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 [1]. 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 [2].

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 [2].

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

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

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. 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