Gruppe 3

Heinz Stadler, Melinda Betz, Phil Heger und Björn Wladasch

FernUniversität in Hagen, Universitätsstraße 47, 58097 Hagen, Deutschland
{vorname.nachname}@studium.fernuni-hagen.de
https://www.fernuni-hagen.de

1 Einleitung

Die vorliegende Aufgabe wurde in einer Gruppe von vier Informatik-Studenten bearbeitet, wovon zwei im Bachelorstudiengang Informatik und zwei im Masterstudiengang Praktische Informatik immatrikuliert waren.

2 Kommunikation und Kommunikationsmittel

Die gruppeninterne Kommunikation erfolgte über den für das Fachpraktikum eingerichteten Discord-Server. Jeden Mittwoch wurden im Sprachchat der Gruppe aktuelle Aufgaben und Probleme besprochen. Anfangs wurde außerdem gemeinsam die grundsätzliche Softwarearchitektur mittels UML-Diagrammen entwickelt. Der Gruppen-Chat wurde darüber hinaus für kurzfristige Absprachen genutzt. Die Quelltextverwaltung erfolgte über das Versionsverwaltungssystem *Github*, dessen Issue-Management ebenfalls verwendet wurde. Hier wurden alle Aufgaben zusammengetragen und jeweils ein Bearbeiter zugeordnet. Aus dem jeweiligen Issue wurde jeweils ein *Feature Branch* erstellt, in dem die Bearbeitung der Aufgabe erfolgte. Abschließend wurden die Änderungen des *Feature Branch* mittels eines *Pull Request* auf den *Master Branch* angewendet. Das Repositorium wurde so konfiguriert, dass ein *Pull Request* erst freigegeben wird, wenn er durch mindestens ein anderes Gruppenmitglied überprüft wurde. Dadurch sollte die Qualität des Quelltextes erhöht und das gemeinsame Verständnis des Quelltextes gefördert werden.

3 Technische Rahmenbedingungen und Softwarebasisarchitektur

Zu Beginn des Praktikums wurden die Vor- und Nachteile der Verwendung einer auf die Aufgabenstellung optimierte Programmiersprachen diskutiert. Ein intuitives Programmiergerüst basierend auf allgemein anerkannten Notationen zur Implementierung logischen Denkens (vergl. [11]) schien reizvoll. Die Aussage von T. Albrecht, dass für viele Teilnehmer am MAPC *Multi-Agent Programming Contest* die Fehlerfindung eine schwierige und zeitintensive Aufgabe darstellt [1, S. 17], bewegte die Gruppe dazu sich für eine Programmiersprache mit umfangreichen Test-Bibliotheken zu entscheiden. Da von der MASSim (*Multi-Agent*

Systems Simulation Platform) ein Basisgerüst für die Teilnahme am MAPC in der Sprache Java bereitgestellt wird und mit JUnit [7] ein sehr umfangreiches Test-Rahmenwerk für die Sprache verfügbar ist, fiel die Entscheidung auf Java.

Das in dieser Arbeit entwickelte Agentensystem basiert auf der BDI-Architektur [5], die das menschliche Denken und Schlussfolgerns abstrahiert bzw. nachbildet. Diese Architektur schien ein für die Gruppe zugänglicher Einstieg in das Thema der Multi-Agentensysteme. Zusätzlich verwendeten alle Teilnehmer am MAPC 2021 entweder eine BDI-Platform oder zumindest einen an die BDI-Architektur angelehnten Programmierstiel [1, S. 10].

Auf Grundlage der gewählten Basisarchitektur wurde anhand von Klassen-UML-Diagrammen konkrete Systemarchitekturen erarbeitet. Die UML-Diagramme stellen aufgrund der übersichtlichen, visuellen Darstellung des komplexen Systems eine gute Diskussionsgrundlage dar.

Aus der Konzeptionsphase resultierten zwei unterschiedliche Ansätze, die in der Folge auch weitestgehend unabhängig umgesetzt und getestet wurden. Die Gruppe erhoffte sich durch das Verfolgen verschiedener Varianten differenzierte Lösungen von Teilproblem, deren Vor- und Nachteile gegenseitig abgewogen werden sollten. Beide Ansätze basieren auf einer geteilten Basisarchitektur. Dies ermöglicht die gemeinsame Nutzung einzelner Architekturbausteine sowie den Austausch von Modulen. Zusätzlich standen durch die zwei Varianten gegenseitige Sparringspartner zur Verfügung, die in diversen Testmatches gegeneinander antraten.

4 Agentensystem V1 - Gruppenbeitrag Heinz Stadler

Nach einer ausführlichen Einarbeitung in die Thematik der Multiagentensysteme erfolgte die Analyse der Ergebnisse des *15. Multi-Agent Programming Contest* [1]. Daraus ging hervor, dass nicht nur die Entscheidungsfindung der Agenten eine Herausforderung darstellt, sondern ebenso der Aufbau einer konsistenten und umfangreichen Wissensbasis (vgl. [1, S. 29]) sowie die effiziente Problemfindung (vgl. [1, S. 17]).
Als Folge dessen entschied sich der Autor mit dem Aufbau der Wissensverwaltung (siehe 4.2) zu beginnen. Diese umfasst eine Datenstruktur zur Speicherung der Simulationsinformationen, sowie eine Lösung zum Aufbau einer globalen Karte des Simulationsgebiets. Nach Fertigstellung der Karte und deren funktionalen Verifikation, wurde parallel an der Konzeptionierung des Agentensystem V1 (siehe 4.1) und dessen Ziel- und Absichtsfindung, sowie der Implementierung einer intelligenten Wegfindung (siehe 4.3) gearbeitet. Um die Ergebnisse effektiv verifizieren zu können, folgte die Erstellung eines grafischen Analysewerkzeugs (siehe 4.5).

4.1 Architektur

Das Agentensystem V1 wurde vom Autor konzipiert, in der Gruppe abgestimmt und implementiert. Es erweitert das BDI-Konzept [5] um zusätzliche Daten-,

Berechnungs- und Entscheidungsebenen, die in Abbildung 1 illustriert sind.
Das System kombiniert den Aufbau eines BDI-Agenten [3, S. 58] mit einer bidirektionalen vertikalen Schichtarchitektur [3, S. 61-62], auf die in Abschnitt 4.4 näher eingegangen wird.

Abb. 1. Architektur Agent V1

Jeder Agent wurde mit einer Vorgesetzteninstanz, die eine zusätzliche Entscheidungsebene bildet, kombiniert und in einem Thread parallelisiert. Werden Agenten zu einer Gruppe zusammengeführt, bleibt eine Vorgesetzteninstanz aktiv. Die sonstigen Instanzen der Gruppe werden passiv und übernehmen im Weiteren nur noch die Weiterleitung von Nachrichten an die aktive Entität.

Die Kommunikation zwischen Instanzen in verschiedenen Threads erfolgt über Nachrichten, die in einer threadsicheren Warteschlange zwischengespeichert werden. Die Verständigung des Agenten mit seinem direkten Vorgesetzten wird mittels Methodenaufrufen realisiert.

Agentengruppen aktualisieren eine gemeinsame Karte mit im Simulationsverlauf erhaltenen Umgebungsinformationen. Die Karten werden zusammen mit dem Modul zur Wegfindung von einem zentralem, threadsicheren Navigationsmodul, das als Einzelstück ausgeführt wurde, verwaltet.

4.2 Wissensverwaltung

Jeder Agent hat Zugriff auf eine individuelle Wissensbasis (*Beliefs*), die von der Simulation bereitgestellte Informationen auswertet und speichert.
Die enthaltenen Umgebungsdaten beschränken sich auf das aktuelle Sichtfeld des Agenten. Um aus den partiellen Umgebungsinformationen eine globale Sicht des Simulationsgebiets zu erhalten, werden die lokalen Daten an das Navigationsmodul weitergeleitet und in einer chronologisch fortgeschriebenen Karte zusammengeführt.

Beim Simulationsstart erhält jeder Agent eine Karte mit festgelegter Initialgröße (Abb. 2 Abschn. 1), die beim Erkunden der Umgebung erweitert wird (Abb. 2 Abschn. 2). Treffen sich zwei Agenten aus unterschiedlichen Gruppen, wird dies vom Navigationsmodul erkannt und an die aktiven Vorgesetzten beiden Gruppen gemeldet. Stimmen beide Vorgesetzte einer Vereinigung zu, werden deren Karten überlagert (Abb. 2 Abschn. 3) und schließlich zusammengeführt

(Abb. 2 Abschn. 4). Die entstandene Karte ermöglicht nun im weiteren Simulationsverlauf die aktuelle Position aller Agenten einer Gruppe untereinander zu bestimmen. Aus dieser Information kann durch jeweils zwei Agenten, die sich in entgegengesetzte Richtungen bewegen und sich durch die periodische Randbedingung [13] des Simulationsgebiets zwangsläufig wieder treffen, die Kartengröße ermittelt werden. Nach erfolgreicher Ermittlung wird die Karte beschnitten, wobei die Informationen abgeschnittener Bereiche auf der gegenüberliegenden Seite eingefügt werden und somit nicht verloren gehen (Abb. 2 Abschn. 5).

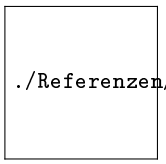

Abb. 2. Die Karte im Simulationsverlauf

4.3 Wegfindung

Unter dem Begriff Wegfindung verstehen wir das Lösen folgender zwei Teilprobleme:

1. Finde den Abstand zweier Punkte im Simulationsgebiet unter Berücksichtigung von Hindernissen.
2. Finde einen idealen, kürzesten Weg zwischen zwei Punkten im Simulationsgebiet unter Berücksichtigung von Hindernissen.

Das Lösen des ersten Teilproblems befähigt den Agenten seine Ziel- und Absichtsfindung auf Grundlage von korrekten Entfernungsdaten, anstelle von approximativen Annahmen durchzuführen. Wurde ein Ziel priorisiert, wird ein ideal kurzer Weg von der aktuellen Position zum Zielpunkt gesucht. Dies entspricht der Bearbeitung des zweiten Teilproblems.

Der Autor hat bereits zum Projektbeginn die Absicht gefasst beide Probleme, bezogen auf die Anforderungen der Simulation, zu lösen. Dabei wurden klassische Suchalgorithmen wie z.B. A* [2] als auch dynamische Echtzeitalgorithmen [3, 182-191] untersucht. Da dynamische Algorithmen lediglich zum Lösen von Teilproblem 2 einsetzbar sind, wurde sich für den A* Algorithmus mit Heuristik Manhattan-Distanz $d(A, B) = |A_x - B_x| + |A_y - B_y|$ entschieden.

Mit der Zellmenge V besitzt der Algorithmus in seiner Grundform eine Laufzeitkomplexität von $O(|V|^2)$. Kombiniert mit geschätzten 50-100 Berechnungen pro Agent und Simulationsschritt schied eine CPU-basierte Lösung aufgrund Zeitbeschränkungen in der Absichtsfindung aus.

Es erfolgte die Implementierung des Suchalgorithmus in der Form eines *Computeshaders* in der GLSL (*OpenGL Shading Language*) [6]. Mit dieser Technolo-

gie konnte die Wegfindung auf über 1000 Berechnungen pro Simulationsschritt parallelisiert werden.

Aus der hohen Parallelisierung und der Implementierung in GLSL ergaben sich zwei Herausforderungen:

1. Integration intelligenter platzsparender Datenstrukturen um die Speicherplatzkomplexität zu bewältigen
2. Implementierung des auf Objektreferenzen basierenden Algorithmus in einer Programmiersprache ohne Objektreferenzen

Objektreferenzen konnten durch die Implementierung einer Vorrangwarteschlange in GLSL mit direktem Speicherzugriff über Arrays vermieden werden. Die Beschränkung der maximalen Warteschlangenlänge und die Speicherung von binären Werten auf Bitebene bewältigte die Speicherplatzkomplexität und führte zu einer praktikablen Lösung, die auf einer integrierten Intel UHD Graphics 620 Grafikkarte verlässlich, innerhalb von ca. zwei Zehntelsekunden ≈ 1000 Wegfindungsergebnisse liefern konnte.

Als Ein- und Ausgabedatenstruktur wurde eine 3D-Textur mit zwei Farbkanälen (siehe Abb. 3) gewählt, die zusätzlich als visuelle Hilfestellung bei der Problemfindung diente. Die erste Ebene der Textur codiert die Eingabekarte der Agentengruppe binär nach Zelltyp „Hindernis“. Jede weitere Ebene dient als gemeinsam genutzte Datenstruktur für alle Berechnungen eines Agenten. Entfernungs- und Richtungsinformationen werden dabei getrennt in den Farbkanälen gespeichert.

./Referenzen/Pathfinding.pdf

Abb. 3. Datenstruktur Wegfindung (aus realer Simulation)

Bei der Wegberechnung wird das Überqueren von mit Hindernissen belegten Zellen mit zusätzlichen Kosten bewertet. Die Agenten erhalten somit einen kürzesten Weg, der sowohl Wege durch Hindernisse, als auch Wege um diese herum enthalten kann.

Die Ermittlung des kürzesten Wegs wird ohne an den Agenten angehängte Blöcke durchgeführt. Die Agenten besitzen zusätzliche Logik zur Rotation eines angehängten Blocks sowie zum Entfernen von Hindernissen, um die ermittelten Bewegungen der Wegfindung kollisionsfrei durchführen zu können.

4.4 Ziel- und Absichtsfindung

Wie bereits in Abschnitt 4.1 beschrieben, erfolgt die Ziel- und Absichtsfindung über zwei Ebenen in einer vertikalen bidirektionalen Schichtarchitektur aus einem BDI-Agenten und dessen Vorgesetzteninstanz. Der Agent ist in der Lage für sich als Individuum sinnvolle Ziele zu entwickeln. Die Vorgesetzteninstanz kümmert sich um die Koordinierung von Agenten zur Bewältigung von Gruppenzielen. Die Kommunikation erfolgt sowohl vom Agenten zur Vorgesetzeninstanz als auch in entgegengesetzter Richtung. Nachfolgend erfolgt die Beschreibung der Ziel- und Absichtsfindung, die zusätzlich in Abbildung 4.4 dargestellt ist.

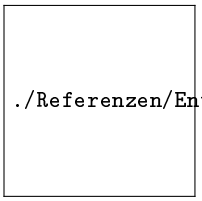

Abb. 4. Diagramm Ziel- und Absichtsfindung Agent V1

Zu Beginn jedes Simulationsschritts werden die aktuellen Simulationsdaten ausgewertet und die Wissensbasis der Agenten aktualisiert. Umgebungsinformationen werden an das Navigationsmodul weitergeleitet, das diese speichert und Zielpunkte für die Wegführung generiert. Zeitgleich sendet der Agent einen Ausschnitt seines aktuellen Zustands an seine Vorgesetzteninstanz. Diese versucht aus allen Agenten in der Gruppe effektive Kombinationen zu bilden, um Mehrblockaufgaben zu konstruieren, die Kartengröße zu erkunden oder die Bewachung einer Zielzone zu koordinieren.

Nachdem die Wegfindung abgeschlossen wurde, wird die Wissensbasis der Agenten aktualisiert. Im Folgenden ergänzt der Agent seine Ziele durch Optionen, die durch neue Einblock-Aufgaben entstanden sind, sowie durch Optionen die von seiner aktiven Vorgesetzteninstanz übermittelt wurden. Nicht mehr erfüllbare Optionen oder bereits erfüllte Gruppenoptionen werden anschließend aus den Zielen gelöscht.

Jede Option besitzt eine Priorität, die im Fall von Einblock-Aufgaben auch dynamisch sein kann. Der Aufbau ist modular, wodurch Optionen auch Unteroptionen enthalten können. Dies ermöglicht Teilfunktionalitäten wiederzuverwenden und die Absichtsfindung zu vereinfachen. Die Auflistung aller implementierten Optionen und Unteroptionen (über 30 Stück) würde den Rahmen dieser Arbeit überschreiten. Im Nachfolgenden werden die Hauptoptionen, sortiert von unwichtig zu wichtig, dargestellt:

- erforsche Karte
- entferne verbundene, aber nicht verwendbare Blöcke
- bearbeite Einblock-Aufgabe
- bearbeite Gruppenaufgaben
- befreie aus festgefahrener Situation
- weiche Meteoriten aus

Die Ziele des Agenten werden nun nach Priorität sortiert und die Absicht mit Algorithmus 1 bestimmt. Aus der gefundenen Absicht wird die nächste Aktion extrahiert und an den Simulationsserver gesendet.

4.5 Verifikation und Problemfindung

Die Validierung verschiedener Strategien erfolgte über die erreichte Punktzahl in Testspielen und lieferte zufriedenstellende Ergebnisse. Die Methoden zur Verwaltung der Karte wurden mittels JUnit Tests [7] verifiziert. Im Gegensatz dazu

Algorithm 1: Absichtsfindung

```
Data: l nach Priorität absteigend sortierte Liste mit Zielen die Unterziele
      enthalten können
Result: Absicht
forall ziel in l do
    forall unterziel in ziel do
        if unterziel nicht erfüllt and unterziel ausführbar then
            return unterziel
        end
    end
    if ziel nicht erfüllt and ziel ausführbar then
        return ziel
    end
end
```

konnte das Verhalten der Agenten nicht effizient durch Einzeltests verifiziert werden, da der Entscheidungsprozess in großer Abhängigkeit mit der dynamischen Wissensbasis der Agenten steht. Als Teststrategie wurde stattdessen das genaue Beobachten der Agenten, analog eines Trainers einer Sportmannschaft, gewählt. Wurden Probleme mitverfolgt, erfolgten gezielte Einzeltests um diese zu beheben.

Dieser Ansatz erforderte zusätzliche visuelle Information, die der Monitor, über den die Simulation beobachtet werden kann, nicht liefert. Aus diesem Grund wurde ein grafisches Analysewerkzeug (siehe Abbildung 5), das einen detaillierten Einblick über den aktuellen Entscheidungsprozess und die Wissensbasis der Agenten liefert, implementiert. Neben allgemeiner Informationen über den selektierten Agenten werden Information über Gruppen- und Individualziele angezeigt. Zusätzlich erfolgt die farbliche Markierung der Agenten anhand ihrer aktuellen Gruppenziele. Vervollständigt wird die Anzeige u.a. mit Simulationsinformationen wie aktuellem Punktestand oder einer grafischen Darstellung der aktuell möglichen Aufgaben.

Abb. 5. Graphisches Analysewerkzeug

Das Analysewerkzeug hat sich als sehr wertvoll erwiesen und ermöglichte, durch die dynamischen Echtzeitinformationen, die effiziente Weiterentwicklung der Agenten.

4.6 Weitere Beiträge am Projekt

Zusätzlich zu den zuvor beschriebenen Beiträgen, hat der Autor die Quellcodeverwaltung [8] administriert, die Projektseite [9] konfiguriert und erstellt, ein

Turnier in Vertretung geleitet und einen Fehler im Simulationsserver, gemeldet [10] und behoben.

5 Gruppenbeitrag Melinda Betz - Agent V2

Neben meiner Tätigkeit als Gruppenlead habe ich einen alternativen Agenten V2 entwickelt.

5.1 Architektur

Der Agent V2 arbeitet mit der Step-Methode, analog dem BasicAgent aus Massim. Der Agent verwendet nicht das komplette Pathfinding des V1, da OpenGL auf meinen AMD Rechnern nicht, zumindest nicht performant, funktionierte. Der Agent stellt also seine eigenen Berechnungen an, um zu einem Ziel zu kommen. Außerdem nutzt er eigene Desires, nicht die des Agenten V1. Die Desires sind dabei aufteilbar in solche, die keine Task benötigen und Desires zur Bearbeitung einer Task. Bei Letzteren ist ein Teil nur für Mehr-Block-Tasks zuständig.

ohne Task
LocalExploreDesire, GoAdoptRoleDesire, ExploreMapSizeDesire

mit Task
GoAbandonedBlockDesire, GoDispenserDesire, GoGoalZoneDesire, SubmitDesire,

MehrBlockTask
MasterMultiBlocksDesire, HelperMultiBlocksDesire, Helper2MultiBlocksDesire, ConnectMultiBlocksDesire, DisconnectMultiBlocksDesire

5.2 Entscheidungsfindung

Ein Agent durchläuft in jedem Step alle Desires und prüft ob sie in seinem momentanen Zustand (Belief) möglich bzw. ausführbar sind. Für alle ausführbaren Desires wird dynamisch, abhängig auch vom Arbeitsstand des Desires, eine Priorität vergeben. Das Desire mit der höchsten Priorität wird dann zur Intention welche in diesem Step vom Agenten ausgeführt wird.

5.3 Strategie und Task Bearbeitung

Wenn zwei Agenten sich treffen, schließen sie sich zu Supervisor-Gruppen zusammen, um ihr Sichtfeld zu vergrößern. Die kleinere Gruppe wird dabei in die größere integriert. Jeder Agent holt sich zuerst die Rolle Worker. Diese Rolle kann alles, manches zwar nicht ganz optimal, aber gut genug.

Alle Worker ohne Block besorgen sich selbständig einen Block. Um eine möglichst breite Auswahl an Blöcken zu erreichen, darf die maximale Anzahl eines

Blocktyps (Setup Variable) nicht überschritten werden. Mit diesem Block bewegen sie sich in Richtung Goal Zone. Jeder Agent in einer Goal Zone, der gerade nicht an einer Mehr-Block-Task arbeitet (AgentCooperations), prüft in jedem Step für alle aktiven Tasks, ob er den innersten Block dieser Task besitzt. Eine Ein-Block-Task kann so direkt bearbeitet werden (Block in Position bringen und submitten). Für eine Mehr-Block-Task benötigt der Agent noch mindestens einen Helper, der einen weiteren passenden Block beisteuern kann, um eine neue Adhoc-Gruppe (AgentCooperation) mit sich als Master bilden zu können. In der Klasse AgentsCooperations wird festgehalten wer gerade in welcher Rolle (Master, Helper) mit welcher Task beschäftigt ist. Für alle Agenten ist dort auch der Stand der Taskabarbeitung aller anderen Beteiligten ersichtlich. Es existiert im Grunde keine zentrale Stelle zur Koordination der Agenten, jedoch gibt es einstellbare Regeln (Setup Variablen) wie viele Mehr-Block-Tasks gleichzeitig möglich sein sollen etc.. Das soll vor Klumpenbildung in der Goal Zone schützen und parallel weiterhin Ein-Block-Tasks ermöglichen.

5.4 Umgebungsfindung und Synchronisation

Ein Agent V2 kennt alles (vor allem Dispenser und GoalZones), was schon einmal in seinem Sichtfeld oder dem eines Agenten seiner Supervisor-Gruppe war. Entfernung und Richtung zu diesen Punkten kann er selbst ermitteln.

Die Synchronisation von Agenten ist, neben der beschriebenen Bearbeitung von Mehr-Block-Tasks, wichtig für die Bestimmung der Mapgröße. Dabei wird für die Ermittlung von Höhe und Breite der Map, jeweils aus den ersten beiden Treffen zweier Agenten, zentral abgewickelt über die Klasse AgentMeetings, eine AgentCooperation gebildet. Der Master läuft einmal um die Map herum. Der Helper wartet bis dieser wieder zurück ist. Aus der Position des Masters bei dem erneuten Treffen kann nun die genaue Größe (Höhe oder Breite) der Map berechnet werden.

5.5 Schwierigkeiten und Lösungsstrategien

Eine Hauptschwierigkeit war, nicht genau zu wissen, was ein Agent gerade attached hat, da diese Information im Percept des Servers nicht detailliert enthalten ist. Die Lösung, sich diese Information in Variablen zu merken, funktionierte zwar prinzipiell, stieß aber spätestens bei Einführung der Clear-Events an Grenzen. Außerdem hat man Schwierigkeiten, wenn ein Agent mit einer Geschwindigkeit größer zwei läuft und in einem Schritt abbricht (Returncode partial_fail). Es ist dann nicht möglich herauszufinden, wie weit der Agent vor dem fail gekommen ist, wo er sich also gerade auf dem Spielfeld befindet. Hier ist die etwas unbefriedigende, aber stets erfolgreiche Lösung die: kein Agent läuft schneller als zwei (was durch die Rolle Worker automatisch gegeben ist).

5.6 Verbesserungspotential

Was ich noch deutlich verbessern müsste ist, wie oben schon erwähnt, zum einen die dynamischere Erkennung der Umgebung (Goal Zones, Dispenser etc.) sowie das Nachführen der Informationen über die aktuellen Attachements der Agenten. Beides ist durch meine Versuche, diese Neuerungen für Turnier 6 einzubauen, eigentlich nur schlimmer geworden und war in „älteren Agenten“ schon mal wesentlich besser.

Potential würde ich auch noch bei meinen oben beschriebenen AgentCooperations sehen. Diese sind derzeit, einmal angelegt, zu starr. Der Master wartet auf seine Helper geduldig bis zum Ende aller Tage, genau genommen bis zur Deadline der betroffenen Task. Eine dynamischere Variante, bei der der Master unterdessen neue Gelegenheiten mit anderen Agenten wahrnehmen kann, wäre wohl zu bevorzugen.

6 Gruppenbeitrag Phil Heger

6.1 Logging

Anfangs wurde ein Logging-Modul (AgentLogger.java) implementiert, in dem der Logging-Output konfiguriert wird. Dieses Modul ermöglicht es, den Logging-Output, der von einem Agenten in einem bestimmten Desire oder Modul ausgegeben wird, in eine eigene Datei zu schreiben. Dadurch sind die Entscheidungsfindung dieses Agenten und seine Zustandsänderungen besser nachvollziehbar, als wenn der gesamte Logging-Output aller Agenten in der gleichen Konsole bzw. Datei ausgegeben wird und nachträglich gefiltert werden muss.

6.2 Strategien zum Stören gegnerischer Agenten

Durch eine gezielte clear-Aktion auf die Position des Gegners wird dessen Energielevel um einen definierten Betrag verringert. Beträgt sein Energielevel 0, dann wird der Agent für einige Runden deaktiviert und verliert die Verbindung zu allen mit ihm verbunden Blöcken. Dabei gibt es zwei Probleme, die die Wirksamkeit einer einfachen Strategie (z. B. Angreifen beliebiger Agenten an beliebigen Stellen der Karte) stark verringern:

- Bei der clear-Aktion muss das Feld angegeben werden, auf dem sich der gegnerische Agent am Ende der Runde befinden wird. Diese Position ist bei gegnerischen Agenten jedoch unbekannt, da er sich in alle Richtungen bewegen oder auch stehenbleiben kann. Die Wahrscheinlichkeit, den Gegner zu treffen, ist dadurch sehr gering.
- Der Schaden einer erfolgreichen clear-Aktion ist nicht sehr groß. So beträgt der max. mögliche Schaden, wenn sich der angegriffene Agent in einem angrenzenden Feld befindet, nur 16 Punkte bei einer Gesamtenergie von 100 Punkten. Der Schaden halbiert sich mit der Distanz zum gegnerischen Agenten. Hinzu kommt, dass die Erfolgswahrscheinlichkeit der clear-Aktion bei allen Rollen (außer der Rolle *digger*) nur 30 % beträgt.

Für ein wirksames Stören der Gegner ist daher eine komplexere Strategie und das Annehmen der Rolle *digger* erforderlich.

Dispenser blockieren Die erste Idee besteht darin, das Sammeln von Blöcken für die gegnerischen Agenten zu erschweren, indem ein eigener Agent auf das gleiche Feld wie ein Dispenser geht und dort verbleibt. Dadurch ist es nicht mehr möglich an diesem Dispenser neue Blöcke zu erzeugen. Dabei werden Dispenser ausgewählt, die für die Bearbeitung der aktuellen Aufgaben am wichtigsten sind. Aufgrund der hohen Anzahl an Dispensern in den Tunier-Konfigurationen zeigte sich jedoch, dass dieser Ansatz nicht praktikabel ist, da eine große Anzahl an eigenen Agenten für das Blockieren der vielen Dispenser notwendig wäre. Um die eigenen Agenten nicht ebenfalls zu behindern, müssen diese über blockierte Dispenser informiert werden.

Goal Zone verteidigen Bei diesem Ansatz wurde der in der Einleitung dieses Unterkapitels erwähnte grundlegende Mechanismus (clear-Aktion auf Position des gegnerischen Agenten) umgesetzt, wobei der eigene Agent gegnerische Agenten nur in der Zielzone angreift, da die gegnerischen Agenten hier oft auf andere Agenten warten und sich in dieser Zeit nicht bewegen. Der eigene Agent kann sich dadurch nähern und mit dem größtmöglichen Schaden angreifen. Ist der gegnerische Agent deaktiviert, werden alle mit ihm verbundenen Blöcke gelöst. Diese können anschließend vom angreifenden Agenten mit clear-Aktionen entfernt werden. Neben dem Deaktivieren des Agenten für einige Runden war auch der Aufwand des Holens der Blöcke für umsonst. Hinzu kommt die Chance, den gegnerischen Algorithmus des Zusammenbauens von größeren Aufgaben zu stören.

In der Umsetzung wurde jeder der zwei in den Turnier-Konfigurationen definierten Zielzonen ein Agent fest zugewiesen. Für eine ausreichend hohe Effektivität muss der ausgewählte Agent die Rolle *digger* annehmen, damit die clear-Aktionen mit einer Wahrscheinlichkeit von 100 % ausgeführt werden.

Für die Umsetzung der Strategie wurden folgende Teilaufgaben gelöst:

- Zuordnung eines Agenten zu jeder Zielzone
- Analyse von Dingen, die mit einem Gegner verbunden sind bzw. sein könnten (direkt an ihn angrenzende, zusammenhängende Blöcke)
- Auswahl eines zu attackierenden Gegners basierend auf den mit ihm verbundenen Dingen und der Distanz zu ihm
- Bewegungen in der Zielzone (um Hindernisse herum, auf den Gegner zu bzw. patrouillieren, wenn sich keine geeigneten Gegner im Sichtfeld befinden)
- Energie des angegriffenen Gegners mitzählen
- Gegnerverfolgung (um die Energie eines Agenten mitzählen zu können und für die Verfolgung und Fortsetzung des Angriffs, wenn er sich bewegt)
- Zuletzt angegriffenen Gegner merken und nicht direkt noch einmal angreifen

Getestet wurde die Strategie, indem die eigenen Agenten auch als Gegnerteam eingesetzt wurden. Die Strategie ist nur wirksam, wenn sich gegnerische,

Blöcke tragende Agenten ausreichend lange in der Zielzone nicht bewegen. Gegen 1-Block-Aufgaben ist die Strategie dadurch völlig wirkungslos, da die gegnerischen Agenten dabei in die Zielzone laufen und sofort die Aufgabe abgeben.

6.3 Sonstiges

Zu Beginn der Gruppenarbeit wurde eine mögliche Architektur für den Umgang mit den *Desires* anhand von UML-Diagrammen entwickelt. In der Folge wurde prototypisch eine Architektur umgesetzt. Diese stellte sich jedoch als nicht praktikabel heraus, da die *Desires* sehr kleinteilig aufgeteilt und in jedem Schritt die *Desire*-Objekte neu erzeugt wurden und somit Zustände nicht gespeichert werden konnten. Das Konzept wurde deshalb überarbeitet.

Zwischenzeitlich wurde außerdem ein *Desire* entwickelt, bei dem ein Agent eigenständig alle Blöcke für beliebig komplexe Aufgaben sammelt und zusammenbaut. Dies wurde jedoch zugunsten von kooperierenden Agenten, die sich gegenseitig Blöcke holen, abgebrochen.

7 Gruppenbeitrag Björn Wladasch

7.1 Kartengrösse bestimmen

Um dem Problem der endlosen Karte zu begegnen sollte eine Methode entwickelt werden die es ermöglicht, die tatsächliche Kartengröße zu bestimmen. Die Basisidee wie soetwas umgesetzt werden könnte, wurde der Dokumentation des letzten Wettbewerbs [12] entnommen.

Da in der Literatur aber keine vollständige Erklärung des verwendeten Algorithmus angegeben war, wurde mit verschiedenen Ansätzen zur Bestimmung der Kartengröße experimentiert. Unter anderem wurde versucht aus dem initialen Abstand zweier Agenten, der Anzahl der gemachten Schritte (in Richtung der zu vermessenden Achse) und dem Abstand bei erneuter Sichtung des Agenten zu berechnen, wie groß die Karte ist, was aber erfolglos blieb.

Nach zahlreichen Versuchen wurde eine verlässliche Lösung gefunden, die im Folgenden vorgestellt wird.

Dazu muss man zunächst ein paar Vorbemerkungen zur Ausgangslage im betrachteten Agenten (BDIAgentV1) machen. Agenten können zu Gruppen zusammengefasst werden, welche isch danneine Karte teilen, so dass die absolute Positionen der Agenten (im Bezug auf diese Karte) bekannt sind. Mittels einer Methode des zugehörigen Kartenobjektes ist es jederzeit möglich, über den Namen eines Agenten dessen bekannte Position auf der Karte zu erhalten.

Die Koordinaten der *Things*, die von den jeweiligen Agenten gesehen werden, werden in relativen Koordinaten angegeben (also dem Abstand auf der x- bzw. y-Achse) bezogen auf die Position des Agenten. Diese befindet sich immer im Zentrum seines Sichtfeldes, und hat somit die relativen Koordinaten (0,0).

Die Grundidee zur Bestimmung der Kartengröße besteht nun darin, zwei Agenten miteinander zu verbinden (es reicht, dass ein Agent den Namen des

anderen Agenten kennt) und in entgegengesetzte Richtungen loszuschicken, jeweils ein Paar (in der Abbildung sind die beiden Agenten der Einfachheit halber mit W und E benannte) entlang der x-Achse in Richtung Westen bzw. Osten und ein Paar in Richtung Norden bzw. Süden entlang der y-Achse.

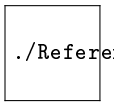

Abb. 6. Startsituation

Auf Grund der sich wiederholenden Karte muss ein Agent der nach Westen läuft zwangsweise seinem Gegenstück das nach Osten gegangen ist wieder begegnen, sobald einer der Beiden (oder beide) die Kartengrenze überschritten haben.In Nord- Süd-Richtung passiert das ganze analog und wird hier nicht weiter betrachtet.

Die mit der Kartenvermessung beauftragten Agenten betrachten nun jeden Agenten aus dem eigenen Team der vor ihnen in Bewegungsrichtung auftaucht und prüfen ob es sich um den gesuchten Agenten handelt.

Die Position diesen entdeckten Agenten (hier als U1 bzw. U2 bezeichnet) auf der gemeinsamen Karte lässt sich bestimmen, in dem die eigene Kartenposition zur relativen Position des gefundenen Agenten hinzu addiert wird. Ist dabei die y-Koordinate (im Nord-Süd-Fall die x-Koordinate) identisch mit der bekannten Position des gesuchten Agenten und die x-Koordinate (bzw. y- Koordinate im Nord-Süd-Fall) entspricht nicht der bekannten Position des gesuchten Agenten so besteht die Möglichkeit, dass der andere Agent gefunden wurde. Wenn dieser an inverser relativer Position ebenfalls einen passend Agenten sieht, haben sich die beiden Agenten wiedergefunden.

Beispiel (in Abbildung) Der Agent U1 ist aus Sicht von Agent W die Koordinaten (-1,-1). Aus Sicht des Agenten E hat U2 die Koordinaten (1,1), was genau die entgegengesetzten Koordinaten sind. Damit sehen die beiden Agenten W und E in dem jeweiligen Agenten U1 bzw. U2 ihr Gegenstück nach Überquerung der Kartengrenze. Die Kartenbreite ist dann | U1.x - E.x | bzw. | U2.x - W.x| .

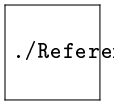

Abb. 7. unbekannte Agenten gefunden

Da auf der Karte aber zahlreiche Hindernisse im Weg der Agenten sein können, ist es wichtig, dass sich die Agenten nciht verpassen. Dazu muss man sicherstellen, dass die beiden Agenten möglichst auf einer vorher festgelegten Li-

nie laufen, damit sie wieder in das Sichtfeld des anderen laufen. Dazu legt man einen y-Koordinaten Wert (im Fall der West- Ost-Vermessung, bei Nord -Süd Vermessung einen entsprechenden x-Wert)) fest auf den die Agenten nach Ausweichmanövern und ähnlichem immer wieder zusteuern.

8 Turniere

Die Gruppe nahm an den Turnieren 2-6 teil. Den Großteil der Spiele bestritt das Agentensystem V1. Das System V2 übernahm jeweils ein Spiel in den Turnieren 2-5.

8.1 Agent V1

Das Agentensystem zeigte sich über alle teilgenommenen Turniere hinweg wettbewerbsfähig. Probleme in einzelnen Begegnungen wurden detailliert dokumentiert und im weiteren Entwicklungsverlauf behoben oder zumindest entschärft. Die Folgenden Absätze geben ein Überblick über die Leistung und Probleme in den einzelnen Turnieren.

Turnier 2 Es war erfreulich, dass 10 Agenten in der Lage waren bis zu 370 Punkte über Einblock-Aufgaben zu erringen. Trotzdem zeigte sich noch großes Verbesserungspotential bei der Befreiung von Agenten aus festgefahrenen Situationen und dem generellen Ausweichen von Agenten untereinander.

Turnier 3 Die erreichte Punktzahl über Einblock-Aufgaben konnte in diesem Turnier auf bis zu 720 gesteigert werden. Dies resultierte hauptsächlich aus besseren Strategien zur Befreiung aus festgefahrenen Situationen. Trotzdem waren weiterhin teilweise übermäßige Gruppenbildung und daraus resultierende gegenseitige Behinderung zu beobachten.

Turnier 4 Die Agenten waren erstmals in der Lage Mehrblock-Aufgaben zu bearbeiten. Das Zusammenspiel der Agenten war leider noch nicht zufriedenstellend, wodurch sich ein Rückgang der erreichten Maximalpunkte auf 680 ergab. Positiv war zu beobachten, dass die Gruppenbildung und daraus resultierender gegenseitiger Behinderung im Vergleich zum Vorturnier abgenommen hat.

Turnier 5 Im Gegensatz zu den vorhergegangenen Turnieren erfolgte nur eine geringe Weiterentwicklung der Agenten. Es wurde jedoch ein massiver Fehler in der Zuweisung von Agenten zu Gruppenaufgaben behoben. Dies führte zu stark verbesserter Leistungsfähigkeit und sicherte der Gruppe den Turniersieg mit maximal 1300 erreichten Punkten. In diesem Turnier wurde erstmalig in einzelnen Begegnungen aggressive Strategien gegen die gegnerischen Agenten eingesetzt.

Turnier 6 Das Turnier 6 bot neue Herausforderungen, da die Kartengröße nicht bekannt war, die Zielzonen sich bewegten und Meteoriteneinschläge auf der Karte ergänzt wurden. Die Agenten bewältigten die neuen Herausforderungen zufriedenstellend und konnten auch dieses Mal den Turniersieg mit einer Maximalpunktzahl von 910 sichern. Die Schwierigkeitserhöhung war deutlich in der Durchschnittspunktzahl sichtbar. Trotzdem konnten die Agenten in einzelnen Spielen die erhöhte Schwierigkeit durch die Abgabe von Dreiblock-Aufgaben kompensieren.

Interessant war weiter zu beobachten, dass speziell die Gruppe 1 sehr aggressiv gegen die Agenten vorging und dadurch eine spannende Begegnung entstand. Das Abwehren solchen Verhaltens wäre eine potenzielle Verbesserungsmöglichkeit für das Agentensystem.

Bonusspiel - Jeder gegen Jeden Zum Abschluss wurde ein Spiel mit 6 x 25 Agenten durchgeführt. Trotz der Vielzahl an gegnerischen und eigenen Entitäten zeigten sich die implementierten Strategien erfolgreich. Es wurden 1370 Punkte erreicht und das Spiel trotz erhöhter Gruppenbildung und gegenseitiger Störung mit deutlichem Abstand gewonnen. Positiv war zu beobachten, dass die 25 Agenten weiterhin performant arbeiteten und in sämtlichen Schritten alle Instanzen eine Aktion an den Simulationsserver übermittelten.

8.2 Agent V2

Bei den Turnieren, mit Ausnahme von Turnier 6, sind immer beide Agenten erfolgreich zum Einsatz gekommen. Bei Turnier 6 war der Agent V2 nicht dabei, da er sowohl Probleme mit dem Erkennen der wechselnden Goal Zones hatte, vor allem mit dem Wegfall von Goal Zones, als auch mit dem Erkennen des Verlustes eines Blocks durch einen Clear-Event und so weniger Tasks als bisher erfolgreich bearbeiten konnte. Im Nachhinein wäre diese Vorsicht jedoch nicht nötig gewesen, da der Leistungsabfall zwar gegenüber dem zu diesem Zeitpunkt genialen Agenten V1 deutlich zu erkennen war, aber die anderen Gruppen anscheinend ebenfalls und mindestens so stark mit diesen neu hinzugekommen Problemen zu kämpfen hatten.

9 Rekapitulation und Ausblick

Die Gruppe konnte einen tiefen Einblick in die theoretischen Grundlagen der Multiagentensysteme erhalten. Die Umsetzung des erarbeiteten Wissens in die jeweiligen Agentensysteme lieferte Lösungen, die speziell in den letzten beiden Turnieren sehr erfolgreich waren (siehe Kapitel 8).
Die Entscheidung zwei Architekturansätze zu verfolgen muss kritisch bewertet werden. Beide Ansätze behinderten sich zwar nicht, es stellten sich aber leider kaum Synergieeffekt ein. Die entwickelten Ziele des Agentensystem V2 wurden auf dessen internen Aufbau optimiert und waren daher nicht für das Agentensystem V1 verwendbar. Daher blieb der erhoffte Austausch verschiedener Lösungen

für das gleiche Problem unter den Systemen aus.
Die Gruppe ist mit der erreichten Funktionalität und deren Qualität zufrieden. Dennoch besteht vielfältiges Verbesserungspotential in der Entscheidungs- und Strategiefindung der Agenten. Eine Weiterentwicklung des Systems durch einzelne Gruppenteilnehmer und eine Teilnahme an einem zukünftigen *Multi-Agent Programming Contest* wird nicht ausgeschlossen.

10 FAQ

10.1 Teilnehmer*innen und ihr Hintergrund

Was war die Motivation an dem Praktikum teilzunehmen?
Der spannende Bereich Künstliche Intelligenz gepaart mit dem kompetitiven spielerischen Thema.

10.2 Statistiken

Wurden die Agenten von Grund auf neu implementiert oder auf einer bestehenden Lösung aufgebaut?
Beide Agentensysteme wurden auf Basis des *javaagents* Gerüst der MASSim (*Multi-Agent Systems Simulation Platform*) aufgebaut.

Wie viel Zeit wurde in die Entwicklung und Organisation des Praktikums gesteckt?
Die investierte Zeit variierte stark innerhalb der Gruppe. Im Mittel wurden ein bis zwei Stunden pro Tag für die Bearbeitung der Aufgabe aufgewendet.

Wie war die investierte Zeit im Verlauf des Praktikums verteilt?
Die Gruppenmitglieder ergänzten sie in diesem Punkt wodurch eine gleichmäßige Verteilung entstand.

Wie viele Zeilen Code wurden ungefähr geschrieben?
Die Gesamtlösung umfasst ca. 25.000 Zeilen inklusive Kommentaren.

Welche Programmiersprache und Entwicklungsumgebung wurde verwendet?
Es wurde die Programmiersprache Java in der Version 17 mit den Entwicklungsumgebungen Eclipse, IntelliJ IDEA und Visual Studio Code verwendet.

Wurden externe Werkzeuge/Bibliotheken verwendet?
Ja, eine Auflistung kann der Projektwebseite [9] entnommen werden.

10.3 Agenten-System Details

Wie entscheiden die Agenten, was sie machen sollen?

- Agent V1: Die Ziel- und Absichtsfindung wurde detailliert in Kapitel 4.4 beschrieben.

- Agent V2: Ein Agent durchläuft in jedem Step alle Desires und prüft ob sie in seinem momentanen Zustand (Belief) möglich bzw. ausführbar sind. Für alle ausführbaren Desires wird dynamisch, abhängig auch vom Arbeitsstand des Desires, eine Priorität vergeben. Das Desire mit der höchsten Priorität wird dann zur Intention welche in diesem Step vom Agenten ausgeführt wird.

Wie entscheiden die Agenten, wie sie etwas machen sollen?

- Agent V1: Es wurden sowohl deterministische Algorithmen als auch stochastische Faktoren integriert.
- Agent V2:In jedem Desire ist festgelegt welche Aktion, abhängig vom aktuellen Zustand des Agenten, der Umgebung und dem Arbeitsstand des Desires als nächstes ausgeführt werden muss.

Wie arbeiten die Agenten zusammen und wie dezentralisiert ist der Ansatz?

- Agent V1: Die gewählte Schichtenarchitektur ermöglicht dem Agentensystem sowohl dezentrale als auch zentrale Entscheidungen zu treffen.
- Agent V2: Die Agenten (der Master) suchen sich selbst ihre Hilfen beim Zusammenbauen der Tasks. In der Klasse AgentsCooperations wird festgehalten wer gerade in welcher Rolle (Master, Helper) mit welcher Task beschäftigt ist. Für alle Agenten ist dort auch der Stand der Taskabarbeitung aller anderen Beteiligten ersichtlich. Es existiert also im Grunde keine zentrale Stelle zur Koordination der Agenten. Die Zusammenarbeit ist individuell in den jeweiligen Desires der Agenten geregelt.

Kann ein Agent das generelle Verhalten zur Laufzeit ändern?

- Agent V1: Die Agenten besitzen keine festen Rollen. Das Verhalten wird dynamisch an das Simulationsgeschehen angepasst.
- Agent V2: Ein Agent führt immer die Aktionen aus welche, abhängig von seinem aktuellen Zustand (Beliefs) und seiner aktuellen Umgebung, für die Durchführung seiner aktuellen Intention notwendig sind. Dies wird alles in jedem Step neu evaluiert.

Wurden Änderungen (z.B kritische Fehler) während eines Turniers vorgenommen?

- Agent V1: In den beiden letzten Turnieren wurde die Aggressivität der Agenten auf die Fähigkeiten der gegnerischen Teams individuell angepasst.
- Agent V2: : Es wurden keine Änderungen während eines Turniers vorgenommen.

Wurde Zeit investiert um die Agenten fehlertoleranter zu machen? Wenn ja, wie genau?

- Agent V1: Die Verifikation und Problemfindung war ein essenzieller Entwicklungsteil. Eine detaillierte Beschreibung kann Abschnitt 5 entnommen werden.
- Agent V2: Ja, es wurden viele Testläufe gemacht und auftretende Fehler versucht zu beheben. Außerdem wurden alle Desires so angelegt, dass, falls ein aktueller Zustand in den Algorithmen nicht abgedeckt sein sollte, das Desire nicht ohne Aktion, sondern zumindest mit einer Skip-Action verlassen wurde. Dies sollte das „hängenbleiben“ des Steps verhindern.

10.4 Szenario und Strategie

Was ist die Hauptstrategie der Agenten?

- Agent V1: Möglichst umfangreiche Informationen über das Simulationsgebiet zu erhalten und anschließend die Bearbeitung von Ein-, Zwei- und Dreiblock-Aufgaben.
- Agent V2: Die Agenten sollen möglichst viele unterschiedliche Blöcke sammeln und in die Nähe einer Goal Zone bringen.

Haben die Agenten selbstständig eine Strategie entwickelt oder wurde diese bereits in die Implementierung eingebaut?

- Die Ziel- und Absichtsfindung erfolgt über klassische Algorithmen.

Wurde eine Strategie implementiert, die Agenten anderer Teams mit einbezieht?

- Agent V1: Ja, gegnerische Agenten werden in der Zielzone angegriffen.
- Agent V2: Nein

Wie entscheiden Agenten, welche Aufgabe sie als nächstes übernehmen?

- Agent V1: Die Ziel- und Absichtsfindung wurde detailliert in Kapitel 4.4 beschrieben.
- Agent V2: : Die Agenten entscheiden es selbst, abhängig von der Verfügbarkeit der für die Aufgabe notwendigen Blöcke. Dabei wird die Anzahl der Agenten beschränkt, welche gleichzeitig an Mehr-Block-Tasks arbeiten können. Dies geschah, um Verklumpung der Agenten in der Goal Zone zu verhindern und Ein-Block-Tasks weiter zu ermöglichen.

Wie koordinieren die Agenten die Arbeit für eine Aufgabe untereinander?

- Agent V1: Die Koordination erfolgte hauptsächlich über die in Abschnitt 4.4 beschriebene Schichtenarchitektur. Zusätzlich kommunizieren Agenten über den Austausch von Nachrichten.
- Agent V2: Die Agenten koordinieren sich bei Gruppenarbeiten (Mehr-Block-Tasks) untereinander, indem sie durch Statusabfragen kommunizieren, wer wann mit was fertig ist und wann mit der nächsten Aktion weitergemacht werden kann.

Welche Aspekte des Szenarios waren am Herausforderndsten?

- Agent V1: Die fragmentierten und stark beschränkten Umgebungsinformationen.
- Agent V2: Die wechselnden GoalZones waren am herausforderndsten. Insbesondere das Entfernen der „alten“ Goal Zones zu erkennen. Ähnlich problematisch war herauszufinden welcher Agent aktuell was attached hat.

10.5 Und die Moral von der Geschichte

Was wurde durch das Praktikum vermittelt?
Es wurden die theoretischen Grundlagen zu Multiagentensystemen und deren praktische Anwendung erlernt.

Welcher Ratschlag wäre für zukünftige Gruppen sinnvoll?
Die Verifikation und Problemfindung ist ein grundlegender und wichtiger Bestandteil der Aufgabe. Es sollten frühzeitig Konzepte und Herangehensweisen zu diesem Thema entwickelt werden.

Was waren Stärken und Schwächen der Gruppe?
Die Gruppe war in der Lage zielgerichtet erfolgreiche Lösungen zu entwickeln. Die fachliche Diskussion über konzeptionelle Entscheidungen und deren Implementierung könnte zukünftig noch verbessert werden.

Was waren Vorteile und Nachteile der gewählten Programmiersprache und weiterer Werkzeuge?
Die Implementierung in Java ermöglichte flexible, individuelle Lösungen. Es konnten zusätzliche Bibliotheken z.B. zur Verwendung von OpenGL eingebunden werden. Im Vergleich zu auf den Anwendungsfall spezialisierten Sprachen, wie z.B. GOAL [11], fiel der Implementierungsaufwand deutlich höher aus.

Welche weiteren Probleme und Herausforderungen kamen im Laufe des Praktikums auf?
Es gab keine nennenswerten Probleme die herausstachen. Es wurde konsequent an der Verbesserung der Einzelsysteme gearbeitet.

Was könnte beim nächsten Praktikum verbessert werden?
Die Schwierigkeitssteigerung zwischen den Turnieren wurde von der Turnierleitung sehr moderat gewählt. Fordernde Konfigurationen hätten die Turniere interessanter gemacht.

Welcher Aspekt der Gruppenarbeit hat am meisten Zeit in Anspruch genommen?
Die Quellcoderezensionen, die vor der Integration in den Hauptstamm der Quellcodeverwaltung durchgeführt wurden.

Literatur

1. Ahlbrecht, T., Dix, J., Fiekas. N. und T. Krausburg: The Multi-Agent Programming Contest 2021, Springer, Heidelberg, 2021
2. Hart, P. E., Nilsson, N. J. und Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, in IEEE Transactions on Systems Science and Cybernetics, 4. Auflage, Nummer 2, Seiten 100-107, Juli 1968
3. Weiss, G.: Multiagent Systems, 2. Auflage, The MIT Press, Cambridge, 2000
4. github.com/agentcontest/massim_2022, agentcontest/massim_2022, https://github.com/agentcontest/massim_2022/blob/main/docs/eismassim.md, EISMASSim Documentation, abgerufe am 21.08.2022
5. Bratman, M.: Intention, plans, and practical reason, Harvard University Press, Cambridge, 1987
6. Kessenich, J., Baldwin, D., Rost, R.: The OpenGL® Shading Language, Version 4.60.7, https://registry.khronos.org/OpenGL/specs/gl/GLSLangSpec.4.60.pdf, abgerufen am 10.09.2022
7. JUnit 5, https://junit.org/junit5, abgerufen am 11.09.2022
8. Projekt-Quellcodeverwaltung, https://github.com/h1Modeling/ss22_fp_mapc_gruppe3, abgerufen am 11.09.2022
9. Projekt-Webseite: https://github.com/h1Modeling/ss22_fp_mapc_gruppe3/tree/master/site/index.html, abgerufen am 13.09.2022
10. Massim 2022 - Issue 7: https://github.com/agentcontest/massim_2022/issues/7, abgerufen am 11.09.2022
11. GOAL: https://goalapl.atlassian.net/wiki/spaces/GOAL/overview, abgerufen am 13.09.2022
12. Ahlbrecht,T. , Dix, J. , Fiekas, N. ,Krausburg T. (Eds.): JaCaMo Builders: Team Description for the Multi-agent Programming Contest 2020/21 in The Multi-Agent Programming Contest 2021 One-and-a-Half Decades of Exploring Multi-Agent Systems, Springer Verlag, 2021, Seite 136
13. Bungartz, H.J., Zimmer, S., Buchholz, M., Pflüger, D.: Modellbildung und Simulation Eine anwendungsorientierte Einführung, 2. Auflage, Springer Spektrum, Berlin Heidelberg, 2013