Gruppe 3: Catchphrase?

...[1], ...[1], ...[1] und ...[1]

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

1 Einleitung

Diese Arbeit wurde Rahmen des Fachpraktikums „Multiagentenprogrammierung“ in einer Gruppenarbeit von vier Informatik-Studenten erstellt. Die Aufgabe umfasste die Bearbeitung des „MASSim 2022: Agents Assemble III“ Szenarios [4]. Hierbei interagieren zwei oder mehr Teams auf einer 2-dimensionalen, aus Feldern bestehenden Karte. Jedes Team kontrolliert eine definierte Anzahl an Agenten, die verschiedene Aktionen ausführen können. Daneben existieren noch Hindernisse, Blöcke verschiedener Typen und sogenannte Dispenser, bei denen diese Blöcke erzeugt werden. Einige Felder der Karte sind als Zielzonen und Rollenzonen definiert. Ziel des Szenarios ist es, mithilfe der Agenten Blöcke zu sammeln, zu einer vorgegebenen Konfiguration zusammenzusetzen und in einer Zielzone abzugeben. Für jede erfolgreich bearbeitete Aufgabe gibt es Punkte.

Das Szenario läuft auf einem zentralen Server. Die Simulation ist rundenbasiert. Der Server sendet zu Beginn jeder Runde Informationen über die Umgebung an die Agenten. Diese verarbeiten die Informationen und senden die in der aktuellen Runde auszuführende Aktion an den Server. Dieser prüft die Eingaben der Agenten und aktualisiert den Zustand der Simulation. Die Aufgabe des Fachpraktikums besteht darin, das Agenten-Programm zu entwickeln. Dabei müssen die Agenten die Aufgaben möglichst effizient und kooperativ bearbeiten, um so viele Punkte wie möglich zu erzielen.

2 Kommunikation und Kommunikationsmittel

Die gruppeninterne Kommunikation erfolgte über den für das Fachpraktikum eingerichteten Discord-Server. Wir trafen uns jeden Mittwoch im Sprachchat der Gruppe, um anfangs die grundsätzliche Softwarearchitektur mittels UML-Diagrammen zu entwickeln und später über aktuelle Aufgaben und Probleme zu sprechen. Der Gruppen-Chat wurde darüber hinaus für kurzfristige Absprachen genutzt.

Die Quelltext-Verwaltung erfolgte über das Versionsverwaltungssystem Github, dessen Issue-Management wir ebenfalls verwendeten. Hier wurden alle Aufgaben zusammengetragen und jeweils ein Bearbeiter zugeordnet. Aus dem jeweiligen

Issue wurde jeweils ein Feature Branch erstellt, in dem Bearbeitung der Aufgabe erfolgte. Abschließend wurden die Änderungen des Feature Branches mittels eines Pull Requests 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

Für die programmiertechnische Umsetzung entschieden wir uns für die Programmiersprache Java, da alle Gruppenmitglieder mindestens über grundlegende Kenntnisse dieser Programmiersprache verfügten. Außerdem ist Java eine stark typisierte Programmiersprache, wodurch einige Programmierfehler schon vor der Ausführung des Programms entdeckt werden können und so die Wahrscheinlichkeit von Laufzeitfehlern reduziert wird. Würde ein Laufzeitfehler während einer Simulation auftreten, dann müsste das Programm manuell neu gestartet werden. Während der Reaktionszeit bis zum Neustart läuft die Simulation (das Server-Programm) weiter. Für die während dieser Zeit laufenden Runden werden keine Aktionen gesendet, wodurch diese ungenutzt bleiben. Außerdem gehen alle bisher gesammelten Informationen (Objektzustände) verloren und müssen neu gesammelt werden.

Das objektorientierte Programmierparadigma, auf dem auch Java basiert, weist einige Gemeinsamkeiten mit dem Konzept der Agenten auf. Sowohl Objekte als auch Agenten besitzen einen gekapselten, inneren Zustand, führen Aktionen (Methoden) aus, um diesen Zustand zu verändern und kommunizieren mittels des Nachrichtenversands miteinander. Dennoch gibt es auch eine Reihe von Aspekten, die nicht standardmäßig im objektorientierten Programmierparadigma enthalten sind, aber trotzdem mithilfe objektorientierter Programmiersprachen implementiert werden können. Dazu zählt beispielsweise, dass eine Methode eines Objektes, solange sie öffentlich ist, jederzeit von anderen Objekten aufgerufen werden kann, während ein Agent selbst entscheidet, wann er eine Aktion ausführt. Ein weiterer Unterschied ist, dass Objekte in einem objektorientierten Programm standardmäßig nur einen gemeinsamen Ausführungsstrang aufweisen. Das Konzept der Agenten sieht hingegen einen eigenen Ausführungsstrang für jeden Agenten vor [3].

Das in dieser Arbeit entwickelte Agentensystem basiert auf der BDI-Architektur. Diese Architektur ist eine Abstraktion des menschlichen Denkens bzw. Schlussfolgerns. Das heißt sie versucht zu beschreiben, wie Menschen sich für ihre nächste Handlung entscheiden, um dadurch übergeordnete Ziele zu erreichen. Die Komponenten der BDI-Architektur sind:

- **Beliefs:** Informationen, die der Agent über seine Umwelt besitzt

– **Desires:** Handlungsoptionen bzw. mögliche Ziele des Agenten
– **Intentions:** Ziele, für die sich der Agent entschieden hat und an deren Realisierung gerade gearbeitet wird

Die Beliefs werden dabei durch eine Funktion auf Basis der Wahrnehmung der Umgebung und der aktuellen Beliefs aktualisiert. Die Desires werden aus den Beliefs und den aktuellen Intentions ermittelt. Die eigentliche Schlussfolgerung der nächsten Handlungen, also die Auswahl der Intentions, erfolgt durch eine Filter-Funktion, bei der die aktuellen Beliefs, Desires und Intentions einfließen. Der Vorteil dieser Architektur ist, dass eine funktionale Zerlegung des Agenten in die oben genannten Subsysteme vorgegeben ist. Die Komponenten erscheinen dabei intuitiv, da die BDI-Architektur auf dem menschlichen Denkmodell basiert [3].

Auf der Grundlage der BDI-Architektur erarbeiteten wir anhand von Klassen-UML-Diagrammen konkrete Systemarchitekturen. Die UML-Diagramme stellen aufgrund der übersichtlichen, visuellen Darstellung des komplexen Systems eine gute Diskussionsgrundlage dar.

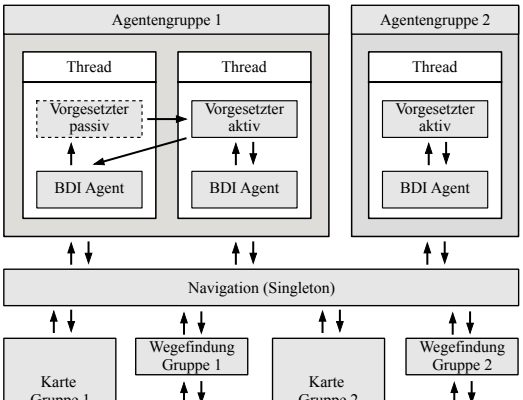
TODO Bdi, 2 Agentensysteme, UML, Java

4 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]), dar.
Als Folge dessen entschied sich der Autor mit dem Aufbau der Wissensverwaltung (siehe 4.2) zu beginnen. Diese umfasst eine Datenstruktur zur Speicherung der von der Simulation übermittelten Informationen, 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) sowie der Implementierung einer intelligenten Wegführung (siehe 4.3) gearbeitet. Um die Ergebnisse der Wegführung und das Entscheidungsverhalten der Agenten effektiv verifizieren zu können, folgte die Erstellung eines graphischen Analysewerkzeugs (siehe 4.5) das im weiten Entwicklungsverlauf stetig weiterentwickelt wurde und sich als sehr hilfreich bei der Fehlerfindung zeigte.

4.1 Agent V1 - Architektur

Das Agentensystem V1 wurde vom Autor konzipiert, in der Gruppe abgestimmt und schließlich selbstständig implementiert.



Es erweitert das BDI-Konzept [5] um zusätzliche Daten-, Berechnungs- und Entscheidungsebenen, die in Abbildung 4 illustriert sind.
Das System kombiniert den Aufbau eines BDI-Agenten (vgl. [3, S. 58]) mit einer bidirektionalen vertikalen Schichtarchitektur [3, S. 61-62], auf die in Abschnitt 4.4 näher eingegangen wird.

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 Vorgesetzeninstanz 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. Es ist weder die Größe des Simulationsgebiets, noch die absolute Position des Agenten bekannt. Um aus den partiellen Umgebungsinformationen eine globale Sicht zu erhalten, werden diese an das Navigationsmodul weitergeleitet und in einer chronologisch fortgeschriebenen Karte zusammengeführt.

Beim Simulationsstart erhält jeder Agent eine Karte mit festgelegter Initialgröße (Abb. 2 Punkt 1), die beim Erkunden der Umgebung erweitert wird (Abb. 2 Punkt 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 und schließlich zusammengeführt (Abb. 2 Punkt 3). Die entstandene Karte ermöglicht nun im weiteren Simulationsverlauf die aktuelle Position aller Agenten einer Gruppe untereinander zu bestimmen. Aus dieser Information kann nun durch jeweils zwei Agenten, die sich in entgegengesetzte Richtungen bewegen

und sich durch das kontinuierliche Simulationsgebiet zwangsläufig wieder treffen, die Kartengröße ermittelt werden. Anschließend wird die Karte beschnitten, wobei die Informationen abgeschnittener Bereiche auf der gegenüberliegenden Seite eingefügt werden und somit nicht verloren gehen (Abb. 2 Punkt 4).

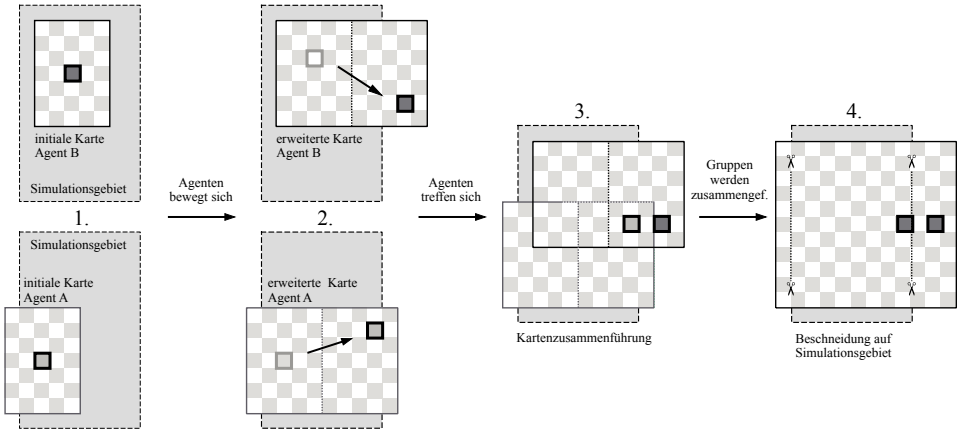


Abb. 2. Erweiterung und Zusammenführung einzelner Karten

4.3 Wegfindung

Unter dem Begriff Wegfindung verstehen wir das Lösen folgender zweier 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 Teilproblems 1 befähigt den Agenten seine Ziel- und Absichtsfindung auf Grundlage von korrekten Entfernungsdaten, anstelle von approximative Annahmen z.B. über die Manhattan-Distanz $d(A, B) = |A_x - B_x| + |A_y - B_y|$, durchzuführen. Wurde ein Ziel priorisiert, muss zu diesem ein ideal kurzer Weg gefunden werden.

Der Autor hat sich bereits zum Projektbeginn als Ziel gesetzt 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 entschieden.

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

Es erfolgte die Implementierung des Suchalgorithmus in der Form eines Computeshaders in OpenGL mittels GLSL (Verweis einführen). Mit dieser Technologie konnte der Algorithmus auf mindestens > 1000 gleichzeitige Berechnungen parallelisiert werden. Es werden hierbei nicht Teile des Algorithmus parallelisiert sondern jeder Berechnungskern führt einen vollständigen Suchlauf zwischen 2 Punkten durch.

PROBLEM SPEICHER UND SEINE LÖSUNG

- Karte ermöglicht Wegfindung - Wie werden Ziele gewählt Grundproblem muss Distanzen kennen um Entscheidung zu treffen - Agenten können Blöcke clearen interessanter aspect - über Eck Die Pathfinding, 2 Stufiges System

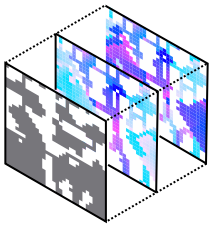


Abb. 3. Datenstruktur Wegfindung

4.4 Ziel- und Absichtsfindung

Desires, verticale Schichtarchitektur

4.5 Verifikation und Problemfindung

Tests / Debugger

4.6 Rekapitulation

Zahlen, sonstiges Server Bug, Turnier gehalten, Github erstellt

4.7 ...

5 Gruppenbeitrag Melinda Betz

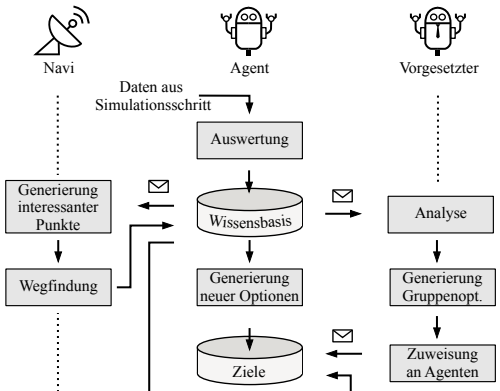
6 Gruppenbeitrag Phil Heger

6.1 Logging

6.2 Strategien zum Stören gegnerischer Agenten

Dispenser blockieren

Goal Zone verteidigen



7 Gruppenbeitrag Björn Wladasch

8 Turniere

Turnier 2

Turnier 3

Turnier 4

Turnier 5

Turnier 6

9 Rekapitulation und Ausblick

Vor- und Nachteile der Entscheidung von zwei Architekturen Was sollte noch verbessert werden Wie sind wir zufrieden

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, 21.08.2022
5. Bratman, M.: Intention, plans, and practical reason, Harvard University Press, Cambridge, 1987