Agent of Rivia

Shaikha Alhammadi
shaikha.alhammadi@studio.unibo.it

Igor Chukarin
igor.chukarin@studio.unibo.it

July 4, 2026

This project is a fantasy creature hunting simulation inspired by The Witcher. The environment consists of a grid-based world where a Hero agent hunts and defeats enemy agents. Battles are triggered when the Hero encounters enemies in neighboring cells, and each victory allows the Hero to improve both health and strength. The simulation includes a health recovery area and tracks the number of Hero deaths. The system is implemented in Java, with agent behaviors programmed in Jason and a graphical interface developed using Java.

Contents

1 Introduction

Intelligent agents are autonomous software entities that perceive their environment and perform actions in order to achieve specific goals. Unlike traditional programs that follow a fixed sequence of instructions, intelligent agents are capable of making decisions based on their current knowledge and objectives. In Multi-Agent Systems (MAS), multiple agents coexist within a shared environment, where they can interact, cooperate, or compete to achieve their goals.

The Jason platform provides a framework for developing intelligent agents using the Belief-Desire-Intention (BDI) model. In this model, an agent's behavior is determined by its beliefs about the environment, its desires or goals, and the plans it uses to achieve those goals. This approach allows agents to exhibit autonomous and goal-directed behavior while adapting to changes in their environment.

The purpose of this project is to demonstrate the concepts of intelligent agents through a fantasy creature hunting simulation inspired by games such as Elden Ring and The Witcher. The simulation consists of a Hero agent operating in a grid-based environment populated by enemy agents. The Hero must locate and defeat all enemies while managing its health and improving its attributes over time. Through this scenario, the project illustrates how intelligent agents can perceive their surroundings, make decisions, pursue goals, and interact within a dynamic environment.

Studying intelligent agents through practical simulations helps in understanding the principles behind autonomous systems that are widely used in fields such as robotics, industrial automation, logistics, and decision-support systems. By implementing the agents in Jason, this project provides hands-on experience with agent-oriented programming and the development of autonomous behaviors.

2 Background and Link to the Theory

To support the design and the implementation of our project, a review of relevant literature was conducted on Multi-Agent Systems (MAS), intelligent agents in game environments, and the Belief-Desire-Intention (BDI) architecture. The selected papers provided both the theoretical foundation and practical design principles that guided the development.

The following sections summarize the key contributions of each paper and discuss their relevance to the architecture and the implementation of the proposed system.

2.1 Literature Review of Papers

2.1.1 Belief-Desire-Intention (BDI) Architecture

This paper [1] explains the theory behind beliefs, desires, intentions and the Belief-Desire-Intention (BDI) reasoning cycle. The paper presents BDI architecture as a practical framework for developing rational agents capable of operating in complex and dynamic environments. The BDI model represents an agent through three mental components: **beliefs**, which describe the agent's knowledge of the environment (and itself), **desires**, which represent the goals the agent aims to achieve and **intentions**, which are the plans the agent executes to achieve the goals. By separating these concepts, the architecture enables agents to balance goal-directed behavior with the ability to react to changes in their environment. The BDI model enables an agent to distinguish between all possible objectives and those that are currently achievable, resulting in behavior that is both goal-oriented and adaptive.

A significant contribution of the paper is its discussion of decision-making within dynamic environments. Rao and Georgeff examine the characteristics of real-time applications domains such as air traffic management. The authors explain that autonomous agents often operate in situations where the environment changes continuously. They identify key characteristics such as nondeterminism (both in the environment and the system), competing objectives and local sensing. Under these circumstances, determining a single optimal solution is rarely feasible, as new information may invalidate previous decisions before they are fully executed.

A central theme is the "Dilemma of Commitment", agents must balance between being goal-directed (sticking to a plan) with being reactive (changing plans when the environment shifts). If an agent, continuously, re-evaluates every possible action whenever new information becomes available, the computational cost becomes excessive and valuable execution time is lost. Yet, blindly committing to an existing plan without considering the changes within the environment can result in poor decisions when the assumptions on which the plan was based are no longer valid. The BDI architecture addresses this problem by allowing agents to maintain commitments while periodically reassessing their beliefs and determining whether their current intentions remain appropriate. This balance enables agents to pursue long-term objectives without sacrificing their ability to respond effectively to unforeseen events.

The paper further strengthens its theoretical foundations by extending classical decision theory into a logical framework based on possible worlds. Within this framework, the relationships between beliefs, desires and intentions are defined through a series of **static** and **dynamic** constraints. Static constraints describe the logical consistency and relationship between an agent's beliefs, desires and intentions (e.g., "Realism where an agent only desires what it believes is possible). On the other hand, dynamic constraints govern how commitments are maintained or abandoned as the environment changes. The authors distinguish between different commitment strategies, including **blindly committed** (denies any changes to beliefs or desires that conflict with current commitments), **single-minded** (accepts changes to beliefs and will drop commitments accordingly) and **open-minded** (allows changes in both beliefs and desires that force commitments to be dropped) agents. Each differing in the conditions under which existing intentions are reconsidered or discarded. These concepts provide a formal explanation of how rational agents can maintain coherent behavior while remaining sufficiently flexible to adapt to changing circumstances.

Although the BDI architecture has strong foundations, the authors emphasize that practical implementations cannot rely on complex logical reasoning alone. This is due to the computational demands of real-time systems. They introduce a concept called **BDI Interpreter Loop**, which is an iterative execution cycle that enables efficient agent reasoning. During each iteration, the agent processes new events from the environment, generates possible options, deliberates on the most appropriate course of action, updates its intentions, executes the selected actions and finally revises its internal state before repeating the process. This cycle provides a mechanism for balancing responsiveness and computational efficiency while preserving the principles of the BDI model. Figure 1 illustrates how BDI Interpreter works.

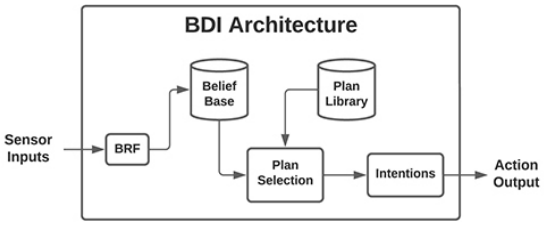

Figure 1: BDI Interpreter Loop

To demonstrate the practicality of the BDI architecture, the authors discuss implementations such as the Procedural Reasoning System (PRS) and dMARS, which replace computationally expensive logical reasoning with predefined plans and intention stacks. This approach improves efficiency while maintaining the modularity and flexibility of the BDI model. The architecture is further validated through the OASIS air traffic management system, where autonomous agents use beliefs, desires, and intentions to coordinate aircraft in a dynamic environment. This real-world application demonstrates that the BDI approach can support reliable, adaptive decision-making in complex, time-critical domains.

2.1.2 AI and Multi-Agent Systems: Collaboration and Competition

This paper [2] discusses how Multi-Agent Systems (MAS) have emerged as a significant advancement in the field of Artificial Intelligence by extending traditional single-agent approaches into environments where multiple autonomous agents interact simultaneously. The authors describe MAS as systems composed of independent, self-governing agents that operate within a shared environment while pursuing individual or collective objectives. Unlike conventional AI systems, where a single entity is responsible for all decision-making, MAS distributes intelligence across multiple agents whose actions influence one another. This distributed approach enables the system to solve problems that are too complex for a single agent while improving adaptability, scalability and robustness in dynamic environments.

The paper identifies a fundamental characteristic of MAS which is the interaction between agents. The authors provide two modes: **collaboration** and **competition**. In collaborative environments, agents exchange information, coordinate their actions and cooperate to achieve a common objective. An example provided was autonomous vehicle networks, where vehicles continuously communicate road conditions and traffic information to improve safety and optimize traffic flow. In contrast, competitive environments involve agents pursuing individual objectives that may conflict with those of other agents. This behavior was illustrated with financial trading systems, where autonomous agents analyze market conditions and adapt their strategies to outperform competing traders. These contrasting interaction models demonstrate how versatile MAS is and explain why they can be applied to a wide variety of domains requiring either cooperation, competition or a combination of both.

Although Multi-Agent Systems provide significant advantages over traditional AI systems, their development introduces several complex challenges. Since multiple autonomous agents operate simultaneously, effective communication and coordination become essential to ensure that decisions remain consistent despite incomplete or conflicting information. Additionally, the behavior of one agent influences the behavior of others continuously, which creates a dynamic learning environment where optimal strategies must constantly evolve. Therefore, maintaining overall system stability becomes increasingly difficult as interacting agents grow. This is a particular issue in safety-critical applications where poor coordination may lead to undesirable or unpredictable outcomes. This paper identifies these challenges as the main obstacles that must be addressed before MAS can be deployed successfully in real-world environments.

The authors discuss several computational techniques that enhance agent decision-making and coordination. **Deep Reinforcement Learning** (DRL) enables agents to learn effective strategies through interaction with the environment by receiving rewards and penalties based on their actions. This allows agents to gradually improve their behavior while also adapting to dynamic conditions. **Game Theory** provides a mathematical framework for modeling competitive interactions, enabling agents to anticipate the behavior of rivals and make more informed strategic decisions. In a collaborative system, **Swarm Intelligence** offers decentralized coordination mechanisms through which groups of relatively simple agents collectively solve complex problems without relying

on centralized control. Together, these approaches improve the adaptability, learning capability and scalability of modern Multi-Agent Systems.

The paper identifies as well two promising research directions for MAS. The first is the integration of Large Language Models (LLMs) into Multi-Agent Systems, enabling agents to communicate using natural language and improving collaboration between humans and intelligent systems in domains such as healthcare, education and customer support. The second is in Edge Computing, which allows autonomous agents to operate efficiently within distributed environments that have limited computational resources. Such advancements are expected to extend the capabilities of MAS into multiple different domains such as disaster response, environmental monitoring, and precision agriculture.

In conclusion, MAS has fundamentally changed how we approach complex, evolving challenges. By leveraging Deep Reinforcement Learning and Game Theory, these systems provide the necessary structure for agents to thrive in both collaborative and competitive settings. While coordination and competitive volatility remain technical challenges, the ongoing integration of advanced AI technologies like LLMs positions MAS as a leading force in the future of intelligent, real-time decision-making.

2.1.3 Games and Agents: Designing Intelligent Gameplay

This paper [3] explores the integration of Multi-Agent Systems (MAS) within modern game engines, addressing the growing demand for more intelligent and believable Non-Player Characters (NPCs). While there are significant advancements in computer graphics with improved realism of modern games, the cognitive behavior of virtual characters remains limited, relying on predefined scripts rather than autonomous reasoning. The authors argue that achieving truly intelligent gameplay requires moving beyond scripted behaviors and adopting autonomous software agents capable of long-term planning, reasoning and interaction. Their work therefore focuses on bridging the gap between game development and agent-oriented Artificial Intelligence by proposing methods that allow intelligent agents to operate effectively within game environments.

A key contribution of this paper is its discussion of what composes a software agent. The authors highlight that the term agent is often interpreted differently within gaming and Artificial Intelligence research community. In many games, a NPC is considered an agent, even if its behavior is entirely scripted. From the perspective of AI, an agent is expected to be proactive, autonomous, reactive and social. This means it possesses its own goals, can make decisions independently, responds to changes in the environment and communicates with other agents. Among the various agent models discussed, the Belief-Desire-Intention (BDI) paradigm is identified as the most suitable for developing believable game characters because it enables explicit goal management, planning and adaptive decision-making over extended periods of time.

The paper identifies a fundamental challenge when integrating Multi-Agent Systems into game engines. While game engines are primarily designed to maximize rendering performance, maintain strict timing constraints, and operate through a centralized game loop, agent platforms prioritize autonomous reasoning, distributed control, and asynchronous decision-making. These conflicting design philosophies create several in-

tegration challenges. The first concern is synchronization, where the asynchronous reasoning cycle of intelligent agents must be coordinated with the synchronous execution of the game engine. The second involves information representation, requiring low-level engine data such as positions, collisions and geometric information to be translated into meaningful concepts that agents can understand. Finally, communication is another issue in which interactions between agents should produce observable effects within the game world rather than existing only as internal message exchanges.

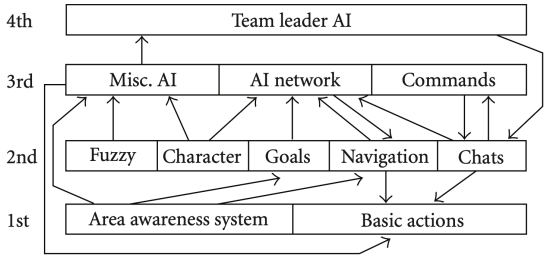

Figure 2: The Quake III bot architecture

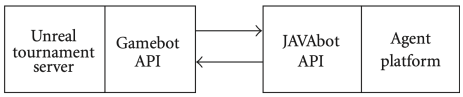

Figure 3: Proposed architecture for integrating a game engine with an external agent platform

To face these challenges, the paper reviews two dominant approaches used to integrate intelligent agents into games.

First, the server-side approach, which is adopted by games such as Quake III and F.E.A.R., performs agent decision-making directly within the game loop. This approach offers high execution speed but it also limits the complexity of reasoning because every decision must be completed within a single simulation step. Figure 2 illustrates the layered architecture adopted. At the lowest level, the Area Awareness System (AAS) translates raw engine information into a representation suitable for agent reasoning by providing navigation data, routing information and knowledge about objects within the environment. Higher layers are responsible for behaviors such as goal management, navigation, fuzzy logic, communication and team coordination. Although this layered organization enables efficient real-time execution, the close coupling between the reasoning process and the game engine restricts the complexity of agent behavior and makes the agents overly dependent on the information provided by AAS.

The second approach is the client-side approach. This is used by systems such as Gamebots and Quakebots. The approach executes agents as separate applications that communicate with the game through network messages. This architecture allows significantly more sophisticated reasoning but suffers from restricted access to environmental

information, which in return provides a weaker synchronization between the reasoning process and the game engine. The architecture in Figure 3 illustrates how to integrate a game engine with an agent platform. Rather than embedding the reasoning process directly into the game engine, the architecture introduces a communication layer between the simulation environment and the agent platform. This separation allows the game engine to manage world simulation while autonomous reasoning is delegated to an external agent platform. The information exchange is done through dedicated APIs.

Another important observation made by the authors is the widespread reliance on scripted behaviors in modern games. Although scripting is computationally efficient, it often produces unrealistic and inflexible NPC behavior. This limits the ability for the agent to adapt to changing situations or communicate effectively.

To address this limitation, the authors propose a holistic framework for integrating intelligent agents into the earliest stages of development. There are three complementary perspectives for doing so. First, an infrastructural stance, which supports asynchronous communication between the game engine and agents. Second, a conceptual stance in which low-level engine data is translated into high-level concepts suitable for reasoning. Lastly, design stance, which advocates incorporating agent-oriented methodologies during early stages of development. Together, these perspectives promote, modular, autonomous and scalable agent behavior while improving communication and interaction within the game environment.

The paper concludes that intelligent gameplay requires more than just isolated technical improvements. Instead, game environments should be designed to support asynchronous communication, meaningful knowledge representation and autonomous agent interaction from the earliest stages of development.

2.1.4 Spyke3D: a new Computer Games oriented BDI Agent Framework

This paper [4] presents Spyke3D, a framework designed to support the development of intelligent Non-Player Characters (NPCs) through the integration of Multi-Agent Systems (MAS) and the Belief-Desire-Intention (BDI) paradigm. The paper reviews the existing approaches used to integrate intelligent agents into game environments and identifies several limitations. The authors distinguish between server-side and client-side architectures, which both are mentioned in this paper [3].

To address the shortcomings of both the client-side and server-side approaches, the authors introduce Spyke3D, a framework that combines MAS technologies with BDI paradigm to simplify the development of intelligent NPCs. The framework integrates three complementary technologies. **SPADE**, which provides the multi-agent communication platform, **PyKE**, a knowledge-based inference engine responsible for logical reasoning and **Panda3D**, which manages the graphical simulation environment. These components allow developers to implement sophisticated agent behavior while hiding much of the underlying complexity associated with communication, reasoning and knowledge management.

One of the framework's most significant contributions is the separation of each NPC into two independent components: the Brain and the Body. The brain is responsible for

high-level reasoning maintaining the agent's beliefs, selecting goals, and generating plans using BDI paradigm. The Body represents the physical entity within the game world, handling sensing, movement and interaction with the environment. Communication between these two components resembles a nervous system, where sensory information flows from the Body to the Brain, while the brain returns intentions that are translated to executable actions within the game engine. This separation provides several advantages, such as improved modularity, reduced coupling between the AI and the game engine, and the ability to reuse the reasoning system across different environments without modification.

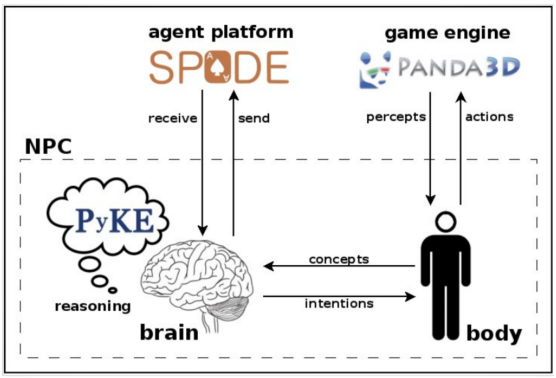

Figure 4: Separation of NPC into: the Brain and the Body

As illustrated in Figure 4, the Body receives percepts from the game engine and forwards them to the Brain, where reasoning is performed using the PyKE inference engine and the SPADE multi-agent platform. After deliberation, the Brain generates intentions that are translated into executable actions by the Body, creating a clear separation between cognitive reasoning and physical interaction with the environment.

The paper extends the traditional BDI model by introducing concepts such as Agent DNA, which influences goal selection and planning behavior, which allows agents with identical knowledge bases to exhibit different personalities and decision-making strategies. The reasoning process is performed using backward chaining, which works from the desired goal towards the actions required to accomplish. This means that plan deliberation is performed after selecting a goal. The agent's current beliefs are evaluated to determine the most appropriate plan for achieving a goal. Knowledge partitioning is also employed through reusable stereotypes, enabling groups of agents to share a common domain knowledge while avoiding unnecessary code duplication. Together, these mechanisms produce more flexible, believable and maintainable intelligent agents.

To evaluate the framework, the authors implemented a resource collection scenario using Explorer and Gatherer agents. The results showed that Spyke3D effectively supports autonomous BDI reasoning by separating perception, decision-making and action execution, all while simplifying the implementation of communication and reasoning mechanisms.

The paper concludes that the BDI paradigm provides an effective foundation for de-

veloping realistic, believable and autonomous NPCs by separating reasoning from the game engine and supporting intelligent decision-making. The authors have also identified several directions for future research, including improving decision-making under uncertainty, enhancing deliberation algorithms, developing tools to simplify agent development and extending the framework to support additional game engines, further improving its flexibility and reusability.

2.2 Link to Project

Collectively, these papers provide the architectural foundation that was adopted in this project. The BDI architecture establishes how autonomous agents reason using beliefs, desires and intentions. The Multi-Agent Systems extend this reasoning to environments containing multiple interacting agents. Within games, this combination enables Non-Player Characters (NPCs) to move beyond scripted behavior by making independent decisions, reacting to environmental changes and interacting with other agents. Rather than simply embedding all decision making in the game engine, modern game-oriented agent architectures advocate separating the game simulation from the reasoning process. The separation improves modularity, simplifies maintenance and allows the agent intelligence to be reused independently of the game implementation.

This design philosophy is directly reflected in the proposed architecture of this project. The game environment is responsible only for maintaining the world state, executing actions and providing/receiving percepts. The agents perform BDI reasoning and decision-making. The environment acts as the communication layer between Jason and the model. This helps with translating low-level game events into percepts and executing actions returned by the agents. This creates a clear separation between game logic and agent logic, allowing each component to evolve independently while preserving a modular system architecture.

Finally, the Spyke3D framework reinforces this architectural decision through its Brain-Body model, where each NPC is divided into a cognitive component responsible for reasoning and a physical component responsible for interacting with the environment. This principle was employed in this project. The Jason agent represents the "Brain", in which beliefs, desires and intentions are generated through the BDI reasoning cycle, whereas the environment functions as the "Body", which manages the communication with the game world. This separation closely mirrors the architecture proposed in the literature. It allows for autonomous reasoning to remain independent from the underlying game engine, resulting in a more reusable, maintainable and scalable design.

3 Analysis

3.1 Requirements

In this section, we present the functional and non-functional requirements of the system.

3.1.1 Functional Requirements

- **Whole System**
  - The system shall initialize a fixed-size grid environment.
  - The system shall create one Hero agent at the Home position.
  - The system shall generate multiple Monster agents at random positions away from the Home and Tavern locations.
  - The system shall assign each Monster agent a type, health, and strength.
- **Agent Movement and Perception**
  - The Hero agent shall explore the environment by wandering from its starting position.
  - The Hero agent shall perceive nearby monsters through percepts provided by the environment.
  - The Hero agent shall move in the four cardinal directions (forward, backward, left, and right).
  - The environment shall validate all movements to ensure agents remain within the arena boundaries.
- **Combat System**
  - The system shall detect interactions between adjacent agents.
  - The Hero agent shall attack Monster agents when they are adjacent.
  - The system shall update the Monster agent's health after an attack.
  - The system shall update the Hero agent's health after combat.
  - The system shall remove defeated Monster agents from the environment.
  - The system shall display combat interactions in the console.
- **Progression and Respawning**
  - The Hero agent shall respawn at the Home position with full health after being defeated.
  - The Hero agent's health and strength shall increase after defeating a Monster agent.
  - The Hero agent's level shall increase when its attributes improve.
  - The environment shall increment the Hero agent's death counter each time the Hero is defeated.

3.1.2 Non-functional Requirements

- Performance → The environment should process agent actions in real time and GUI updates should occur without noticeable delay.
- Reliability → The simulation should continue running without crashing.
- Maintainability → The project should separate the environment, model, view and utility into independent packages. New monster types should be easy to add without modifying unrelated components.
- Modularity → Jason agent logic should remain independent from Java implementation. The environment should communicate with agents only through percepts and actions.
- Scalability → The system should support adding additional monster agents with minimal code changes.
- Correctness → Agents should never move outside the arena boundaries and only living agents should be able to perform actions. Health values should always be updated correctly after combat.

3.2 Top-down analysis

The objective of the project is to develop a Multi-agent System (MAS) in which a Hero agent and multiple Monster agents interact within a shared two-dimensional environment.

3.2.1 Overall Problem

Develop a Multi-Agent System capable of simulating autonomous agents that perceive their environment, make decisions, interact with one another and display the state of the simulation to the user via a console.

3.2.2 High-Level Decomposition

The overall problem can be divided into four major sub-problems:

- **Environment Management**: Responsible for initializing the arena, managing the simulation, executing agent actions and providing percepts to the agents
- **Agent Behavior**: Responsible for processing percepts, determining the current behavioral state, selecting appropriate actions
- **State Management**: Responsible for maintaining the current state of the simulation including agent positions, orientations, health, combat and agent status
- **Visualization**: Responsible for presenting the current state of the arena and updating the graphical interface after each simulation step

Each sub-level can be further refined into smaller responsibilities according to the flow of the system. The Environment Management component acts as the communication layer between the BDI agents and the simulation. The environment receives actions generated by the agents and forwards them to the model, where the simulation state is updated. The updated state is then passed to the visualization component which refreshes the graphical interface. The environment also collects information from the model, converts it into percepts and sends them back to the agents. This helps with completing the perception-action cycle.

4 Design

4.1 Structure

The project consists of a Witcher agent, monster agents and an arena in which these agents interact. The arena is the central abstraction, it represents the shared world where movement, combat and other events take place.

The arena is divided into three main responsibilities: the environment, the model, and the view. The model represents the state of the world, the environment mediates the interaction between agents and the world, and the view displays the state of the world to the user.

- The model is responsible for storing the state of the arena. It contains information about the arena size, the positions of agents, health and strength of each monster, and the Witcher death counter. The model provides operations that allow the state to be changed in a controlled way, such as moving an agent, applying damage to a monster, or changing monster status to "dead" or "alive". This design was chosen because the simulation requires a single source of truth. If each agent stored and modified its own version of the world, the system could easily become inconsistent.

- The environment acts as an intermediate layer between the agents and the model. Agents do not directly change the arena state. Instead, they perform actions, and the environment interprets these actions and applies them to the model. The environment also provides percepts to the agents, allowing them to observe the world and make decisions. This design was chosen because it keeps the agents focused on decision-making, while the validation of actions is moved to the environment and the update of the world state is performed through the model.

- The view is responsible for presenting the current state of the arena to the user. It observes the model and updates the graphical representation when the model changes. As a result, different means of visualization can be chosen for implementation.

Arena structure is illustrated in Figure 5. Most of the operations are assigned to the model, since it is responsible for maintaining and updating the state of the arena. This decision was made to keep the domain logic centralized and to avoid spreading state-changing behavior across multiple components.

The use of interfaces for the model and the view makes the dependencies between components more explicit. The environment and the graphical representation rely on abstractions rather than concrete implementations, which makes the system easier to modify or extend in the future. The environment acts as the connection point between the agents and the arena. It interacts with the agents through messages: percepts are sent to the agents to describe the current state of the world, while actions are received from the agents and interpreted as requests to change the arena state.

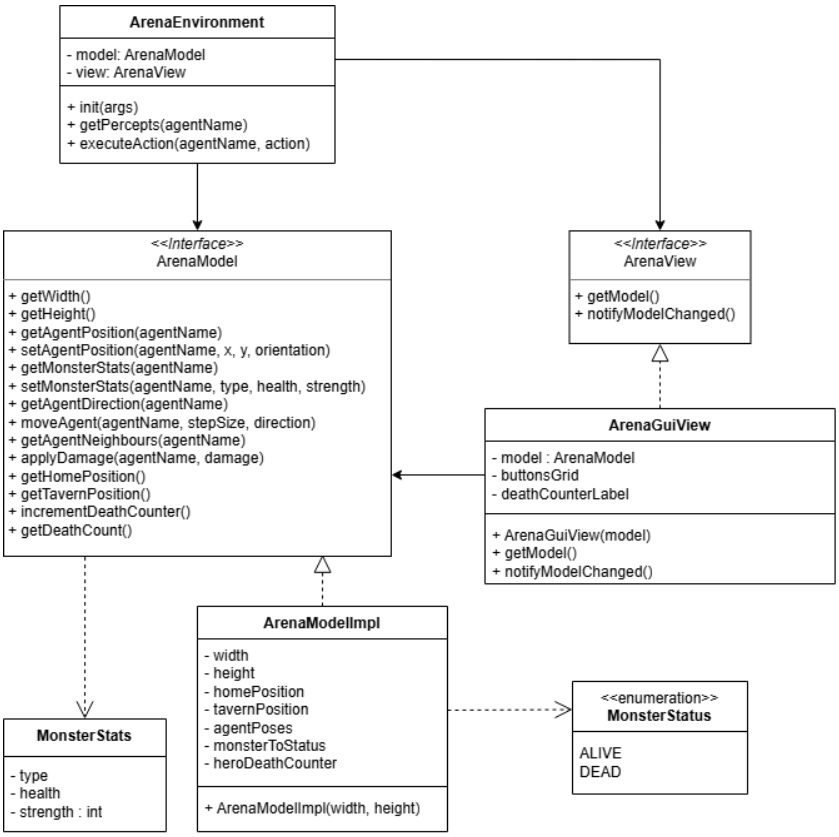

Figure 5: Arena Class Diagram

4.2 Perception-Action Flow

The interaction is based on a perception-action cycle. The agent does not directly control the state of the arena. It receives observations about the current world state, reasons about them, and performs actions. The arena validates these actions and updates the world state accordingly.

Figure 6 shows the activity flow of the Witcher agent. At the beginning of the cycle, the Witcher checks whether there are any alive monsters in the arena. If no alive monsters remain, the hunting process is completed: the Witcher goes to the tavern to celebrate and then returns home. If at least one monster is still alive, the Witcher first evaluates if he has enough health to start hunting.

If the Witcher's health is below the chosen threshold, he goes to the tavern and restores his health before continuing the hunt. A higher threshold makes the agent more cautious, since it starts prioritizing healing earlier, while a lower threshold makes it more risk-oriented, allowing it to continue fighting monsters even with low health. Thus, this parameter works as a way to tune the agent between defensive and aggressive behavior.

If the Witcher is healthy enough, he selects a target. After selecting a target, the Witcher checks whether the monster can be hunted. This check is necessary because a monster may be alive but too strong. If the target cannot be hunted, the Witcher returns to target selection.

Once the target is encountered, the Witcher evaluates whether he is strong enough to fight. If he is not, he retreats and returns to the target selection process. This design makes the agent more rational: fighting depends on the estimated risk of the encounter. A simpler alternative would be to pick every fight, but this was rejected because it would ignore the role of monster statistics in decision-making.

If the Witcher is strong enough, combat begins. During combat, damage is applied and the state of both participants is updated. After each fight step, the system checks whether the monster has been defeated or whether the Witcher's health has fallen below zero. If the monster is defeated, the Witcher levels up and continues hunting the remaining monsters. If the Witcher dies, the death counter is increased and the Witcher respawns before returning to the hunting cycle.

The loop continues until all monsters have been defeated. This interaction model reflects the behavior expected from an autonomous agent: the Witcher does not follow a fixed script from beginning to end, but reacts to the current state of the world and to the parameters of other agents.

4.3 Behavior

The Witcher agent has a set of states connected by condition-based transitions. This choice was made to model the Witcher as an autonomous entity whose behavior depends on both the current world situation and his own internal condition. Figure 7 presents the state diagram of the Witcher agent.

The *Idle* state represents the neutral decision point of the Witcher. From this state, the Witcher evaluates the situation and chooses the next mode of behavior. From this

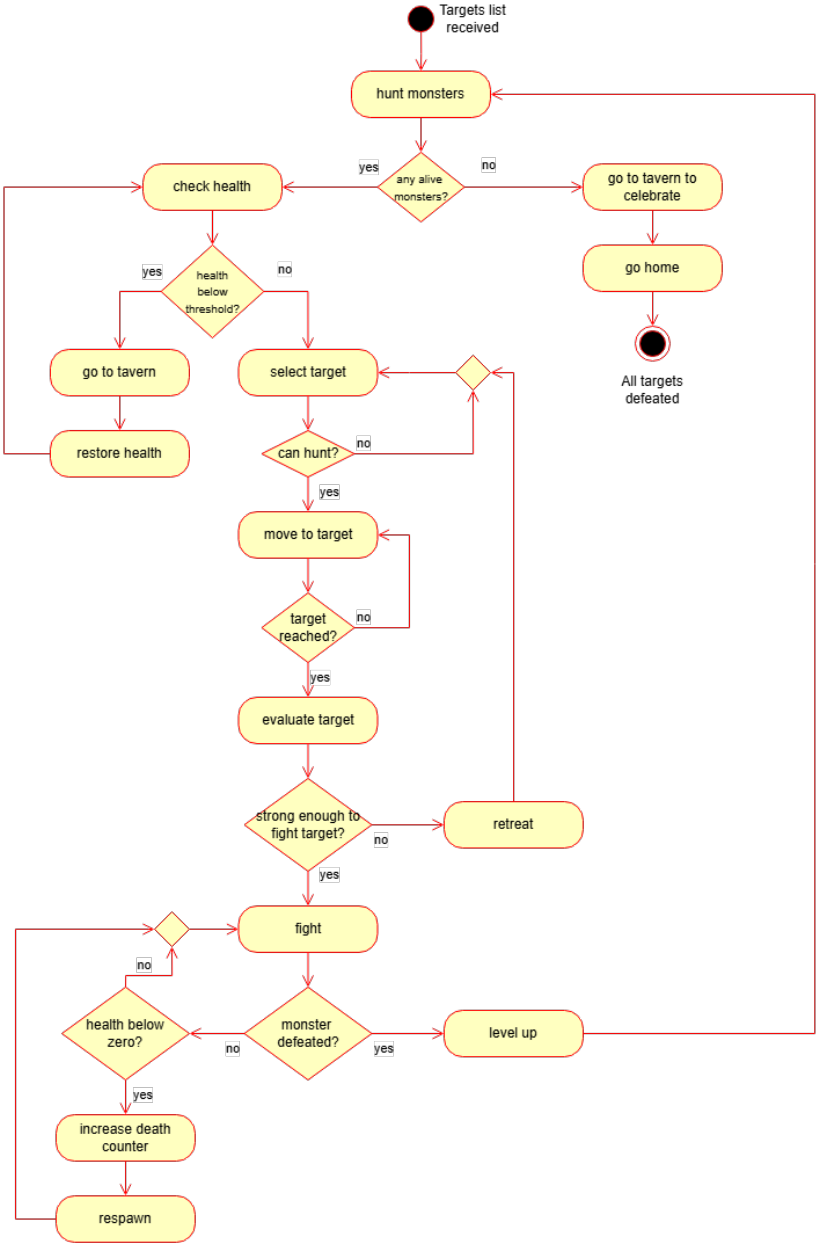

Figure 6: Witcher Activity Diagram

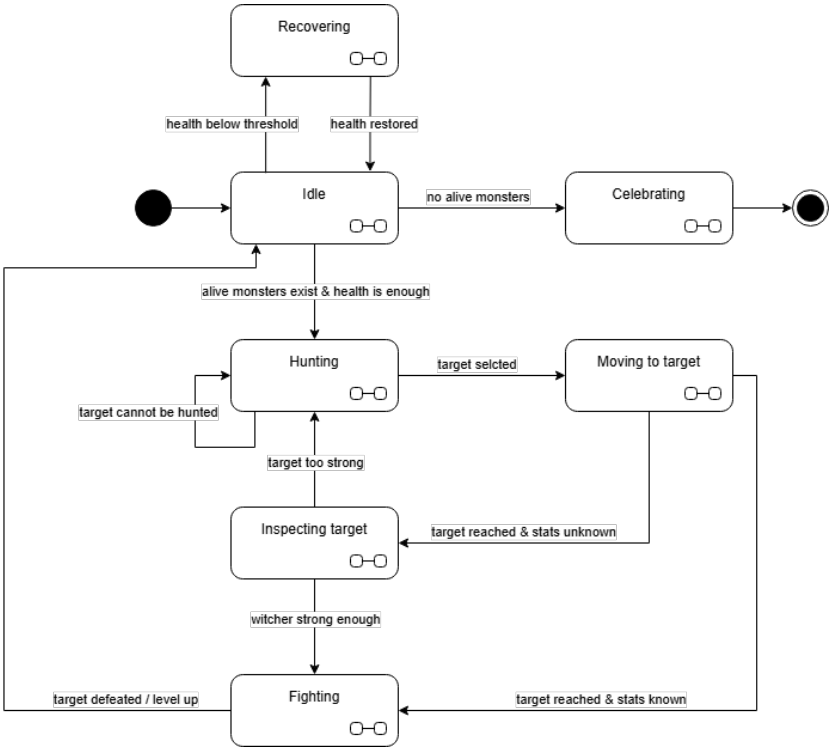

Figure 7: State Diagram

state he can go to *Recovering*, *Hunting* or *Celebrating* states. The choice depends on current health and monsters presence.

The *Hunting* state represents the search for a suitable target. If a valid target is selected, the Witcher goes to *Moving to target*. If the target cannot be hunted, the process remains within the hunting decision cycle.

The *Fighting* state represents direct combat. It is separated from inspection because combat is not only another decision step, but a distinct mode of behavior with different consequences. A successful fight leads to target defeat and level progression, after which the Witcher returns to the hunting cycle.

The chosen behavioral design follows a finite-state approach. A less structured approach was rejected because it made recursive behavior harder to control and reason about.

The monster agents have a much simpler behavioral role in the current design. Their behavior does not require a complex state machine, since they mainly act as targets within the arena and are characterised primarily by their presence and strength.

4.4 Architecture

The architecture of the system is organised around four main software modules: agents, environment, model, and GUI view. This organisation follows the design decision introduced earlier: decision-making, world-state management, and visualization should be separated into different responsibilities. Figure 8 shows the component diagram of the system.

The agents module contains the Witcher agent and the monster agents. These agents are responsible for autonomous behavior and decision-making. They observe the state of the world by means of percepts sent by environment. They affect the world state by means of actions such as "kill" and "apply damage".

Interfaces are used between the main components. The environment depends on the model and view abstractions rather than on concrete implementations. The GUI view also accesses the arena state through the model interface. This reduces coupling between components and makes the system easier to modify.

Overall, the component architecture supports modularity and maintainability. Each module has a clear role: agents decide, the environment mediates, the model stores and updates the world state, and the GUI view presents the result to the user.

4.5 Corner Cases

Several corner cases were considered during the design of the simulation and are handled as part of the normal behavior of the arena and the agents.

The first group of corner cases concerns invalid actions sent by agents to the environment. Since agents interact with the arena through action requests, the environment must check whether a received action is known and can be interpreted. If an action is not recognised, it is ignored and the state of the arena remains unchanged. This prevents unsupported or invalid requests from affecting the shared world state.

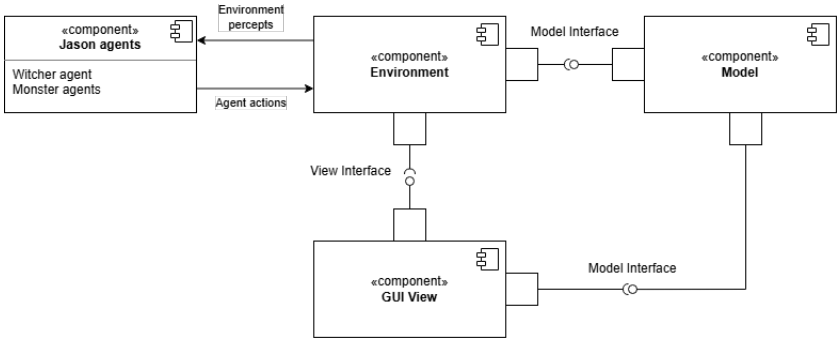

Figure 8: Component Diagram

For monsters, the main terminal case is defeat. When a monster is defeated, its status is changed to *dead*. Dead monsters are no longer treated as active targets and should not participate in target selection or combat. They are also removed from the visual representation of the arena, so the displayed state remains consistent with the simulation state.

A limitation discovered during implementation is the situation in which all remaining monsters are stronger than the Witcher. Without a fallback strategy, the Witcher may become stuck in a loop of selecting and rejecting unsuitable targets. This limitation is discussed further as future work.

5 Implementation

The project is divided into two main parts. The first part is the arena, implemented in Java. It contains the environment, the model, and the graphical representation of the world map. The second part is the agent logic, implemented in Jason. The Witcher and monster agents are written in a pure Jason style, using beliefs, goals, and context-dependent plans.

The graphical user interface is implemented using Java Swing. It provides a visual representation of the arena, fixed locations, agents, and relevant state changes during the simulation. The arena is displayed as a grid in which each agent can move in one of four directions. The tavern is marked with the letter `T`, while the home location is marked with the letter `H`. When the Witcher dies, the graphical view temporarily renders him in red before returning to the normal color, and the Death counter at the top of the GUI is incremented.

In addition to the GUI, the system also produces execution logs that describe the adventures of the Witcher. Examples of these messages are shown in Figure 9. The messages report important events such as monster defeat, level-up, Witcher death, and other state changes. The logs are mainly used for debugging and for following the internal progress of the simulation, since some decisions made by the agents are not always immediately visible in the graphical representation.

The Java part of the project is documented using Javadoc. The generated documentation describes the purpose and usage of the main methods. This documentation is included with the submitted artifacts in `docs/javadoc/index.html`. The Jason agent files also include annotations and comments that simplify navigation through the agent logic.

Error reporting is limited to diagnostic messages printed during execution. A more advanced error-reporting mechanism was considered unnecessary for the current scope of the project.

6 Deployment Instructions

The only prerequisite for launching the project is Java 17. The produced software artifacts can be installed and launched in a completely clean environment using the Gradle wrapper included in the project.

Install Java 17. Then clone the project repository to the target machine. Open a terminal in the root directory of the project, where the Gradle wrapper file is located, and run:

```
./gradlew runProjectMas
```

This command builds and launches the application using the predefined Gradle task `runProjectMas`.

Figure 9: Logs

7 Conclusion and Final Comments

This report presented the analysis, design and development of a Multi-Agent System based on the Belief-Desire-Intention (BDI) paradigm using the Jason framework. By combining agent-oriented programming with established software engineering principles such as modular design and separation of concerns, a simulation was developed in which autonomous agents perceive their environment, reason about their actions and interact within a shared world.

7.1 Future Works

There are four main features which could make this simulation more interesting:

- Randomized damage values could make the behavior of the system less predictable and more engaging. At the moment, monsters and the Witcher always deal a fixed amount of damage. Introducing a chance to miss an attack or perform a critical hit would make battles more dynamic and less deterministic.
- More complex monster behavior could also make the simulation more meaningful. Currently, monsters remain static at their assigned locations. A possible improvement would be to allow monsters to move within a limited area, so that the Witcher would have to track them before starting combat.
- The simulation currently assumes that there is always at least one monster weaker than the Witcher. If all monsters are stronger than the Witcher, he may become stuck in a loop of repeatedly rejecting targets. To address this situation, an alternative way of gaining experience or increasing strength could be introduced, allowing the Witcher to progress without defeating a monster first.
- Another possible improvement would be to apply a strict separation between the agent's bodies and minds [4]. In this approach, the agents would be responsible mainly for decision-making, while physical properties such as health, strength, level, and position would be stored in the model. The level-up operation could also be moved from the agent logic to the model or environment, since it changes the physical state of the Witcher rather than representing a reasoning step. This would make the architecture more consistent with the idea that the model is the single source of truth for the simulation state.

7.2 Lessons Learned

The project provided an opportunity to apply established software engineering knowledge to the field of intelligent agents. Although the Jason framework and the Belief-Desire-Intention (BDI) paradigm were entirely new concepts, the previously acquired concepts facilitated the integration of BDI into the simulation. This provided evidence of how fundamental software engineering principles can be successfully transferred to emerging technologies and paradigms.

Working with Jason provided deeper understanding of how autonomous agents reason and interact with their environment. The project introduced a goal-oriented approach in agents instead of solely relying on traditional procedural programming. Throughout the course of developing the simulation, we noticed how agents perceive their surroundings, update their beliefs, deliberate on their actions and execute actions accordingly. Implementing these concepts within a simulation environment made the BDI reasoning cycle more intuitive and helped demonstrate how intelligent behaviors can emerge from the interaction between multiple autonomous agents.

Although the current project focuses on a simulation, it also highlights the potential of intelligent agents in modern game development. As discussed in the literature, many contemporary games still rely on scripted Non-Player Characters (NPCs), limiting their ability to adapt to changing situations or exhibit believable autonomous behavior. The architecture that was proposed in the project demonstrates how BDI agents can be integrated into a game environment while maintaining a clear separation between reasoning and game logic. This provides a foundation that could be extended beyond a simulation into a fully featured video game. Such an extension would represent an exciting direction for future work, and could even be relevant to the gaming industry as a whole.

Overall, the project illustrates the value of combining traditional software engineering practices with agent-oriented programming. By integrating various concepts, it was possible to design a modular and maintainable system that bridges conventional software development with intelligent autonomous behavior.

References

[1] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[2] Deekshitha Kosaraju. Ai and multi-agent systems: Collaboration and competition in autonomous environments. *International Journal of Advanced Research in Science, Communication and Technology*, 2024.

[3] F. Dignum, J. Westra, W. A. van Doesburg, and M. Harbers. Games and agents: Designing intelligent gameplay. *International Journal of Computer Games Technology*, 2008.

[4] A. M. G. Pinheiro et al. Spyke3d: A new computer games oriented bdi agent framework. *International Journal of Computer Games Technology*, 2008.