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

Unsere Gruppe besteht aus vier Informatik-Studenten, wovon zwei im Bachelorstudiengang Informatik und zwei im Masterstudiengang Praktische Informatik immatrikuliert sind.

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 die 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 Grundkenntnisse dieser Programmiersprache verfügten. Außerdem bietet Java als stark typisierte Programmiersprache die Möglichkeit bestimmte Programmierfehler schon vor der Ausführung des Programms zu entdecken. Dadurch wird die Wahrscheinlichkeit von Laufzeitfehlern reduziert. Ein weiteres Argument für die Nutzung von Java ist das von den Entwicklern des Szenarios mitgelieferte Grundsystem

des Agentenprogrammes, in dem die Kommunikation mit dem Server schon implementiert ist. Der durch die Nutzung dieses Grundsystems entfallene Aufwand konnte für die Programmierung der eigentlichen Agentenfunktionalität genutzt werden.

In der Dokumentation des vergangenen Multi-Agent Programming Contests berichtete das Team MLFC, dass das Debugging aufgrund der Verwendung einer Vielzahl von zusätzlichen Frameworks und Tools erschwert wurde [3]. Deshalb entschieden wir uns gegen den Einsatz solcher externen Tools.

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 besondere Herausforderung dieses Ansatzes besteht in der Frage, wie die Intentions ausgewählt und wie lange sie verfolgt werden. Werden Intentions zu lange verfolgt, sind die Grundbedingungen für deren erfolgreichen Abschluss ggf. nicht mehr gegeben. Werden sie zu schnell gewechselt, wird u. U. nie eine Intention erfolgreich abgeschlossen [2].

Der Vorteil der BDI-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 [2]. Aufgrund der Ähnlichkeiten zum menschlichen Denkmodell ist diese Architektur gut verständlich und bietet einen einfachen Einstieg die Umsetzung, weshalb wir uns für die Verwendung dieses Ansatzes entschieden.

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.

Aus der Konzeptionsphase resultierten zwei unterschiedliche Ansätze, die in der Folge auch weitestgehend unabhängig umgesetzt und ausprobiert wurden. Dadurch sollten die Vor- und Nachteile der unterschiedlichen Ansätze untersucht werden.

4 Gruppenbeitrag Heinz Stadler

4.1 Agent V1 - Architektur

Aufbau

4.2 Wissensverwaltung

Belief

4.3 Wegfindung

Pathfinding

4.4 Ziel- und Absichtsfindung

Desires

4.5 Verifikation und Problemfindung

Tests / Debugger

4.6 ...

5 Gruppenbeitrag Melinda Betz

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

Als grundlegender Mechanismus für das Stören gegnerischer Agenten ist die clear-Aktion auf die Position des gegnerischen Agenten im Szenario vorgesehen. Ist diese erfolgreich, dann verringert sich das Energielevel des gegnerischen Agent um einen definierten Betrag. Beträgt sein Energielevel 0, dann wird der Agent für eine ebenfalls in der Server-Konfiguration festgelegte Anzahl von Runden deaktiviert. Wenn der Agent deaktiviert ist, verliert er alle mit ihm verbundenen Blöcke. Wenn die Blöcke nicht (mehr) mit einem Agenten verbunden sind, können sie mit einer einmaligen clear-Aktion entfernt werden.

Beim Stören und Deaktivieren gegnerischer Agenten 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 (bei den in den Turnieren verwendeten Konfigurationen) klein. 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 „digger“-Rolle) nur 30 % beträgt.

Für ein wirksames Stören der Gegner ist daher eine komplexere Strategie und das Annehmen der „digger“-Rolle 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 auch 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 „digger“-Rolle 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 nicht noch einmal angreifen (z. B. wenn Blöcke in seiner Nähe sind, die aber mit einem anderen Agenten verbunden sind, dann können diese nicht durch clear-Aktionen entfernt werden und der gleiche Gegner würde immer wieder angegriffen werden)

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 entwickelten wir eine mögliche Architektur für den Umgang mit den Desires anhand von UML-Diagrammen. In der Folge realisierte ich prototypisch eine Architektur. 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 entwickelte ich ein Desire, 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.

Für die Unittests wurde das Mocking Framework „mockito“ eingesetzt, das es ermöglicht das Desire-Objekt, das externe Abhängigkeiten, wie z. B. das Belief-Objekt, besitzt, unabhängig von diesen zu testen, indem die Rückgaben von Methodenaufrufen auf dem externen (nicht zu testenden) Objekt vordefiniert werden.

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. github.com/agentcontest/massim_2022, agentcontest/massim_2022, https://github.com/agentcontest/massim_2022/blob/main/docs/eismassim.md, EISMASSim Documentation, 21.08.2022
2. Weiss, G.: Multiagent Systems, 2. Auflage, The MIT Press, Cambridge, 2000
3. Ahlbrecht, T., Dix, J., Fiekas. N. und T. Krausburg: The Multi-Agent Programming Contest 2021, Springer, Heidelberg, 2021
4. 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
5. Bratman, M.: Intention, plans, and practical reason, Harvard University Press, Cambridge, 1987