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 Minecraft-based multi-agent testbed for developing and evaluating belief-desire-intention (BDI) agents programmed in Jason. Argus provides the infrastructure required to integrate Jason agents with Minecraft, modeling autonomous non-player characters (NPCs) to perceive, reason about, and act within a dynamic game environment. The testbed supports the implementation of diverse agent strategies as well as the collection of performance metrics for studying agent behaviour in collaborative and competitive multi-agent settings. To demonstrate the capabilities of the platform, we present a survival scenario in which BDI agents must cooperate to gather resources, craft tools, defend against zombies, and maximize their score. We further implement two example agent strategies for this environment and present a sample experiment to illustrate the kinds of data and behaviours Argus enables researchers to study.

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 [9]. 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 [8]. AgentSpeak [7] is a language for describing BDI systems with Jason [3] as one of its notable implementations.

While Jason has been successfully applied across a variety of domains, there remains a need for rich, interactive environments that support the study of multi-agent behaviours such as cooperation, coordination, and competition. To address this need, we introduce Argus, an extensible Minecraft-based testbed for developing and evaluating Jason agents. Argus provides the underlying infrastructure required to connect Jason agents to Minecraft, enabling perception, communication, and action within a dynamic multi-agent environment. The primary contributions of this work are the integration framework that bridges Jason and Minecraft and the Argus environment itself, which supports the development and evaluation of diverse agent strategies. To demonstrate the capabilities of

the platform, we additionally implement two example BDI agent strategies and present a brief analysis of their behaviour within the environment.

1.2 Problem Description

Minecraft is a popular game developed by Mojang Studios. In this game, players explore different worlds and fight enemies [11]. Within Argus, we developed a custom Minecraft scenario consisting of a forested environment populated by zombies and autonomous agents. The primary objective of each agent is to survive zombie attacks while maximizing its score through resource gathering, crafting, combat, and cooperation.

In this environment, agents may engage in a variety of actions, including attacking zombies or competing agents, chopping down trees for wood, building houses for shelter, crafting weapons, and forming alliances to share resources and coordinate behaviour. This setting provides opportunities to study a range of multi-agent phenomena, including cooperation, competition, resource management, and strategic decision-making in a dynamic environment [1].

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) [8]. AgentSpeak is a Prolog-like language that is used for specifying these plans [7]. Jason [3] is a popular interpreter for AgentSpeak implemented in Java.

Jason represents beliefs as predicates. It represents desires in the form of achievement and test goals. Achievement goals represent the desire to satisfy a specific predicate. Similarly, test goals represent the desire to verify whether a specific predicate holds. Jason also adds a notion of triggers, which represent the addition or removal of specific beliefs, achievement goals, or test goals [3].

BDI allows the agents to react to live stimuli and choose the correct plan in order to achieve their overall goals. This makes it suitable for fast-paced and non-deterministic environments like games. Additionally, since both Minecraft and Jason are Java-based, connecting them would be quite straightforward.

2.2 Paper, Citizens & CommandHelper

Various tools exist for creating custom Minecraft modifications. The main tool used for this project was `Paper` [12], which is a Minecraft game server with extensive API support for custom modifications to the game. Paper also supports

[1] Demos of Argus are available online: `https://youtube.com/playlist?list=PL5eOOo-XDNOE4PsrYV26QpLOeGOk5jG7&si=opX6XVHEN-4KCHtQ`

a library of plugins to extend the server's capabilities. A notable plugin used for this project was the `Citizens` plugin [6], which helped with the creation and maintenance of non-player characters (NPCs). Another plugin used was `CommandHelper` [14], which helped automate various aspects of the game.

2.3 Related Work

There have been many efforts to apply Jason and BDI to different domains. [4] 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. The environment used in [4] consists of complex, multi-step tasks that inspired some of the tasks in Argus, such as building different houses and artifacts; however, the city environment does not support direct agent-to-agent communication and instead relies on shared perceptions. Another example is [13], which applies 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 to simulate inter-agent communication in Argus for resource sharing.

The game Minecraft has been a popular environment for multi-agent research, especially with the rise of LLMs. [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. [5] is another attempt at connecting LLMs to Minecraft. The authors argue that multi-agent solutions can be a lot more effective for a game like Minecraft due to its complex nature. The varying coalition size in Argus allows the user to explore this idea and assess the importance of coalitions. Additionally, [5] proposes causality-based decomposition of the complex tasks in Minecraft. This idea directly transfers to BDI for breaking complex goals into sets of small desires. We used a similar idea for Argus when generating the plans for crafting items.

Unlike these systems, Argus specifically targets the integration of Jason-based BDI agents with Minecraft, providing a reusable platform for experimentation with cooperative and competitive multi-agent behaviour.

3 Methodology

To demonstrate Argus, we developed a survival scenario in which agents must survive zombie attacks while maximizing their score. In this scenario, the agents 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. To make the environment more realistic, each agent was presented with a limited view of its surroundings (a parameter of 30 blocks), requiring them to constantly move around and explore the environment. 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. These messages were sent in the form of beliefs. The agents could use this feature to let their allies know of their houses or possible nearby enemies. 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. The attacker and capitalist agents are example strategies included to demonstrate how Jason agents can be implemented within Argus. More strategies can be implemented for this environment via new ASL files.

3.2 Connecting Jason to Minecraft

The core contribution of Argus is the underlying architecture that allows communication between Jason and Minecraft. The most important aspect of this architecture 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

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

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. Using CommandHelper, we were able to automate the environment setup and the process of spawning the agents with a custom script. This script allows users to customize the environment as much as possible. It is also important to note that this architecture can be run headless, without the need to launch the game itself, allowing for repeated experimentation and data collection. Figure 1 shows an overview of this proposed architecture.

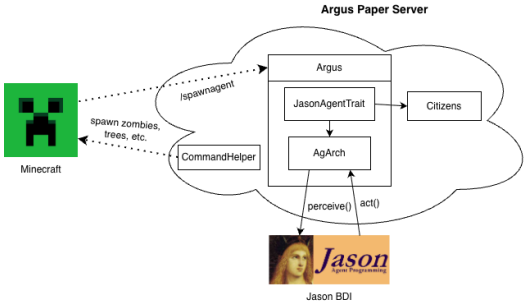

Fig. 1. An overview of the overall system architecture.

Additionally, it was important to ensure proper time synchronization between Jason and Minecraft. By default, Minecraft has its own clock that performs 20 ticks per second (one tick every $0.05s$). Minecraft uses these ticks as its update unit. We allocated two ticks per Jason's reasoning cycle, which equates to an update every $0.1s$. We noticed that Jason could get overwhelmed with higher update frequencies due to the number of plans and agents. Additionally, the $0.1s$ refresh rate is supported by the findings of [10], which demonstrates that Jason's reasoning remains under $0.1s$ for up to 10000 beliefs and plans.

4 Example Use: Survival Scenario Analysis

The proposed survival environment allows for the collection and analysis of various metrics, such as each agent's final score, number of wood pieces chopped or donated, 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 2 shows some of the metrics that can be collected in this environment. Looking closer at these charts, we can see how coalitions can change the behaviour of individual strategies. For example, in this initial experiment, we notice that with

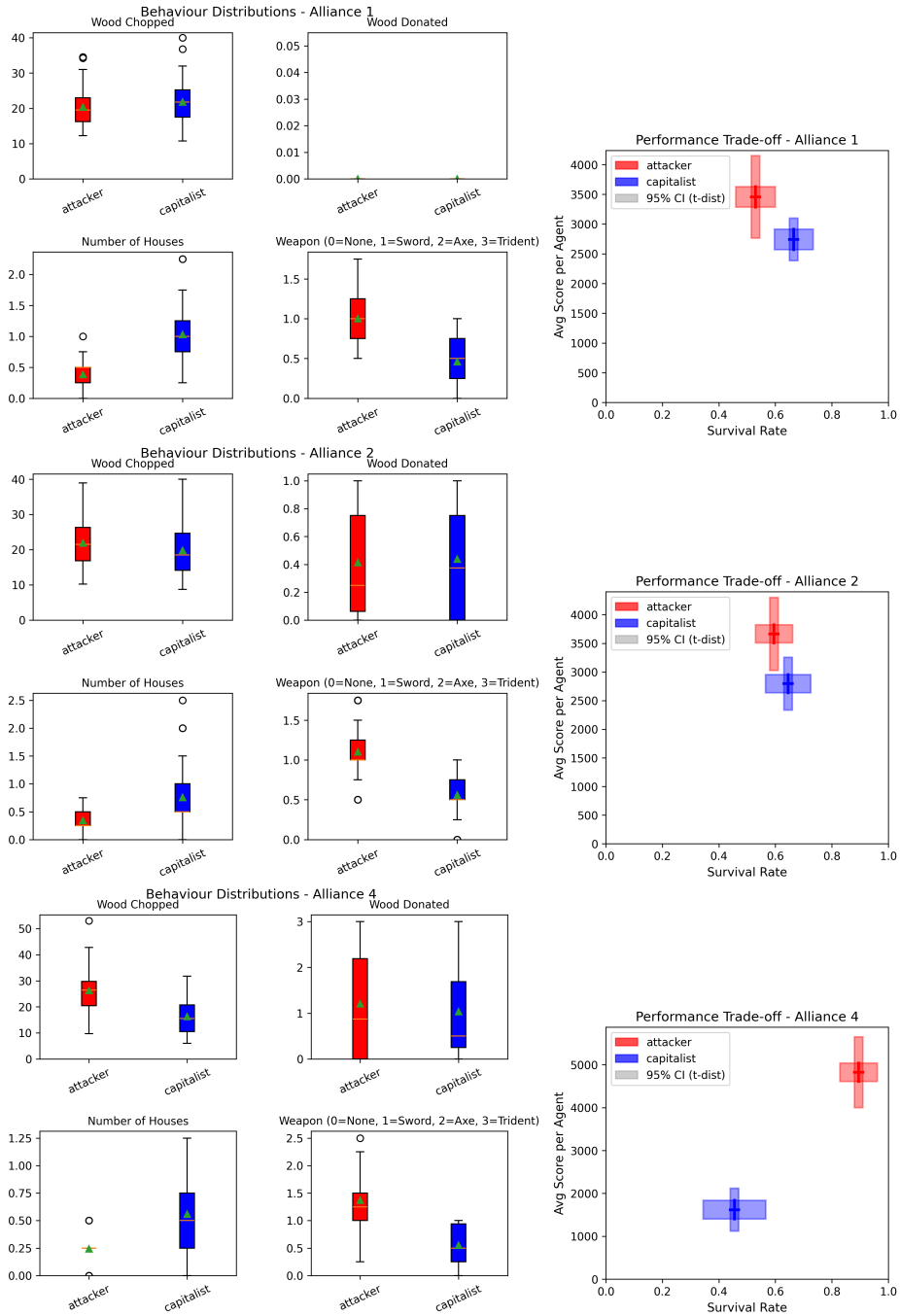

Fig. 2. Analyzing the attacker and capitalist strategies with different coalition sizes.

the introduction of coalitions, the attacker agents are spending less resources on individual survival (i.e., building houses) and instead rely more on sharing wood and houses. This allowed them to perform more weapon upgrades and get a much better performance. This observation is an example of priority inversion in the hard-coded plans for attackers, since they are all tasked with building a house first; however, some are bypassing that requirement since they are now being offered a house by their peers. This experiment is just an example of the type of analysis that could be conducted when complex multi-agent interactions are introduced to hard-coded BDI agents, but due to the unpredictable nature of Minecraft, more detailed experimentation is needed to draw any conclusions about the performance of any single strategy. Similarly, while the current experiment uses eight agents, the architecture is not restricted to this scale, and future work can investigate larger populations.

5 Conclusion

In this paper, we described a multi-agent Minecraft-based testbed for programming BDI agents, named Argus. We also defined a simple survival environment to showcase the capabilities of Argus. In this environment, the agents, programmed in Jason, behaved as NPCs and followed the BDI paradigm. They worked together to defeat zombies and collect points. Through a sample experiment, we demonstrated the range of behavioural and performance metrics that Argus makes available to MAS researchers.

5.1 Future Work

Future work could look into various directions. One direction could be developing more complex strategies for the existing sample environment to further investigate how an agent's individual plans can be prioritized to serve both itself and its coalition. These future strategies could look into more complex coalition behaviour, such as the possible addition of betrayals. Another direction could be extending Argus to support different agent paradigms, specifically some of the existing notable LLM Minecraft agent architectures (i.e., Theory of Mind), and comparing their behaviour with hard-coded BDI agents. Furthermore, due to the richness of Minecraft, this environment is extensible to new actions and mechanisms. A direct update to this system could be developing a more complex sample environment and further extending the architecture to generate beliefs directly from images, increasing the scope to potentially play the full Minecraft game, making it very similar to its LLM counterparts. Additionally, the current architecture supports the addition of human players as well, so this environment can be used to study human-agent interactions. Finally, the architecture proposed in this paper is quite generic and can be directly applied to any Java agent framework, such as Jade [2].

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. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: Jade — A Java Agent Development Framework, pp. 125–147. Springer US, Boston, MA (2005). https://doi.org/10.1007/0-387-26350-0_5
3. 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
4. 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
5. Chai, Q., Zheng, Z., Ren, J., Ye, D., Lin, Z., Wang, H.: CausalMACE: Causality empowered multi-agents in Minecraft cooperative tasks. In: Christodoulopoulos, C., Chakraborty, T., Rose, C., Peng, V. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2025. pp. 14410–14426. Association for Computational Linguistics, Suzhou, China (nov 2025). https://doi.org/10.18653/v1/2025.findings-emnlp.777
6. Citizens: Citizens API (2025), https://wiki.citizensnpcs.co/API
7. D'Inverno, M., Luck, M.: Engineering AgentSpeak(L): A Formal Computational Model. Journal of Logic and Computation **8**(3), 233–260 (1998)
8. 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
9. 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
10. Miller, J., Esfandiari, B.: Analysis of the Execution Time of the Jason BDI Reasoning Cycle. In: Alechina, N., Baldoni, M., Logan, B. (eds.) Engineering Multi-Agent Systems. pp. 218–236. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-97457-2_13
11. Minecraft: What is Minecraft? (2025), https://www.minecraft.net/en-us/about-minecraft
12. PaperMC: PaperMC (2025), https://papermc.io/
13. 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
14. SpigotMC: CommandHelper (2025), https://www.spigotmc.org/resources/commandhelper.64681/