Fachpraktikum Künstliche Intelligenz: Multiagentenprogrammierung

Alexander Lorenz, Miriam Wolf, Sebastian Loder und Jan Steffen Jendrny

Fernuniversität Hagen, Universitätsstraße 47, 58097 Hagen
https://www.fernuni-hagen.de/

1 Einleitung

In diesem Kapitel wird die Auswahl der technischen Rahmenbedingungen, mit denen die Gruppe gearbeitet hat, genauer erläutert. Ebenfalls ist eine kurze Beschreibung über die Aufgabe, die von den Teammitgliedern bearbeitet wurden, erläutert.

1.1 Auswahl technischer Rahmenbedingungen

Da die Schnittstelle der Programmierkenntnisse aller Gruppenmitglieder auf Java fiel und dies auch mit der MASSim Umgebung sowie der Agentvorlage korrelierte, wurde in dieser Sprache entwickelt. Unsere Agentenvariante lief unter der Bezeichnung *NextAgent*. Alle Klassendateien beginnen mit dem Präfix *Next-*, um die Zuordnung im Sourcecode zu erleichtern. Debugging erfolgte je nach Präferenz des Teilnehmers sowohl über Printausgabe in Echtzeit, als auch über Logfiles und Debugger. In kritischen Bereichen wurden Unittests umgesetzt.

Innerhalb des Quellcodes wurde auf Wunsch eines Teammitglieds eine spezielle Formatierung eingeführt. Dabei wurden die *public* Methoden großgeschrieben, während *private* Methoden weiterhin klein geschrieben wurden. Somit können öffentliche Methoden schneller erkannt und verwendet werden.

1.2 Aufteilung Gruppenmitglieder - Arbeit

Mit dem Ziel, den erarbeiteten Stand und weiteres Vorgehen zu besprechen, wurden wöchentliche Treffen abgehalten. Dieses konnten sehr konsequent umgesetzt werden. Initial wurde innerhalb der Gruppe versucht, in agiler Form mit Hilfe von *issues* in GitHub zu arbeiten. Aufgrund der Komplexität des Themas und mangelnder Vorerfahrung der Teammitglieder, insbesondere im Kontext der Multiagentensysteme, haben wir mehr Zeit für die Einarbeitung des tieferen Verständnisses benötigt.

Aus diesem Grund wurden unser Konzept in vier Themenbereiche aufgeteilt, die wie folgt zugewiesen wurden:

- Herr Jendrny: Festlegen der Aufgaben für den Agenten, basierend auf *Tasks*, Normen und dem aktuellen Zustand der Welt.
- Frau Wolf: Reaktion des Agenten auf die Umgebung und Wahl der auszuführenden *Action*, mit Umsetzung der reaktiven Anteile.
- Herr Lorenz: Verarbeitung der *Percepts* und Interaktion mit dem Server, optimale Wegfindung und Gruppenbildung, sowie Kommunikation.
- Herr Loder: Verarbeitung der Karte und das Zusammenführen der Karten für die Gruppe.

Zusätzlich gab es spontan koordinierte Treffen in kleinen Teams, um Schnittstellen zu diskutieren und Lösungen für Teilbereiche zu erarbeiten.

Alexander Lorenz
Herr Lorenz setzte die Grundvariante des Agenten um, welche die Basis für weitere Entwicklung bildete. Der Schwerpunkt lag auf der sauberen Verarbeitung der Serverdaten, der Möglichkeit zwischen den Simulationen zu wechseln und der Fähigkeit, die ersten *actions* an den Server zu senden. Dabei reagierte der Agent noch rein reaktiv und wählte die nächste Aktion basierend auf einer vorgegebenen Priorisierung.

Im Verlauf des Fachpraktikums setzte Herr Lorenz die Logik der Gruppenbildung um, sowie eine Variante der Kommunikation innerhalb der Gruppe. Des Weiteren beschäftigte er sich mit den Möglichkeiten zur Optimierung der Wegfindungsalgorithmen und erweiterte die initiale A* Variante, um weitere Modifikationen, die in Kapitel *2.2* genauer beschrieben werden. Als Schnittstelle zwischen Percepts, Karte und Entscheidungsfindung wurde viel Zeit in das Bugfixing, Fehlersuche, sowie die Interpretation des Agentenverhaltens investiert.

Bei Herrn Lorenz lag ein Fehler in der Konfiguration der IDE, so dass Github Commits nicht sauber zugeordnet waren. Dies wurde Anfang Juni bemerkt, und behoben.

Miriam Wolf
Um einen Schritt nach vorn zu gehen oder einen Block zu zerstören, müssen die Agenten eine *Action* an den Server schicken. Um einen optimalen Weg durch die Blöcke zu finden und sich zur Endzone durchzukämpfen, muss die nächstmögliche Aktion herausgefunden werden. Frau Wolf hat sich um die Möglichkeit gekümmert, welche Aktion die nächstmögliche ist. Sie hatte die Aufgabe, das Verhalten der Agenten in der lokalen Sicht zu bearbeiten. Die Erklärung der lokalen Sicht des Agenten wird später in Kapitel 2.3 genauer erläutert.

Weiter hatte sie die Aufgabe der Praktikumsorganisatorin inne. Hierfür musste sie die Treffen der Gruppenleiter koordinieren, die Turnierplanung übernehmen und auch die Turniere begleiten. Dazu gehörten unter anderem das Erstellen der Turnierkonfiguration oder das Starten des Servers selbst beim Turnier. Abschließend kam noch die Planung des Präsentationsnachmittags dazu und das Ausarbeiten des allgemeinen Dokumentationsteils.

Sebastian Loder
Kommt noch :)

Steffen Jendrny
Herr Jendrny hat sich um die Aufgabenplanung und -verteilung unter den Agenten gekümmert. Zu Beginn der Entwicklung hat jeder Agent selbstständig entschieden, welche Aufgabe ausgewählt wird und welche Teilaufgabe momentan erfüllt werden muss. Mit der Einführung von Agentengruppen wurde die Aufgabenauswahl auf die Gruppe ausgelagert und ein Agent hat lediglich entschieden, welche Teilaufgabe momentan erfüllt werden muss. Außerdem hat Herr Jendrny zusammen mit Frau Wolf die Abgabe von Aufgaben mit mehreren Blöcken implementiert.
Zusammen mit Frau Wolf war Herr Jendrny bei den Lead-Treffen dabei, um Gruppe 5 zu vertreten.

2 Softwarearchitektur

Im ersten Unterkapitel 2.1 wird zunächst einen Einblick in die Erkundung und die Speicherung der Karte gegeben. Hierfür wird auf die Implementierung und die Aktualisierung der Karte eingegangen sowie auf die Veränderung in Gruppen. Ebenfalls werden die Besonderheiten der wiederholenden Karte und das Koordinatensystem erläutert. Im Kapitel 2.2 wird kurz auf die verwendeten Rollen, die Aufgaben der *Agents* sowie auf die Gruppenbildung und die Wegführung eingegangen. Im Kapitel 2.3 wird der Unterschied zwischen lokaler und globaler Sicht des Agenten aufgezeigt. Als Abschluss in Kapitel 2.4 wird die Kommunikation der Agenten kurz erläutert.

2.1 Erkundung und Speicherung der Karte

In jedem Schritt eines Spiels kann über die *Percepts* abgerufen werden, welche Objekte für einen *Agent* aktuell sichtbar sind. Wie weit ein *Agent* sehen kann, hängt von der Rolle ab. Die von einem *Agent* wahrgenommenen *Things* werden hierbei mit relativer Distanz zur Position des *Agents* angegeben. Wenn sich beispielsweise ein *Obstacle* zwei Felder rechts neben dem *Agent* befindet, wird dieses im *Percept* mit den Koordinaten [2, 0] angegeben. Bewegt sich der *Agent* um ein Feld nach rechts, wird das *Obstacle* dann im Abstand [1, 0] gesehen.

Informationen vergangener *Percepts* sind nicht verfügbar, d.h. ein *Agent* besitzt zunächst einmal kein „Gedächtnis“. Um eine zielgerichtete Planung der *Agents* umzusetzen, ist es daher erforderlich, die in jedem *Step* wahrgenommenen Sichten zu speichern und so Schritt-für-Schritt eine Karte aufzubauen.

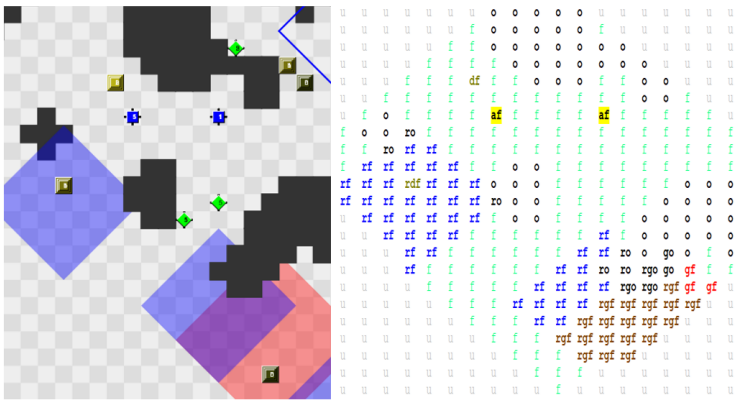

Abb. 1. Links: Screenshot der Karte, Rechts: *NextMap* -Objekt einer Gruppe mit 2 Agents als Output in eine .txt-Datei.
Legende: agent, dispenser, free, goal zones, obstacles, role zones, unknown

Es ergeben sich folgende Anforderungen:

1. Jeder *Agent* soll die in den einzelnen *Steps* wahrgenommenen Dinge speichern und auf diese zu einem späteren Zeitpunkt zugreifen können.
2. Da manche Informationen der Karte dynamisch sind (z.B. können *Obstacles* per *clear()* entfernt werden), ist auch eine Aktualisierung bereits gespeicherter Positionen erforderlich.
3. *Agents* sollten bestenfalls dieselbe Karte verwenden, sodass jeder *Agent* aus der Erkundung / aus dem Wissen der anderen *Agents* schöpfen kann.
4. Die Spiele werden mit einer sich wiederholenden Karte mit unbekannter Kartengröße gespielt. Um eine effiziente Planung zu ermöglichen, soll die Kartengröße ermittelt und die Wiederholungen bei Speicherung der Karte berücksichtigt werden.

Implementierung
Es wurden im Wesentlichen folgende Klassen zur Umsetzung implementiert:

- *NextMapTile*: Beschreibung eines einzelnen Feldes mit den Eigenschaften Position, Typ und *Step* (Schritt, wann die Position zuletzt gesehen wurde).
- *NextMap*: Beschreibung der Karte als Ganzes durch Speicherung der wahrgenommenen *Things* je *Step* als *NextMapTiles* .

Um eine Mehrfach-Belegung einer Position zu ermöglichen (z.B. *Goal Zone* und *Obstacle* auf derselben Position), werden die verschiedenen Typen in jeweils eigenen Datenstrukturen gespeichert (Karte der *Obstacles* , Karte der *Goal Zones* , etc.).
- *NextGroup* : Gruppierung von 1...n *Agents* , welche dieselbe Karte nutzen. Jede Gruppe enthält genau ein *NextMap* -Objekt.

Aktualisierung
Je *Agent* wird geprüft, ob im letzten *Step* ein erfolgreicher *move* -Befehl ausgeführt wurde. Falls ja, wird die Position des *Agents* aktualisiert und es werden die im *Percept* übermittelten *Things* zur Karte hinzugefügt bzw. bereits vorhandene Daten aktualisiert. Hierbei werden *Dispenser* , *Goal Zones* , *Role Zones* und *Obstacles* gespeichert. Andere *Agents* und Blöcke werden in der Karte nicht gespeichert, da sich diese meist sehr dynamisch ändern und daher in der langfristigen Wegeplanung nicht relevant sind.

Gruppen
Bei Start des Spiels wird für jeden *Agent* eine eigene Gruppe erzeugt, welche genau eine Karte enthält. Alle wahrgenommenen Dinge werden in dieser Karte gespeichert und sind zunächst nur für diesen *Agent* sichtbar. Wenn sich zwei *Agents* während des Spiels in entgegengesetzter Richtung wahrnehmen und sie das einzige Paar sind, welches sich in dieser Richtung und Entfernung sieht (Eindeutigkeit), wird die Gruppe des einen *Agent* mit der Gruppe des anderen *Agent* zusammengeführt. Hierbei werden alle *Agents* sowie alle Daten der Karte übertragen. Sofern in beiden Karten Informationen für dieselben Positionen vorhanden sind, bleibt nur das aktuellere *NextMapTile* erhalten und das ältere wird verworfen. Aktualisierungen der Karte aller *Agents* sind anschließend für alle anderen *Agents* der Gruppe sichtbar.

Bei den im Fachpraktikum gespielten Turnieren erfolgte die erste Zusammenführung von Gruppen meist nach nur wenigen Schritten. Innerhalb des ersten Drittel des Spiels haben die Agents i.d.R. in nur noch einer Gruppe agiert, sodass die gemeinsame Datenbasis der Karte über einen Großteil des Spiels genutzt werden konnte.

Wiederholende Karte
Typischerweise werden die Spiele des *Multi Agent Programming Contest 2022* mit einer randlosen Karte gespielt. Die Größe der Karte ist zu Beginn nicht bekannt und die Grenzen sind für die *Agents* bei der Erkundung der Karte nicht erkennbar, sodass sie sich auf einer scheinbar „unendlichen“ Karte bewegen.

Die Kartengröße kann herausgefunden werden, indem sich zwei *Agents* treffen und anschließend in entgegengesetzte x- und y-Richtung laufen, um so die Karte „auszumessen“. Sobald sich die *Agents* erneut treffen, kann durch die Anzahl der zurückgelegten Schritte die Kartengröße bestimmt werden.

Die Bestimmung der Kartengröße konnte leider bis zum letzten Turnier nicht mehr implementiert werden, jedoch wurde die Funktionalität der Karte bereits entsprechend vorgesehen:

Bei Bekanntwerden der Kartengröße kann diese über die Gruppenkommunikation an alle Gruppen bzw. deren Karten mitgeteilt werden. Anschließend werden die Positionen der *NextMapTiles* und der *Agents* , welche größer als die Kartengröße sind, per modulo auf die tatsächliche Kartengröße angepasst. Sofern Informationen für dieselbe Position vorhanden sind, bleibt das aktuellere *NextMapTile* erhalten und das ältere wird verworfen. Weitere Aktualisierungen der Karte passieren nur noch auf der tatsächlichen Kartengröße. Da deutlich weniger Positionen gespeichert werden, ergibt sich ab „Bekanntgabe" der Kartengröße eine deutliche Effizienzsteigerung. Zudem können Optimierungen bei der Wegfindung vorgenommen werden. Tests haben gezeigt, dass deutliche Effizienzsteigerungen v.a. bei der Speicherung erzielt werden können, da die Karte nur noch ein mal in ihrer tatsächlichen Größe (und nicht mehrfach) gespeichert wird.

Koordinatensystem
Im Laufe der Implementierung wurden zwei verschiedene Koordinatensysteme verwendet, welche hier näher beleuchtet werden sollen:

- *Erste Implementierung*: Startpunkt des Agents wird als 0/0 definiert.
  Vorteile dieser Definition sind, dass sich die aktuelle Position eines *Agents* durch Aufsummierung der einzelnen *move* -Befehle ergibt. Die Aktualisierung der Karte erfolgt durch Addition der Position des Agents mit der relativen Postitionsangabe in den *Percepts* . Nachteile dieser Implementierung sind, dass die Datenstrukturen negative Koordinaten unterstützen müssen und v.a. das Debugging deutlich erschwert wird, da eine bestimmte Position auf der Karte verschiedene Koordinaten besitzen kann - je nachdem von welchem *Agent* die Position „gesehen" werden.
- *Finale Implementierung*: Die Ecke oben/links wird als 0/0 definiert.
  Die zuvor genannten Nachteile sind mit dieser Lösung nicht mehr vorhanden, was v.a. das Debugging deutlich erleichtert. Bei „Wachsen" der Karte in negativer Richtung (d.h. nach links oder oben), müssen lediglich alle bereits vorhandenen Positionen aktualisiert werden, was mit vergleichsweise geringem Aufwand möglich ist. Eine Unterstützung von negativen Koordinaten ist mit dieser Lösung nicht mehr erforderlich.

2.2 Entscheidungsverhalten der Agenten

Rollen
Während der initialen Entwicklung des *Agents* wurde mit einer modifizierten *default* Rolle gearbeitet, bei der alle *Actions* für die Verwendung freigeschaltet waren. Ab Turnier vier wurde eine Möglichkeit eingebaut die Rollen dynamisch nach den geforderten Fähigkeiten auszuwählen.

Somit konnte der *Agent* auch auf neue, unbekannte Rollen reagieren. Allgemein wurde die Nutzung der *worker*-Rolle priorisiert, die für den *Agent* bis 2er *task* ausreichend war. Weiterhin können die Rollen *explorer* und *digger* für die Kartenerkundung genutzt werden. Die Rolle *constructor* wurde nicht eingesetzt.

Aufgaben
Um Punkte zu erzielen, müssen *Agents* verschiedene Aufgaben lösen. Der Server teilt den *Agents* mit, wenn es neue Aufgaben zu erledigen gibt. Die *Agents* müssen Blöcke bei den *Dispenser* abholen und in die Goalzones bringen. Damit eine Aufgabe abgegeben werden kann, müssen die *Agents* ihre Blöcke in einer bestimmten Position anordnen. Es gab Aufgaben mit einem, zwei, drei und vier Blöcken. Ab einem fixen Zeitpunkt im Spiel oder nach einer gewissen Anzahl an Abgaben, akzeptiert der Server eine Abgabe dieser Aufgabe nicht mehr. Die Abgabefrist wird den *Agents* bei der Bekanntmachung einer Aufgabe mitgeteilt. Über das Erreichen des Abgabelimits bekommen die *Agents* allerdings keine Information.

Zu Beginn des Praktikums wurde die Klasse *NextTaskPlanner* implementiert. Da sich die Praktikumsgruppe dazu entschieden hat, dass in den ersten Turnieren nur Aufgaben mit einem Block auftreten, sollten unsere *Agents* die Aufgabenplanung nicht absprechen, sondern die individuell beste Lösung finden. *NextTaskPlanner* hat neue Aufgaben entgegengenommen und für alle Aufgaben eigene Pläne erstellt. Die Pläne wurden als Baumstruktur angelegt. Die Wurzel des Baumes war die Klasse *NextPlanSolveTask*. Die Zweige des Baumes waren dann jeweilige Unterpläne, repräsentiert durch verschiedene Klassen der Teilaufgaben. Die Blätter des Baumes entsprechen den auszuführenden Teilaufgaben für jeden *Agents*. Durch eine *pre-order*-Suche wird die aktuell zu erfüllende Teilaufgabe gesucht. Jeden Schritt wird durch die Wurzel des Baumes geprüft, welche Teilaufgaben bereits erledigt sin. Diese werden dann durch die Suche nicht mehr zurück gegeben. Jeder Plan berechnet, wie viele Schritte für die Erfüllung einer Aufgabe notwendig sind. *NextTaskPlanner* wählt dann den Plan aus, der die meisten Punkte in Relation zu den benötigten Schritten verspricht und lässt sich dann die jeweilige Teilaufgabe ausgeben um diese dem *Agents* zu übergeben. Falls eine Aufgabe nicht erfüllt werden kann, weil entweder keine *Goalzone* bekannt ist, oder die benötigten *Dispenser* fehlen, erzeugt *NextPlanSolveTask* die benötigten Unterpläne, um die jeweiligen Sachen auf der Karte zu finden und löscht diese Unterpläne wieder, wenn dem *Agents* alle Orte bekannt sind.

Bei späteren Turnieren wurden Aufgaben freigeschaltet, bei denen mehrere Blöcke abgegeben werden konnten. Da sich die *Agents* zu einer Gruppe zusammenschließen, hat die Gruppe die weitere Planung der Aufgaben übernommen. Der Agent war allerdings weiterhin für die Auswahl der Teilpläne zuständig. Wenn eine Aufgabe mit zwei Blocken ausgewählt wurde, dann sind zwei *Agents* ausgewählt worden, um diese Aufgabe zu erfüllen.

Die Klasse *NextTaskPlanner* in der Gruppe teilte die *Agents* den Aufgaben zu. Dabei wurde die Anzahl der Schritte pro verdientem Punkt minimiert und gleichzeitig verhindert, dass zu oft die gleiche Aufgabe *Agents*paaren zugeordnet wurde, damit die *Agents* sich auf verschiedene Dispensertypen verteilen. Die Klasse *NextTaskPlanner* bestimmte auch, welcher *Agent* zu welchem *Dispenser* läuft. Dazu wurden in den *Agents* ermittelt, wie viele Schritte benötigt werden, um den *Dispenser* zu erreichen und die effizienteste Aufteilung gewählt. Es kann ebenfalls vorkommen, dass Aufgaben mit einem Block effizienter waren oder es eine ungerade Anzahl an *Agents* in einer Gruppe gab. In diesem Fall wurden den *Agents* Aufgaben mit einem Block zugeordnet.

Die Entscheidung, welche Teilaufgabe zum aktuellen Zeitpunkt erfüllt werden muss, wurde dann durch die Klasse *NextTaskHandler* berechnet. Dies geschah wiederum durch die oben beschriebene Baumstruktur.

Wegfindung

Das *MASSim* Szenario hat ganz besondere Anforderungen an die Wegfindungsalgorithmen. Es wird ein zweidimensionales Gitter mit 4 Nachbarn (oben, unten, links und rechts) genutzt, ohne die Diagonalen zu berücksichtigen. Die Bewegungskosten sind einheitlich, jedoch lassen sich manche Hindernisse (*obstacle*, *block*) zerstören. Dies hat zur Folge, dass die Karte einem permanentem Wandel unterliegt was Optimierungsalgorithmen, die auf Vorausberechnungen basieren, ausschließt. Es wurden ebenfalls keine klassischen *Multi-Agent Path Finding (MAPF)* Algorithmen verwendet, da der Agent innerhalb der Simulation nur die Kontrolle über einen Teil der Einheiten verfügt und somit immer reaktiv reagieren muss.

Nach ersten Versuchen mit rein zufallsbasierten Bewegungsmustern, wurden das Konzept für die Planung der Bewegungen analog zur Berechnung der Manhattandistanz, sowie zur Erforschung der Karte anhand eines Spiralmusters konzipiert. Letzteres funktionierte sehr gut bei einem *Agent*, erwies sich aber bei mehreren als ineffizient, da das bekannte Gebiet oft unnötig mehrmals durchlaufen wird. In der finalen Version wird deswegen immer das *Manhattan*-Verfahren verwendet, sobald sich ein Zielpunkt außerhalb des der Karte bekannten Gebiets befindet. Ansonsten nutzt der *Agent* für die Berechnung des Weges den A* Algorithmus. Hier wird jedoch zwischen der Planerstellung und Planausführung unterschieden.

Für die Berechnung der Optionen kann man annehmen, dass das Ziel immer im bekannten Gebiet der Karte liegt. Die momentane Position der anderen *Agents* und Blöcke wird ignoriert, da davon ausgegangen wird, dass diese sich zum Ausführungszeitpunkt nicht mehr an dieser Position befinden werden. Als Berechnungsoptimierung wird das *A* Jump Point Search* Verfahren (A* JPS) [5] von Harabor und Grastien verwendet. Es nutzt das Konzept der Pfadsymmetrie, um die Menge der zu untersuchenden Punkte einzusparen.

Dabei sind zwei Pfade dann symmetrisch, wenn sie den Start- und Endpunkt teilen und der eine aus dem anderen abgeleitet werden kann, wenn die Reihenfolge der sie bildenden Vektoren vertauscht wird.

Bei der Planausführung wird das klassische A* Verfahren erweitert, indem das Konzept des Wegspeichers eingeführt wird. Die Karte erfasst den geplanten Pfad, und blockiert den Weg, falls sich zu dem entsprechenden Zeitpunkt an dieser Position ein anderer Agent befindet. Wird die Ausführung einer Bewegung durch ein Ereignis unterbrochen, wird der geplante Weg wieder freigegeben. Das Verfahren optimiert vor allem das Agieren der *Agents* bei einer lokalen Anhäufung.

Das gleiche Problem wird auch über die lokale Neuanpassung des Pfades gelöst. Ist der nächste Schritt des *Agents* blockiert, wird basierend auf den Informationen der lokalen Sicht, wie in Kapitel *2.3* beschrieben, ein Teilabschnitt neu berechnet, welcher neben *obstacle*, auch *block* und *agent* berücksichtigt. Diese Neuberechnung wird jedoch aktiv während der Wahl der nächsten Aktion getriggert.

Als eine letzte Optimierung der Wegeberechnung wurde die Zentrierung der Karte umgesetzt. Diese erfordert zwar eine bekannte Kartengröße, durch die Möglichkeit des *Agents* über den Rand zu gehen, wird aber automatisch die optimale Distanzheuristik im A* Algorithmus angewendet.

Gruppenbildung

Der Ablauf der Gruppenbildung stützt sich auf den Ansatz, der von der Gruppe FitBut in „The Multi-Agent Programming Contest 2021“ beschrieben wurde. *“If two agents see other agent at the same distance but in the opposite direction, and no other agent sees another agent at the same distance and direction, the two agents can be sure that they see each other.”*[4]

Für die Umsetzung wurde die durch *BasicAgent* bereitgestellte Kommunikationsplattform verwendet. Vor der Verarbeitung der *Percepts*, besitzt jeder Agent den Wissenstand der letzten Runde. Dieser Umstand wird genutzt, um auf den synchronen Datenstand zurückgreifen zu können. Befindet sich in der lokalen Sicht ein anderer Agent, wird eine allgemeine Nachricht mit den gesichteten Koordinaten an alle rausgeschickt. Jeder Agent prüft nun, ob sich lokal auf der gegenüber liegenden Koordinate ein Agent befindet. Diese Information wird an den ursprünglichen Sender zurück übermittelt.

Diese Nachrichten werden gesammelt, und im nächsten Schritt ausgewertet. Sehen sich genau zwei *Agents*, wird der Gruppenbildungsprozess initialisiert. Bei mehr als 2 *Agents*, werden diese für diese Koordinate während der Runde gesperrt.

2.3 Globale und lokale Sicht

Die Sicht der *Agents* können in eine globale und eine lokalen Sicht unterschieden werden. Die globale Sicht besteht aus einer gespeicherten Karte pro *Agent*, die sich durch die Bewegungen der *Agents* auf der Karte erweitert. Hier werden die verschiedenen Dinge wie *Dispenser*, *Blöcke* oder Zonen gespeichert. Sobald sich die *Agents* in Gruppen finden, werden die Karten synchronisiert. Eine genauere Erklärung zu den Karten befindet sich im Kapitel *2.1*.

Die lokale Sicht der *Agents* ist auf eine festgelegte Größe beschränkt. Die Sichtweite der *Agents* wird in der Serverkonfiguration festgelegt. Bei einer Sichtweite von 5 sieht der *Agent* 5 Kästchen nach links, rechts, oben und unten, wie in Abbildung 2 dargestellt.

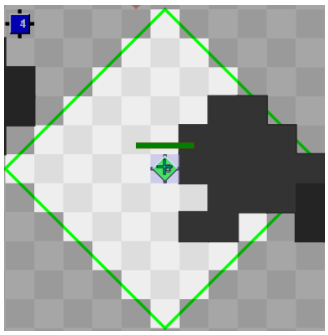

Abb. 2. Sicht des *Agents*

Im ersten Schritt ermittelt der *Agent* einen möglichen Weg, wie er zu seinem Ziel gelangt. Hierfür wird der entsprechende Wegalgorithmus verwendet, welcher in Kapitel *2.2* genauer erläutert wird. Sobald der Weg ermittelt wurde und bevor ein Schritt des Weges beschritten wird, prüft der *Agent* auf Aktionen, welche vorher ausgeführt werden müssen. Diese Aktionen können einen Rollenwechseln beinhalten, ungenutzte Blöcke fallen lassen, einen Block vom *Dispenser* anfordern oder einen Block aufnehmen, eine Aufgabe in der Endzone abgeben oder sich mit einem anderen *Agents* verbinden. Diese Aktionen sind Momentaufnahmen, die der *Agent* ausführen muss, bevor er zu einem neuen Ziel geht. Wenn keine dieser Aktionen passt, wird der *Agent* den Weg zu seinem Ziel gehen. Hier kommt nochmals eine Entscheidungsmöglichkeit für den *Agents* in Frage. Ist der nächste Schritt frei und ich kann den Weg gehen, dann geht er diesen. Sollte aber beispielsweise ein Block in der Richtung sein, in die der *Agent* gehen möchte, so muss er diesen zunächst zerstören. Wenn ein *Agent* in der Richtung steht, so wird ein Weg um diesen *Agents* herum erstellt. In Abbildung 2 wäre der Weg in Richtung Westen durch einen Block gehindert, sodass er diesen zunächst zerstören muss, bevor er in diese Richtung gehen kann.

Zusammengefasst muss der *Agent* in jedem Schritt entscheiden, ob es eine Aktion gibt, die gerade notwendig ist, wie einen Block aufzunehmen oder ob der Schritt, den er gehen möchte, möglich ist.

2.4 Synchronisation und Kommunikation

Die Kommunikation der *Agents* funktioniert, sobald sie sich in einer Gruppe befinden. Die Synchronisation der Gruppen ist in Kapitel *2.2* genauer beschrieben. Für die Kommunikation wurde eine Schnittstelle entwickelt, welche die Nachricht, den Senderagenten und den Empfängeragenten in einer Nachrichtenbox bereithält. Stehen zwei *Agents* um einen *Dispenser* und beide möchten einen Block anfordern, so wird der Agent zunächst prüfen, ob eine Nachricht für ihn vorliegt. Ist dies nicht der Fall untersucht der Agent, ob andere, zu dem Team gehörende *Agents,* in der nähe des *Dispenser* stehen. Wenn die Prüfung erfolgreich ist, so wird er eine Nachricht an den *Agent* senden und dieser wird dann warten, bis der *Dispenser* frei ist. Er selbst stellt eine Anfrage an den *Dispenser* und nimmt den Block dann auf.

3 Turniere

In diesem Kapitel wird auf die Turniere eingegangen. Zunächst werden die Schwierigkeiten im Turnierverlauf erklärt und anschließend passende Lösungsstrategien. Im letzten Kapitel wird kurz auf die Interaktion mit und gegen andere Agenten beschrieben.

3.1 Schwierigkeiten im Turnierverlauf

Zu Beginn der Turniersaison steckten die *Agents* noch in den Kinderschuhen. Sie mussten sich zunächst auf der Karte orientieren und das gesehene Verarbeiten. Die Kartierung war zu Beginn noch ein zufälliger Weg, der in weiterer Folge der Turniere auf einen Spiralförmigen Weg bis hin zu dem richtigen Wegfindungsalgorithmus A* ausgearbeitet wurde. Die Kommunikation mit dem Server musste erst umgesetzt und verstanden werden ebenso wie die abzugebenden Aufgaben. Die Probleme zu Beginn lagen darin, nicht unnötig gegen Wände zu laufen oder sogar die Blöcke zu zerstören.

In weiterer Folge der Turniere wurden die Hindernisse zerstört und die *Agents* sind gezielt zu den Dispensern gelaufen, sobald sie welche gesehen haben. Die ersten Versuche mit dem A* waren weniger erfolgreich und stellte das Team vor eine größere Herausforderung, die es zu Lösen gab. Ein weiteres Problem bestand darin, dass die Agenten zu lange gebraucht haben, um die *Dispenser* oder die *Goal Zones* / *Role Zones* zu finden.

Bei Halbzeit des Praktikums haben die *Agents* einen großen Schritt nach vorn machen können. Sie konnten sich den Weg durch die *Obstacles* schlagen und die benötigten Blöcke in die korrekte Position drehen. Sie konnten mittels einer Aktion *survey* schneller zu dem gesuchten Ziel gehen, was das Team vor ein weiteres Problem gestellt hat.

Bei der Karte setzten wir vom Beginn der Entwicklung auf eine randlose Karte. Dies führte zu einigen Problemen, die auch unsere Performance in den Turnieren stark beeinträchtigten. Z.B. trat ein Fehler im Verhalten zu einem bestimmten Zeitpunkt auf, welcher zuerst als gewolltes Feature fehlinterpretiert wurde. Die *Agents* wanderten nach links oben und haben dadurch die Größe der Karte massiv gestreckt, wie in Abbildung 3 dargestellt. Dies führte zu sehr vielen redundanten Bewegungen und kostete leider unnötig Zeit.

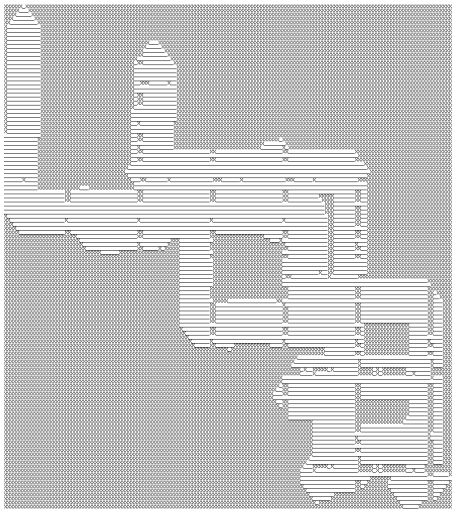

Abb. 3. Beispiel für eine gestreckte Karte basierend auf einer 24x24 Welt

In unseren Versuchen setzten wir auf spezialisierte Umgebungen, um Teilbereiche zu optimieren. Hier trat das Verhalten nur selten auf und wurde dadurch kaum beobachtet. In der Turnierumgebung wurde es jedoch sehr dominant, und spätestens im fünften Turnier wurde dieses Verhalten als extrem kritisch eingestuft und konnte nach einiger Zeit beseitigt werden. Bis zum vierten Turnier hatten wir noch Probleme mit der Synchronisierung der Karte, welche durch dieses Verhalten verschärft wurden. Das nächste Problem, welches das Team zu bewältigen hatte war die Auswahl der Tasks nach Profitabilität und die Gruppenbildung, damit die *Agents* im Team die Aufgaben lösen können.

Im letzten Drittel des Praktikums konnten die *Agents* Aufgaben auswählen und in einer Gruppe zusammenarbeiten. Hierbei ist aufgefallen, dass sich die *Agents* gegenseitig im Weg standen und somit sich selbst blockiert haben. Dieses Problem hat eines der Gruppenmitglieder extrem lange aufgehalten.

Beim Abschlussturnier haben die *Agents* Aufgaben profitabel ausgewählt, in Gruppen zusammengearbeitet, die benötigten *Dispenser* oder Zonen erfolgreich gesucht und sind mit einem performanten Weg zu den Zielen gelaufen. Die gegenseitige Blockierung konnte mittels Kommunikation gelöst werden, sodass am Ende viele Punkte erzielt werden konnten.

3.2 Lösungsstrategien

Die Lösungsstrategien werden zu den Themenbereichen aufgeteilt und kurz erläutert:

Wegfindung Zu Beginn sind die *Agents* zufällig über die Karte gelaufen. Als einfachen Wegfindung wurde dann ein Spiralförmiger Weg gewählt, um die Karte zu erkunden und zufällig passende *Dispenser* zu finden. Die Entwicklung des A* begann ziemlich früh und wurde die meiste Zeit des Praktikums optimiert und weiterentwickelt.

Kommunikation Die Kommunikation unter den *Agents* wurde bei uns nur in der Gruppe umgesetzt. Da die Agenten meistens in der Mitte des Spiels zu einer Gruppe gefunden haben, hat es uns nicht eingeschränkt. Hierfür wurde eine Art Nachrichtenbox erstellt, aus der die *Agents* Nachrichten auslesen oder hineinlegen konnten. Auch das Problem mit dem gegenseitigen Blockieren konnte mittels der Kommunikation gelöst werden

Karte Während des Suchens der verschiedenen Ziele wurde unsere Karte immer nach links oben erweitert. Dies war ein simpler Fehler, der die Gruppe einige Wochen gekostet hat. Dabei hat sich beim zufälligen Suchen ein Fehler eingeschlichen, der die Agenten immer in eine Richtung hat gehen lassen. Nachdem der Fehler behoben war, blieb die Karte kleiner und der Wegfindealgorithmus hat noch besser funktioniert.

3.3 Interaktion mit/gegen andere Agenten

Die anderen Agenten wurden primär als Fremdkörper interpretiert, denen ausgewichen werden konnte, jedoch wurde keine aktive Interaktion mit diesen angestrebt. Ein Ansatz, statt den Agenten die Blöcke anzugreifen, wurde durch das MASSim Regelwerk verhindert.

4 Fazit und Ausblick

Die Gruppe wurde während des ganzen Praktikums und auch der Turniere vor immer neue Herausforderungen gestellt, die es gemeinsam zu Lösen gab. Die Einarbeitung in ein neues, unbekanntes Thema, die Programmiersprache und das Zusammenarbeiten im Team über eine größere Distanz bzw. nur Online war zu Beginn für alle neu. Durch eine offene Kommunikation hat jeder etwas betragen können, jeder der Gruppenmitglieder konnte seine Erfahrungen aus dem eigenen Bereich einbringen können und somit konnte erfolgreich an den Turnieren teilgenommen werden.

Während der ganzen Zeit sind verschiedene Probleme aufgetreten, diese wurden dann diskutiert, bewertet und bearbeitet. Manche Probleme empfanden wir

als kritischer und andere weniger kritisch. Die wöchentlichen Treffen hat zu einem guten Austausch geführt und hat jeden einzelnen in der Entwicklung weitergebracht. Es waren auch nicht so erfolgreiche Turniere dabei und das Team musste lernen, auch mit einer Niederlage oder Enttäuschung umzugehen.

Was die *Agents* betrifft haben wir auch noch einiges an potential, was Zeitlich nicht mehr möglich gewesen wäre. Das Ausmessen der Karte, um ein schnelleres und besseres Ergebnis bei der Karte zu erzielen, war zwar in der Implementierung vorgesehen, aber aufgrund der Zeit nicht mehr umsetzbar. Ebenfalls haben wir auf Ereignisse, die vom Server kamen, keine Reaktion implementiert.

Das Bilden von Kleingruppen mit den ersten beiden Agenten, die sich über den Weg laufen, ist ausbaufähig. Die *Agents* haben zu oft aufeinander gewartet. Hierfür wäre eine dynamische Zusammenarbeit der *Agents* lohnenswerter gewesen.

Literatur

1. https://www.eclipse.org/
2. https://www.jetbrains.com/idea/
3. https://netbeans.apache.org/
4. Ahlbrecht Tobias, Dix Jürgen, Fiekas Niklas, Krausburg Tabajara: The Multi-Agent Programming Contest 2021. One-and-a-Half Decades of Exploring Multi-Agent Systems, pp. 27. Springer Nature, Singapore(2021)
5. D. Harabor, A Grastien: Online Graph Pruning for Pathfinding on Grid Maps. In: In Proceedings of the 25th National Conference on Artificial Intelligence (AAAI). San Francisco, USA (2011)