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

1.1 Auswahl technischer Rahmenbedingungen

Für die Auswahl der technischen Rahmenbedingungen hat sich die Gruppe an der Serverprogrammierung orientiert. Der Server ist in Java implementiert und deswegen sind die Agenten ebenfalls in Java entwickelt worden. Ein weiterer Grund dafür war, dass die Schnittstelle der Programmierkenntnisse aller Gruppenmitglieder auf Java fiel. Zum Einsatz kamen verschiedene Editoren wie Eclipse[1], IntelliJ[2] oder NetBeans[3].

Unsere Agentenvariante lief unter der Bezeichnung *NextAgent*. Alle Klassendateien beginnen mit dem Präfix *Next-*, um die Zuordnung zu erleichtern.

Innerhalb des Quellcodes wurde auf den Wunsch eines Teammitglieds eine spezielle Formatierung eingeführt. Dabei wurden die *public* Methoden groß geschrieben, während *private* Methoden weiterhin klein geschrieben wurden.

1.2 Aufteilung Gruppenmitglieder - Arbeit

Als Kompromiss innerhalb der Gruppe hat man sich auf ein konstantes Treffen 1x jede Woche geeinigt, mit dem Ziel, den erarbeiteten Stand zu besprechen. Dieses konnten wir sehr konsequent umsetzen.

Initial wurde es 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 im Kontext der Multiagentensysteme, haben wir mehr Zeit für das tiefere Einarbeiten benötigt. Deswegen haben wir die Aufgabe in 4 Bereiche aufgeteilt und wie folgt zugewiesen:

- 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.

Debugging erfolgte je nach Präferenz des Teilnehmers sowohl über Printausgabe in Echtzeit, als auch über Logfiles und Debugger. In kritischen Bereichen wurden Unittests umgesetzt.

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* abzugeben. 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 der Optimierung der Wegfindungsalgorithmen, und erweiterte die initiale A* Variante um weitere Modifikationen, die in in Kapitel *2.3* 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 gemerkt, und behoben.

Miriam Wolf
Um einen Schritt nach vorn zu gehen oder einen Block zu zerstören, müssen die Agenten eine Aktion 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, was die lokale Sicht des Agenten ist wird später 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

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, um Gruppe 5 zu vertreten.

2 Softwarearchitektur

2.1 Strategie

2.2 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 und wird basierend auf der Manhattan-Distanz berechnet. 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. 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.

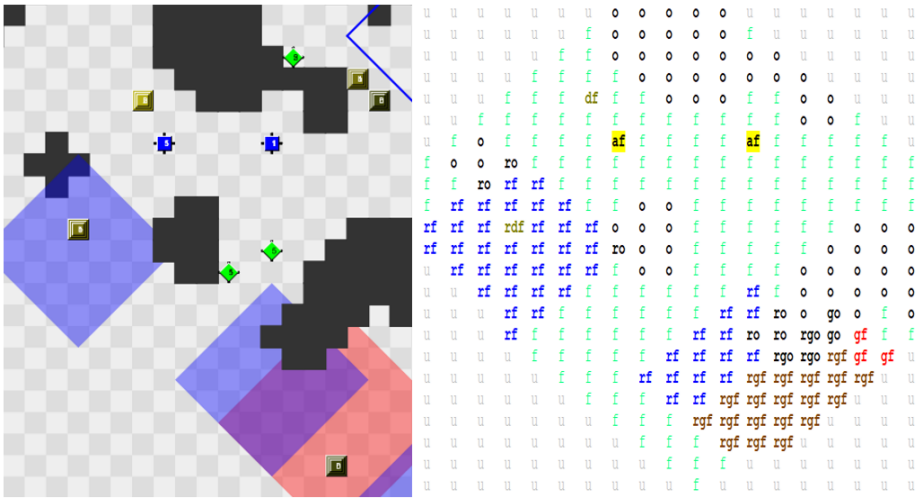

Abb. 1. Links: Screenshot der Karte, Rechts: *NextMap* -Objekt einer Gruppe mit 2 Agents als Output in eine .txt-Datei.
Legende: agent, dispenser, free, goal zones, obstacles, role zones, unknown

4. Üblicherweise werden die Spiele mit einer sich wiederholenden Karte mit unbekannter Kartengröße gespielt. Um eine effiziente Planung zu ermöglichen, sollen 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 werden 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, welche sich in der sog. Moore-Nachbarschaft beliebig oft wiederholt. 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 Koordinatensystem 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.

2.3 Entscheidungsverhalten der Agenten

Rollen

Während der initialen Entwicklung des Agenten wurde mit einer modifizierten *default* Rolle gearbeitet, bei der alle *actions* für die Verwendung freigeschaltet waren. Ab Turnier 4 wurde eine Möglichkeit eingebaut die Rollen dynamisch nach den geforderten Fähigkeiten auszuwählen, so dass der Agent auch auf neue, unbekannte Rollen reagieren konnte. Allgemein wurde die Nutzung der *worker* priorisiert, die für den Agenten bis 2er *task* ausreichend war. Weiterhin können die Rollen *explorer* und *digger* für die Kartenerkundung genutzt werden. Die Rolle *constructor* wurde nicht eingesetzt.

Wir verzichteten in den Turnieren auch auf die Möglichkeit 2er und 3er Schritte auszuführen, da nach einigen Rückschlägen die Komplexitätsreduktion und ein stabil laufender Agent angestrebt wurde.

Aufgaben
Um Punkte zu erzielen müssen Agenten verschiedene Aufgaben lösen. Der Server

teilt den Agenten mit, wenn neue Aufgaben hinzukommen. Die Agenten müssen Blöcke bei den Dispensern abholen und in die Goalzones bringen. Damit eine Aufgabe abgegeben werden kann, müssen die Agenten die Blöcke in einer bestimmten Position angeordnet sein. Es gab Aufgaben mit einem, zwei, drei und vier Blöcken. Ab einem fixen Schritt im Spiel oder nach einer gewissen Anzahl an Abgaben, akzeptiert der Server eine Abgabe nicht mehr. Die Abgabefrist wird den Agenten bei der Bekanntmachung einer Aufgabe mitgeteilt. Über das Erreichen des Abgabelimits bekommen die Agenten 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 Agenten die Aufgabenplanung nicht absprechen, sondern die individuell beste Lösung finden. *NextTaskPlanner* hat neue Aufgaben entgegen genommen und für alle Aufgaben 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 für verschiene Teilaufgaben. Die Blätter des Baumes repräsentieren die auszuführenden Teilaufgaben für einen Agenten. 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 Agenten 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 Agenten alle Orte bekannt sind.

Bei späteren Turnieren wurden Aufgaben freigeschaltet, bei denen mehrere Blöcke abgegeben werden konnten. Da sich bei uns die Agenten zu einer Gruppe zusammenschließen, hat die Gruppe die weitere Planung der Aufgaben übernommen. Die Klasse *NextTaskPlanner* wurde in die Gruppe ausgelagert. Der Agent war allerdings weiterhin für die Auswahl der Teilpläne zuständig. Die vorherigen Aufgaben des *NextTaskPlanner* übernahm die Klasse *NextTaskHandler*.

Wenn eine Aufgabe das Ziel hatte zwei Blöcke abzugeben, dann wurden zwei Agenten ausgewählt um diese Aufgabe zu erfüllen. Die Klasse *NextTaskPlanner* in der Gruppe teilte die Agenten den Aufgaben zu. Dabei wurde die Anzahl der Schritte pro verdientem Punkt minimiert und gleichzeitig verhindert, dass zu oft die gleiche Aufgabe Agentenpaaren zugeordnet wurde, damit die Agenten sich auf verschiedene Dispensertypen verteilen. Die Klasse *NextTaskPlanner* bestimmte auch, welcher Agent zu welchen Dispenser läuft. Dazu wurden die Agenten gefragt, wie viele Schritte benötigt werden um zu einem Dispenser zu laufen und die effizienteste Aufteilung gewählt.

Es kann ebenfalls vorkommen, dass Aufgaben mit einem Block effizienter waren oder es eine ungrade Anzahl an Agenten in einer Gruppe gab. In diesem Fall wurden den Agenten Aufgaben mit einem Block zugeordnet.

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

Klasse	Teilaufgabne
NextPlanExploreMap	Erkundung der Karte
NextPlanDispenser	Gang zu einem bestimmten Dispenser
NextPlanGoalzone	Gang in die nächste Goalzone
NextPlanRolezone	Gang in die nächste Rolezone
NextPlanSolveTask	Wurzel des Baumes
NextPlanSurveyDispenser	Suche nach nächstem bestimmten Dispenser
NextPlanSurveyGoalZone	Suche nach nächster Goalzone
NextPlanSurveyRoleZone	Suche nach nächster Rolezone
NextPlanSurveyRandom	Zufällige Schritte um Karte zu Beginn zu erkunden
NextPlanConnectToAgent	Verbindung zu einem anderen Agenten aufbauen
NextPlanDiscoverMapSize	Kartengröße bestimmen
NextPlanCleanMap	Goalzone und angrenzende Bereiche von Hindernissen befreien

Wegfindung
Das *MASSim* Szenario hat ganz besondere Anforderungen an die Wegfindungsalorithmen. 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-Agenten-Pfadfindungsalgorithmen 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. Das letztere funktioniert sehr gut bei einem Agenten, erwies sich aber bei mehreren als ineffizient, da das bekannte Gebiet oft unnötig mehrmals durchlaufen wird.

In der finalen Version wird deswegen, falls sich ein Zielpunkt außerhalb des der Karte bekannten Gebiets befindet, immer das „Manhattan “Verfahren ver-

wendet. 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 Agenten und Blöcke wird ignoriert, da davon ausgegangen wird, dass diese sich zum Ausfürungszeitpunkt 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 zu dem entsprechenden Zeitpunkt an dieser Position sich 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 Agenten bei einer lokalen Anhäufung.

Das gleiche Problem wird auch über die lokale Neuanpassung des Pfades gelöst. Ist der nächste Schritt des Agenten blockiert, wird basierend auf den Informationen der lokalen Sicht *??*lokaleSicht** 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 Agenten 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 Agenten, wird der Gruppenbildungsprozess initialisiert. Bei mehr als 2 Agenten, werden diese für diese Koordinate während der Runde gesperrt.

2.4 Globale und lokale Sicht

Die Sicht der Agenten 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 des Agenten auf der Karte erweitert. Hier werden die verschiedenen Dinge wie Dispenser, Blöcke oder Zonen gespeichert. Sobald sich die Agenten in Gruppen finden, werden die Karten synchronisiert. Eine genauere Erklärung zu den Karten befindet sich im Kapitel *2.2*.

Die lokale Sicht des Agenten ist auf eine festgelegte Größe beschränkt. Die Sichtweite des Agenten 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.

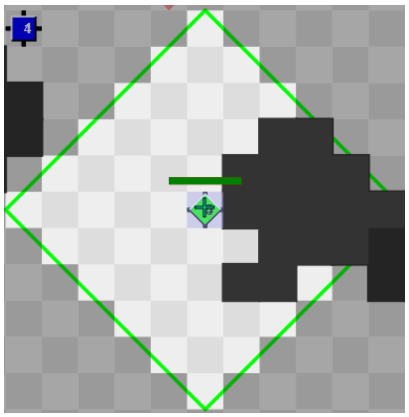

Abb. 2. Sicht des Agenten

Im ersten Schritt ermittelt der Agent einen möglichen Weg, wie er zu seinem Ziel gelangt. Hierfür wird der entsprechende Wegalgorithmus verwendet, welcher in Kapitel *2.3* 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 Agenten 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 Agenten 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 Agenten 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.

2.5 Synchronisation und Kommunikation

Die Kommunikation der Agenten funktioniert, sobald sie sich in einer Gruppe befinden. Die Synchronisation der Gruppen ist in Kapitel *2.3* genauer beschrieben. Für die Kommunikation wurde eine Schnittstelle entwickelt, welche die Nachricht, den Senderagenten und den Empfängeragenten in einer Nachrichtenbox bereithält. Stehen zwei Agenten 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 Agenten in der nähe des Dispensers stehen. Wenn die Prüfung erfolgreich ist, so wird er eine Nachricht an den Agenten 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

Turnier 1 Das erste Turnier wurde nach einer ziemlich kurzen Entwicklungszeit abgehalten. Um die Erfahrung eines Turniers aber nicht zu verlieren, nahm das Team dennoch teil. Zu diesem Zeitpunkt waren die Agenten gerade in der Lage, die Karte zu erkunden. Der Fokus der ersten Entwicklungszeit lag auf der Erkundung der Karte, der Kommunikation mit dem Server und sich mit den Aufgaben und den Gegebenheiten des Turniers vertraut zu machen. Aufgaben konnten zu dem Zeitpunkt nicht abgegeben werden.
Das Ziel für das nächste Turnier war es, die Blöcke zu zerstören und zu den verschiedenen Zonen und Dispensern zu gehen. Eine bessere Wegfindung war ebenfalls notwendig.

Turnier 2 Beim zweiten Turnier wurde die Kartierung verändert, warn allerdings immernoch simpel. Der Agent ist die Karte mit einem spiralförmigen Weg abgelaufen und hat somit verschiedene Dinge entdeckt. Sobald er einen Dispenser in seinem Sichtfeld hatte, ging er hin und die Blöcke wurden aufgenommen.

Die ersten Entwicklungsschritte des A* waren ebenfalls weniger erfolgreich. Die Bewegung des Agenten war das größte Problem.
Für das nächste Turnier sollten die Agenten eine performante Wegfindung erhalten, die unnötigen Blöcke fallen lassen und Aufgaben klüger auswählen.

Turnier 3 Im Turnier 3 konnte der Agent Aufgaben nach Rentabilität auswählen. Das herausfinden von Dispensern und Zonen wurde mittels der Aktion *survey* umgesetzt. Der Agent konnte sich den Weg durch die Blöcke schlagen und den Block in die korrekte Position drehen. In das Abgeben von Tasks wurde vorher viel Zeit investiert. Durch technische Probleme am Turniertag wurden leider wieder keine Punkte gemacht. Es war ziemlich enttäuschend für die Gruppe.
Fürs vierte Turnier soll es endlich Punkte regnen.

Turnier 4 Im vierten Turnier wurde kurz vor dem Turnier ein Fehler entdeckt, der durch die Testkonfiguration nicht aufgefallen ist. Deswegen wurde das Rollenkonzept noch kurzfristig geändert. In dem Turnier konnten dann erfolgreich die ersten Aufgaben abgegeben werden und verschiedene Rollen wurden angenommen. Die Wegfindung erfolgte effizient und die rentabelsten Aufgaben wurden ausgewählt. Allerdings gab es noch Probleme, wenn viele Agenten zusammen sind. Dies wurde vor allem deutlich, wenn sich auch die Agenten des anderen Teams um einen Dispenser oder in einer Goalzone bewegen.

Turnier 5 Beim Vorletzten Turnier hat die Gruppe schon an der Entwicklung der 2er Tasks gearbeitet. Da die Turniere vorher jedoch so schlecht liefen, hat sich die Gruppe entschieden, den Agenten mit 1er Tasks zu starten. Durch die veränderte Konfiguration sind wenige 1er Tasks entstanden und die Agenten haben leider wenige Punkte gemacht.
Fürs letzte Turnier sollten die 2er Tasks unbedingt funktionierten und die Verbindung von zwei Agenten.

Turnier 6 Um beim Abschlussturnier nicht letzter zu werden, wurden vorher noch einige Stunden investiert und es hat sich gelohnt. Die Entwicklung der 2er Tasks war erfolgreich, die Agenten konnten miteinander kommunizieren und sich miteinander verbinden. Die Umschaltung zwischen 1er und 2er Tasks hat sehr gut funktioniert und die Agenten haben sich auch noch selten gegenseitig blockiert.
Alles in allem war es das beste Turnier des Teams und zur großen Erleichterung der Teammitglieder haben die Agenten einiges an Punkten geholt. Außerdem wurde noch eine Strategie gegen Team 1 ausgetestet. Es wurde probiert die Blöcke der Agenten des anderen Teams gezielt zu zerstören. Leider hat sich das Turnier so entwickelt, dass die Teams sich in unterschiedlichen Goalzones aufgehalten haben und unsere Teststrategie deswegen nicht aufgegangen ist.

3.1 Problemerkennung

Im laufe der Turniere sind uns verschiedene Probleme begegnet. Die Aufgaben, die im Hintergrund ablaufen und weniger Reaktion auf spontane Ereignisse erfordern konnten erfolgreich umgesetzt werden. Vor allem der Umgang nicht sich verändernden Bedingungen war schwierig zu implementieren. Unter anderem konnten wir im letzten Turnier feststellen, dass unsere Strategie feste Kleingruppen zu bilden, um Aufgaben mit zwei Blöcken abzugeben, nicht die effizienteste Strategie war, da wir teilweise lange Wartezeiten in Kauf nehmen mussten.

3.2 Verbesserungsmöglichkeiten

Eine Änderung der Strategie hätte die Wartezeiten verringern können. Die Gruppe hätte die wichtigsten Blöcke berechnen und Agenten zuteilen können, die Entscheidung, welche Agenten sich dann miteinander verbinden und eine Aufgabe abzugeben, hätten die Agenten dann unter sich treffen können.

3.3 Lösungsstrategien

Optimierung der Lösungsstrategien

3.4 Interaktion mit/gegen andere Agenten

Optimierung der Strategie zur Handhabung gegnerischer Agenten

4 Fazit

4.1 Schwierigkeiten

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 Agenten 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.

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 Turnier5 wurde dieses Verhalten als extrem kritisch eingestuft, und konnte beseitigt werden. Bis zum Turnier4 hatten wir noch Probleme mit der Synchronisierung der Karte, welche durch dieses Verhalten verschärft wurden.

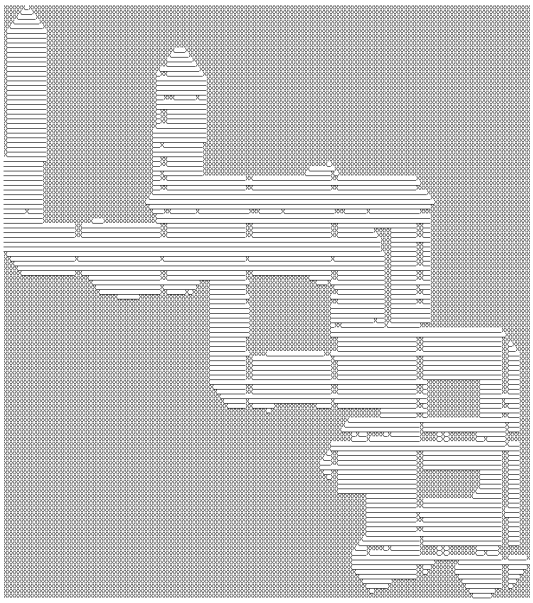

Abb. 3. Beispiel für eine gestreckte Karte basierend auf einer 24x24 Welt

4.2 Ausblick und weitere Möglichkeiten

5 FAQ

Hier stehen wir Frage und Antwort :)

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)