3 und zeigt alle Ausgaben an. Mögliche Werte: `ERROR = 0`, `WARNING = 1`, `GAME_INFO = 2`, `INFO = 3`
`-d` `--debug`	Erlaubt es, erweiterte Debug-Informationen auf der Konsole auszugeben.
`-h` `--height=<boardHeight>`	Legt die Höhe des simulierten Spielfeldes fest. Der Standardwert ist `10`.
`-l` `--logFileDirecotry=<logFilePath>`	Ändert das Verzeichnis, in welchem die LOG-Dateien abgelegt werden. Der Standardwert ist das Verzeichnis „log“ im Ausführungsverzeichnis. Der Wert “/” deaktiviert diese Funktion.
`--lower-deadline=<limit>`	Legt die untere Grenze für die zufällige Bestimmung von Deadlines in Sekunden fest. Der Standardwert ist 2.
`-m` `--max-thread-count=<maxThreadCount>`	Legt die maximale Anzahl der zu verwendenden Threads fest. Der Standardwert ist 4. Der Maximalwert richtet sich nach den Hardware-Spezifikationen.
`-s` `--solvers=<solverTypes>`	Legt fest, mit welchem Lösungsansatz die Spieler an dem simulierten Spiel teilnehmen sollen. Hierfür müssen die Spielertypen kommasepariert angegeben werden. Es müssen mindestens 2 und dürfen maximal 6 Spielertypen angegeben werden. Der Standardwert sind zwei Graph-basierte Spieler. Durch `-s=?` können möglichen Werte eingesehen werden.
`--upper-deadline=<limit>`	Legt die obere Grenze für die zufällige Bestimmung von Deadlines in Sekunden fest. Der Standardwert ist `15`.
`-v` `--viewer`	Visualisiert das Spiel in einem separaten Fenster, welches die verschiedenen Runden zeigt und den Lösungsweg darstellt. Diese Option verhindert das automatische Schließen der Anwendung.
`-w` `--width=<boardWidth>`	Legt die Breite des Simulierten Spielfeldes fest. Der Standardwert ist `10`.

Tabelle 6: Kommandozeilenargumente für den SIMULATED-Modus

Grafische Benutzeroberfläche

> ! Die Verwendung der grafischen Benutzeroberfläche verhindert das automatische Beenden der Anwendung, nachdem ein Spiel beendet wurde.

Die grafische Benutzeroberfläche steht ab der zweiten Runde zur Verfügung, wenn der Solver mit dem Kommandozeilenargument `-v` gestartet wird. Es wird für jeden kontrollierten Spieler ein separates Fenster angezeigt, sodass im LIVE-Modus immer genau ein Fenster zur Verfügung steht und im SIMULATED-Modus je Spieler ein Fenster verwendet wird. Sobald eines dieser Fenster oder das Konsolenfenster geschlossen wird, beendet sich die gesamte Anwendung. *Abbildung 12* zeigt das Fenster für einen Spieler in der ersten Runde.

Im unteren Bereich ist eine Bildlaufleiste verfügbar, mit welcher zwischen den bereits stattgefundenen Runden navigiert werden kann. Wenn sich der Regler an der rechten Kante befindet, wird automatisch die nächste beendete Runde angezeigt. Über der Bildlaufleiste auf der rechten Seite steht die Art des Lösungsalgorithmus, welcher die Entscheidungen trifft. Die Farbe des Textes entspricht der Farbe des eigenen Spielers.

In der rechten oberen Ecke werden Informationen über das Spiel bereitgestellt:

- Der Rundenzähler gibt die Anzahl der bereits verstrichenen Runden an.
- Die verfügbare Zeit ist die Anzahl an Sekunden, welche aus der Deadline des Servers und dem internen Deadline-Management errechnet wird.
- Die „performed action“ gibt an, welche Aktion für die dargestellte Situation gewählt wurde.
- Die benötigte Zeit ist die Anzahl an Sekunden, welche für die Bestimmung der gesendeten Aktion benötigt wurden.
- Über den Button „Save Slice“ können die aktuell dargestellten Bilder auf dem Computer gespeichert werden.

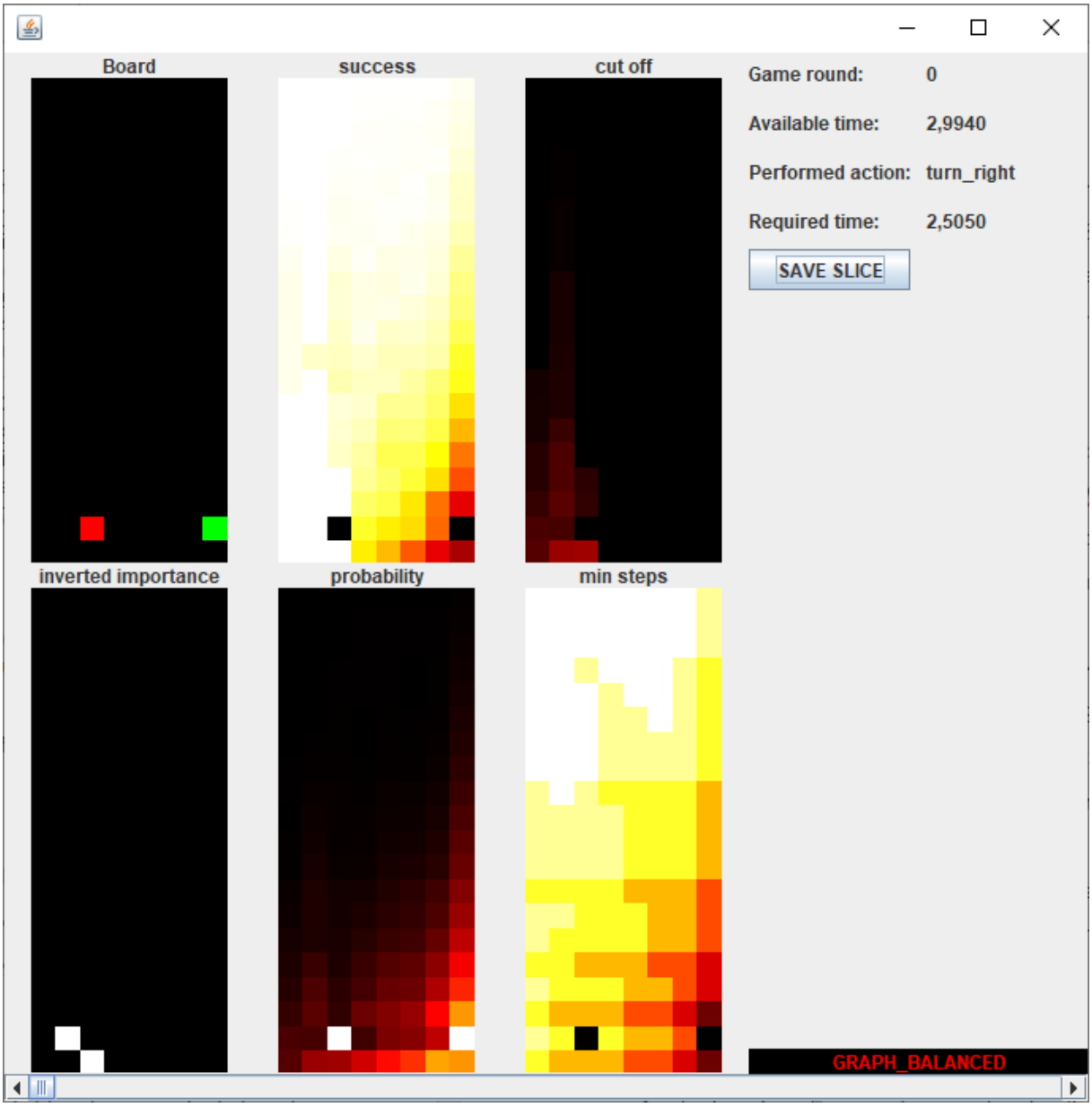

Abbildung 12: Grafische Benutzeroberfläche

Die dargestellten Bilder innerhalb der Benutzeroberfläche stellen die Nachverfolgbarkeit der Entscheidung sicher. Jedes Bild ist das Ergebnis einer Berechnung, welche zu der Entscheidungsfindung beiträgt. Helle, gelbliche Farben stehen hierbei jeweils für einen hohen Wert und rote, dunkle Farben für einen niedrigen Wert.

> ! Die dargestellten Bilder sind von dem verwendeten Algorithmus abhängig

- **board**: Zeigt das aktuelle Spielfeld mit den gezeichneten Pfaden der Spieler. Kollisionen werden weiß dargestellt.
- **success**: Stellt die Zellen dar, welche laut der Berechnung durch die gewählte Aktion erreichbar sind. Ein niedriger Wert bedeutet, dass die Zelle durch die Aktion eines Gegners möglicherweise schwerer zu erreichen ist.
- **cut off**: Wird für Zellen angezeigt, welche mit Sicherheit vor einem Gegner erreicht werden können. Hierauf basierend wird der Weg des Gegners abgeschnitten, um dessen Aktionen einzuschränken.
- **inverted importance**: Zeigt für die direkt erreichbaren Zellen, was deren invertierte Wichtigkeit (=Irrelevanz) ist. Zellen, welche weniger wichtig sind, weißen hohe Werte auf. Wichtige Zellen werden mit einem geringen Wert versehen. Die wichtigsten Zellen werden mit 0 bewertet und somit schwarz dargestellt.
- **probability**: Visualisiert die Wahrscheinlichkeit des gegnerischen Aufenthalts in den nächsten Runden an jeder dargestellten Stelle.
- **min steps**: Visualisiert in wie vielen Runden ein Gegner welche Zellen erreichen kann. Der Wert der Zelle entspricht hierbei der Runde, in welcher der Gegner diese Zelle erreicht.

Errorhandling

Bei der Ausführung der Anwendung kann es zu Fehlern kommen. Im Folgenden wird ihnen der Umgang mit diesen Fehlern erläutert. Wenn Sie bei der Fehlerbehandlung Unterstützung brauchen, können Sie über eine E-Mail an team@lehnurr.de mit einem Entwickler in Kontakt treten. Halten Sie, falls möglich, hierfür bitte die erstellte LOG-Datei bereit.

Beim Starten der Anwendung werden die möglichen Parameter angezeigt

Diese Meldung erscheint, wenn Sie fehlerhafte Startparameter angegeben haben. Beachten Sie den Hinweis in der ersten Zeile. Hier wird ihr Fehler beschrieben und mögliche Lösungen aufgezeigt. Korrigieren Sie Ihre verwendeten Parameter und starten Sie die Software neu.

Die Verbindung wurde vom Webservice getrennt

Es besteht die Möglichkeit, dass der spe_ed Webservice aufgrund eines Fehlers die Verbindung zu Ihnen trennt. Auf dieses Verhalten haben Sie keinen Einfluss. Sie können ein neues Spiel starten.

Ein Spieler ist ohne ersichtlichen Grund gestorben

Im Live-Modus kann es vorkommen, dass ein Spieler seine Aktion zu spät oder gar nicht sendet. In diesem Fall stirbt der Spieler ohne eine sichtbare Kollision. Wenn es sich um Ihren eigenen Spieler handelt, können Sie die Antwortzeit des Servers am Anfang der Konsolenausgabe oder der Log-Datei prüfen. Liegt diese Zeit über einer Sekunde, ist ihre Internetverbindung zu dem spe_ed Webservice nicht schnell genug.

Die LOG-Datei konnte nicht erstellt werden

Wenn keine Log-Datei erstellt wurde liegt das daran, dass als Verzeichnis für das Logging „/“ angegeben wurde oder die Software nicht die Berechtigungen hat die Datei zu schreiben. Starten Sie die Anwendung mit Administrationsrechten und ändern Sie gegebenenfalls das Verzeichnis für die Log-Dateien.

Es wird die Meldung OutOfMemory angezeigt

Wenn auf einem System mit wenig Arbeitsspeicher mit sehr vielen Threads gearbeitet wird, kann es zu einem Speichermangel kommen. Tritt dieser Fehler bei Ihnen auf, verwenden Sie bitte die Standardeinstellung für die maximale Anzahl an Threads oder setzen Sie diese zusätzlich mit `-m=1` herab.

11 Literaturverzeichnis

[1] G. Coleman und R. Verbuggen, „Rapid Application Development (RAD),“ *Software Quality Journal,* Nr. 7, pp. 108-110, 1998.

[2] informatiCup, „spe_ed: informatiCup 2021 - Aufgabe,“ 26 November 2020. [Online]. Available: https://github.com/informatiCup/InformatiCup2021/blob/master/spe_ed.pdf. [Zugriff am 10 Januar 2021].

[3] W. Leininger und E. Amann, Einführung in die Spieltheorie, Dortmund: Universität Dortmund, 2008.

[4] W. Kantschik, Genetische Programmierung und Schach, Dortmund, 2006.

[5] M. Holler, G. Illing und S. Napel, Einführung in die Spieltheorie, Deutschland: Springer, 2019.

[6] M. Soll, „InformatiCup2021: Specs der Referenzsysteme,“ 04 Oktober 2020. [Online]. [Zugriff am 13 Januar 2021].

[7] P. Klaus und R. Chris, Basiswissen Requirements Engineering 4. Auflage, Heidelberg: dpunk.verlag, 2015.

[8] python.org, „global interpreter lock,“ 22 Dezember 2020. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock. [Zugriff am 14 Januar 2021].

[9] Oracle America, Inc, The Java Language Specificaion - Java SE 15 Edition, 2020.

[10] Google LLC, „Google Java Style Guide,“ [Online]. Available: https://google.github.io/styleguide/javaguide.html. [Zugriff am 14 Januar 2021].

[11] M. Fowler, „Continuous Integration," 01 Mai 2006. [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html. [Zugriff am 13 Januar 2021].

[12] M. Soll, „InformatiCup2021: Gleichzeitiges ausscheiden der letzten 2 Spieler," 26 Oktober 2020. [Online]. Available: https://github.com/informatiCup/InformatiCup2021/issues/18. [Zugriff am 12 Januar 2021].

12 Anhangsverzeichnis

Anhang 1: Auswertung Spielfeldgröße

Die Excel Tabelle, die verwendet wurde, um die Spielfeldgröße auf der Online API zu untersuchen ist unter folgendem Link zu finden:

https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/board_size.xlsx

Anhang 2: Auswertung verfügbarer Zeit

Die Excel Tabelle, die verwendet wurde, um die verfügbare Zeit auf der Online API zu untersuchen ist unter folgendem Link zu finden:

https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/available_time.xlsx

Anhang 3: Aufzeichnungen durchgeführter Spiele

Folgende Videos wurden als Beispiele für das entwickelte Verhalten aufgezeichnet:

- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_0.mp4
  Der Spieler (Rot) erkennt, dass er durch einen Gegner (Blau) bedroht wird und weicht er Gefahr aus. In Runde 432 geht er bewusst das Risiko einer Kollision ein, um sich den größeren zusammenhängenden Bereich zu sichern. Im Anschluss schneidet er dem Gegner den Weg ab, wodurch dieser einige Runden später ausscheidet.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_1.mp4
  Der Spieler (Rot) verteidigt sich mehrfach gegen einen Gegner (Grün) und greift diesen mehrfach an. Gegen Ende des Kampfes sichert sich der Spieler den größeren sicheren Bereich, was zum Sieg führt.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_2.mp4
  Der Spieler (Grün) füllt seinen Bereich sinnvoll und langsam aus, um möglichst viele Runden zu überleben.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_3.mp4
  Der Spieler (Rot) füllt seinen Bereich sinnvoll und langsam aus, um möglichst viele Runden zu überleben.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_4.mp4
  Der Spieler (Grün) schneidet dem Gegner (Gelb) einen möglichen Fluchtweg ab.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_5.mp4
  Der Spieler (Rot) schneidet einem Gegner (Hellblau) den einzig möglichen Fluchtweg ab, wodurch dieser ausscheidet.

- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_6.mp4
  Der Spieler (Grün) erkennt eine mögliche Gefahr, wodurch er der Risikozone entkommt. Im Anschluss nutzt er das Fehlerverhalten des Gegners (Rot), um ihm den Weg abzuschneiden.
- https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/videos/video_7.mp4
  Der Spieler (Blau) schneidet einem Gegner (Gelb) einen Fluchtweg ab.

Anhang 4: Vergleich zu anderen Teams

Die vollständigen Daten zum Vergleich gegen andere Teams sowie die verwendeten Rechenvorschriften können hier gefunden werden:

https://github.com/Lehnurr/spe-ed-solver/blob/main/elaboration/appendices/comparison_online_api.xlsx