Jason Plays Minecraft

Bardia Parmoun[1][0009−0001−4952−7343] and Babak Esfandiari[1][0000−0002−4441−3435]

Carleton University, Ottawa, Ontario, Canada
bardiaparmoun@sce.carleton.ca, babak@sce.carleton.ca

Abstract. This paper introduces Argus, a BDI multi-agent development environment based on the game Minecraft. The environment comprises non-player characters (NPCs), programmed as BDI agents in Jason, to survive against zombies and to collect as many points as possible by building tools and collaborating. The goal of this paper is to present Argus as a testbed for evaluating Jason in an interactive, multi-agent, collaborative environment. Argus supports the development of diverse agent strategies and the collection of various metrics. Following this, two sample strategies and an analysis of them are provided as starter guides.

Keywords: BDI, Jason, Multi-Agent Systems, Minecraft

1 Introduction

1.1 Motivation

Autonomous agents are often described as computer systems that follow a constant cycle of observation and reaction to their environment to achieve an overall objective [7]. A widely used paradigm for designing such agents is the belief-desire-intention (BDI) model. BDI groups the different aspects of an agent as the three concepts of beliefs, desires, and intentions [6]. AgentSpeak is a theoretical language for describing BDI systems [5]. Jason is a notable implementation of AgentSpeak [2]. While Jason has been applied in a variety of domains, there remains a need for richer environments that support the study of multi-agent behaviours such as competition and coordination. To address this gap, we introduce Argus as an extensible multi-agent testbed built on top of Minecraft.

1.2 Problem Description

Minecraft is a popular game developed by Mojang Studios. In this game, players explore different worlds and defend themselves against enemies [8]. We developed Argus as a Minecraft world with zombies and trees. The players were modeled as autonomous BDI agents with the main objective of surviving the zombie attacks and collecting the highest number of points. Players were free to attack both the zombies and other players; chop trees and collect wood; build houses to hide and recover; build weapons; and form alliances with others to share resources [1].

[1] Demos of Argus are available online: https://youtube.com/playlist?list=PLOsC27zWBS6fVtT3lOkh32wWJWRePIk7G&si=Y2DbMxHCDqOI4o-Y

2 Background

2.1 AgentSpeak and Jason

The belief-desire-intention (BDI) model represents an agent's behaviour in terms of the three concepts of beliefs, desires, and intentions. At a given time, a BDI agent has specific knowledge (beliefs) from its environment; using these beliefs, the agent follows a set of plans (intentions) that help it achieve a goal (desire) [6]. AgentSpeak is a Prolog-like language that is used for specifying these plans [5]. Jason [2] is a popular interpreter for AgentSpeak.

Jason represents beliefs as predicates. It represents desires in the form of achievement and test goals. Achievement goals represent the desire to reach a specific predicate. Similarly, test goals represent the desire to check if a specific predicate is true. Jason also adds a notion of triggers, which represent the addition or removal of specific beliefs, achievement goals, or test goals [2].

2.2 Paper, Citizens & CommandHelper

Various tools exist for creating custom Minecraft modifications. The main tool used for this project was `Paper` [9], which is a Minecraft game server with extensive API support for custom modifications to the game. Paper also supports a library of plugins to extend the server's capabilities. A notable plugin used for this project was the `Citizens` plugin [4], which helped with the creation and maintenance of non-player characters (NPCs). Another plugin used was `CommandHelper` [11], which helped automate various aspects of the game.

2.3 Related Work

There have been many attempts to apply Jason and BDI to different domains. [3] applies Jason to the 2018 Multi-Agent Programming Contest, which involved two teams of agents competing to score points by building and attacking "wells" using a realistic city map. The agents are implemented in Jason and rely on its intention selection functionality to handle concurrent intentions. [10] is an example of applying Jason to a disaster rescue simulation, `NetLogo`. The tool allows the user to populate the map with fires, blockades, obstacles, and victims. The objective of the agents, implemented in Jason, is to rescue the civilians. They use Jason's built-in messaging functionalities to coordinate with each other. We used a similar idea for coalitions in Argus. [1] explores Theory of Mind in Minecraft using LLMs, a paradigm closely related to BDI, and argues that building artifacts in the game requires complex knowledge transfer between agents. We adopted a similar strategy and introduced coalitions to model these complex interactions.

3 Methodology

As previously described, in Argus, the main objective of the agents is to survive zombie attacks. They can build houses to escape or build weapons of different

grades (swords, axes, and tridents) to fight against zombies. Wood is the main currency in this world, which is required for building houses and weapons. To obtain wood, agents can chop the trees around them or ask other agents for donations. The agents are rewarded with points for doing specific actions such as building houses, attacking zombies, donating to others, and surviving the game. Since they want to maximize these points, they might have the incentive to kill other competing agents. They can also form coalitions with one another to share wood and houses, but houses can only fit two agents at a time [2].

3.1 Defining Agents in Jason

We defined the agents in Jason. They would get perceptions from the environment about their inventory (wood, houses, and weapons), their health, nearby trees, zombies, and agents in the form of beliefs. Each agent also had a list of actions available to it for finding trees, zombies, and other agents; chopping wood; building houses and weapons; entering and leaving houses; and escaping attacks. These actions were all external and defined in the agent architecture file `AgArch`. Additionally, the agents used Jason's built-in messaging action (`.send`) to communicate with one another, i.e., to form coalitions and send donations. The agent's strategy was its AgentSpeak (ASL) file, which also included the rules for coalition formations and donations. These could differ between strategies.

Two sample strategies were defined: `capitalist` and `attacker`. The capitalist strategy focused on building as many houses as possible and only built the basic weapon. Conversely, the attacker's strategy focused on maximizing its weapon and only built one house. In both strategies, coalitions were formed at the beginning and were never broken. In the implementation of these strategies, an explicit confirmation was required from both sides when forming a coalition. Each side needed to send a request asking for a coalition and waited for the other party's response. They also required all the agents in a given coalition to have the same strategy. Similarly, these strategies only allowed donations within coalitions and required the receiving agent to be nearby.

3.2 Connecting Jason to Minecraft

The main part of the Argus environment is a Paper server that Minecraft can talk to. Within this server, there are three plugins: Citizens, CommandHelper, and our custom Argus plugin. In this plugin, we created a custom trait file `JasonAgentTrait` in compliance with Citizens. The trait describes the NPC. Finally, the mentioned trait file connects to a custom Jason agent architecture file (`AgArch`), which Jason uses for its perceptions and external actions.

4 Experiments & Results

The current environment allows for the collection and analysis of various metrics, such as each agent's final score, number of wood pieces chopped or donated,

[2] The full implementation is available online: `https://github.com/NMAI-lab/argus`

houses built, weapon upgrades, and health. To show Argus' capabilities, an experiment setup was created with eight agents (four attackers and four capitalists) fighting eight zombies. Three experiments were conducted with different coalition sizes: no coalitions (size 1), two groups of attackers vs. two groups of capitalists (size 2), and all attackers vs. all capitalists (size 4). Each experiment was repeated 50 times, with each round lasting 1.5 minutes. Figure 1 shows how

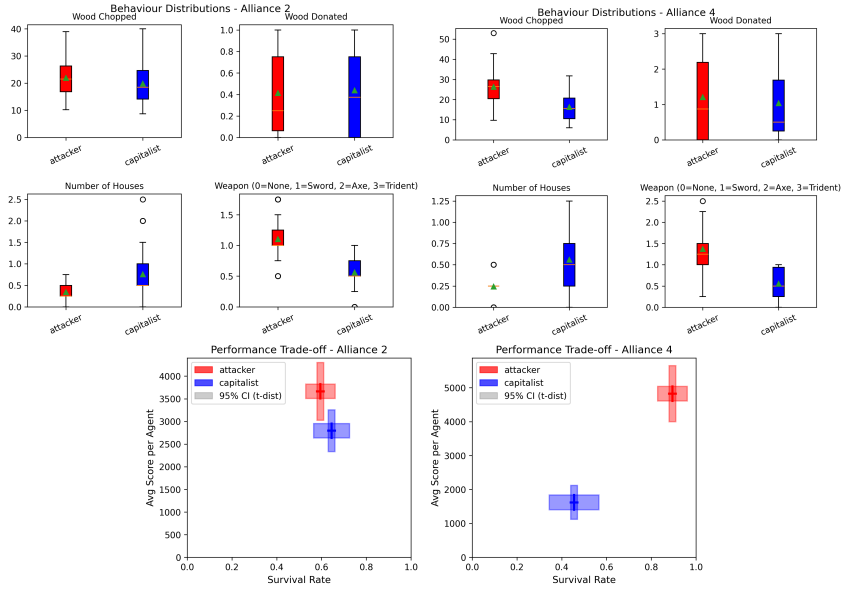

Fig. 1. Analyzing the attacker and capitalist strategies with coalition sizes of 2 and 4.

coalitions change the behaviour of individual strategies. This study is an example of the effects of multi-agent interactions on hard-coded BDI agents.

5 Conclusion and Future Work

In this paper, we described a multi-agent environment based on Minecraft. The agents, programmed in Jason, behaved as NPCs and followed the BDI paradigm. They worked together to defeat zombies and collect points. This environment can be used to investigate the creation of emergent multi-agent behaviours among BDI agents. Future work could examine how an agent's individual plans can be prioritized to serve both itself and its coalition. Additionally, due to the richness of Minecraft, this environment is extensible to new actions and mechanisms.

Acknowledgments. This project could not have been completed without the support of Dr. Alan Tsang from the Department of Computer Science and Mr. Curtis Davies from the Department of Systems and Computer Engineering at Carleton University.

References

1. Bara, C.P., CH-Wang, S., Chai, J.: MindCraft: Theory of mind modeling for situated dialogue in collaborative tasks. In: Moens, M.F., Huang, X., Specia, L., Yih, S.W.t. (eds.) Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 1112–1125. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021). https://doi.org/10.18653/v1/2021.emnlp-main.85
2. Bordini, R.H., Hübner, J.F.: BDI Agent Programming in AgentSpeak Using Jason. Computational Logic in Multi-Agent Systems **3900**(3), 143–164 (2006). https://doi.org/10.1007/11750734_9
3. Cardoso, R.C., Krausburg, T., Baségio, T., Engelmann, D.C., Hübner, J.F., Bordini, R.H.: SMART-JaCaMo: an organization-based team for the multi-agent programming contest. Annals of Mathematics and Artificial Intelligence **84**(1), 75–93 (Oct 2018). https://doi.org/10.1007/s10472-018-9584-z, https://doi.org/10.1007/s10472-018-9584-z
4. Citizens: Citizens API (2025), https://wiki.citizensnpcs.co/API
5. D'Inverno, M., Luck, M.: Engineering AgentSpeak(L): A Formal Computational Model. Journal of Logic and Computation **8**(3), 233–260 (1998)
6. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention Model of Agency. In: Müller, J.P., Rao, A.S., Singh, M.P. (eds.) Intelligent Agents V: Agents Theories, Architectures, and Languages, Lecture Notes in Computer Science, vol. 1555, pp. 1–10. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-49057-4_1
7. Jennings, N.R., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems **1**, 7–38 (1998). https://doi.org/10.1023/A:1010090405266
8. Minecraft: What is Minecraft? (2025), https://www.minecraft.net/en-us/about-minecraft
9. PaperMC: PaperMC (2025), https://papermc.io/
10. Ramirez, W.A.L., Fasli, M.: Integrating netlogo and jason: A disaster-rescue simulation. In: 2017 9th Computer Science and Electronic Engineering (CEEC). pp. 213–218 (2017). https://doi.org/10.1109/CEEC.2017.8101627
11. SpigotMC: CommandHelper (2025), https://www.spigotmc.org/resources/commandhelper.64681/