



Together Is Better! Integrating BDI and RL Agents for Safe Learning and Effective Collaboration

Manuel Parmiggiani¹, Angelo Ferrando²^a and Viviana Mascardi¹^b

¹University of Genoa, Italy

²University of Modena and Reggio Emilia, Italy

Keywords: BDI Model, Reinforcement Learning, Among Us.

Abstract: What happens when symbolic knowledge is injected into the learning process of a sub-symbolic agent? What happens when symbolic and sub-symbolic agents collaborate? And what happens when they do not? This paper explores an innovative combination of symbolic – *i.e.*, Belief-Desire-Intention (BDI) – and sub-symbolic – *i.e.*, Reinforcement Learning (RL) – agents. The combination is achieved at two different logical levels: at the single agent level, we show how symbolic knowledge may be exploited to drive the learning process of a RL agent; at the multiagent system level, we show how purely BDI agents and purely RL agents behave in the complex scenario of the ‘Among Us’ videogame, and – more interestingly – what happens when BDI agents compete against RL agents, and when BDI and RL agents cooperate to achieve their goals.

1 INTRODUCTION

Multi-Agent Systems (MAS) represent a fascinating field at the intersection of artificial intelligence, distributed computing, and complex systems theory. These systems consist of multiple autonomous agents, each capable of perceiving their environment, making decisions, and executing actions to achieve individual or collective goals. Agents in a MAS can range from simple rules-based entities to sophisticated learning algorithms, and interact with each other and their environment to accomplish tasks that may be beyond the capabilities of any single agent (Wooldridge, 2009).


One key distinction within the realm of MAS lies in the approaches used for agent design and decision-making: symbolic agents and sub-symbolic agents (Ilkou and Koutraki, 2020). These categories represent different paradigms for modelling and reasoning within MAS, each with its own strengths, weaknesses, and applications.


Symbolic agents, also known as deliberative or logical agents, operate based on explicit representations of the rules of knowledge and reasoning (de Silva et al., 2020; Calegari et al., 2021). These agents employ symbolic logic, rule-based systems, or knowledge graphs to represent and manipulate knowl-

edge about their environment, goals, and other agents. They use formal methods of reasoning, such as deductive or abductive reasoning to derive conclusions and make decisions. Symbolic agents excel in domains where reasoning about abstract concepts, planning, and high-level decision-making are crucial.

In contrast, sub-symbolic agents, sometimes referred to as reactive or connectionist agents, rely on distributed, parallel processing, and statistical learning techniques to navigate their environment and make decisions (Ciatto et al., 2024; Calegari et al., 2020). These agents typically do not rely on explicit symbolic representations of knowledge; instead, they learn patterns and associations directly from data through techniques like neural networks, reinforcement learning, or evolutionary algorithms. Sub-symbolic agents are adept at tasks requiring pattern recognition, adaptive behaviour, and learning from experience, making them well-suited for applications such as robotics, game playing, and autonomous vehicle control.

While symbolic and sub-symbolic approaches represent distinct paradigms, they are not mutually exclusive, and hybrid approaches combining elements from both paradigms are common in MAS research. The choice between symbolic and sub-symbolic agents depends on factors such as the complexity of the task, the availability of domain knowl-

^a <https://orcid.org/0000-0002-8711-4670>

^b <https://orcid.org/0000-0002-2261-9926>

edge, computational resources, and the desired level of autonomy, adaptability, explainability.

In this paper, we investigate the combination of symbolic agents conceptualised following the Belief-Desire-Intention cognitive architecture (BDI (Rao and Georgeff, 1995; Rao, 1996)) and sub-symbolic Reinforcement Learning (RL) agents (Sutton and Barto, 1998). Both are implemented using the Jason interpreter for the AgentSpeak(L) declarative agent language (Bordini et al., 2007). The combination is carried out in two different ways:

1. At the agent level, we demonstrate that the implementation of the safe RL approach based on the teacher (García and Fernández, 2015) only requires the addition of a few lines of code to the RL agent implemented in Jason and gives the advantage of full transparency of the process, explainability, and traceability of the information learnt from the teacher;

2. At the MAS level, we challenge the BDI agents and RL agents to play together within a very complex environment based on the popular video game “Among Us”, and show that the teams involving both BDI and RL agents play better than homogeneous teams.

The paper is organised in the following way: Section 2 introduces the BDI model, the SARSA algorithm for RL, and the notion of ‘safe RL’; Section 3 describes SafeRLJ, our implementation of safe RL agents in Jason; Section 4 describes the ‘Among Us’ case study and how BDI and RL agents were implemented to play that game; Section 5 illustrates experimental results; Section 6 compares SafeRLJ with the related literature; finally, Section 7 concludes and outlines the future directions of our work.

2 BACKGROUND

2.1 The BDI Model and the Jason Language

The Belief-Desire-Intention (BDI) Architecture, started with the philosophical work of Bratman (Bratman, 1987) and pioneered by Rao and Georgeff (Rao and Georgeff, 1995), is renowned for its deliberative nature while also accommodating reactive components.

Developed to model intelligent agents capable of goal-directed behaviour while remaining responsive to their environment, the BDI architecture combines reactive components for swift responses to stimuli with deliberative components supporting higher-level reasoning and decision-making.

AgentSpeak(L) is a logic programming language that provides an abstract framework for programming BDI agents (Rao, 1996). In the following, we provide a simplified description of the main constructs of Jason (Bordini et al., 2007), a widely used interpreter for AgentSpeak(L). Elements of the language that are not relevant for our work are omitted.

The beliefs of an agent determine what an agent currently knows about itself, the other agents in the system, and the environment. They are defined as atomic formulae. The beliefs defined by the programmer at design time make up for the initial belief base. Other beliefs may be added dynamically during the agent’s lifetime either because the agent generated a new belief for itself, or as the result of perception of the environment, or as the effect of communication.

An achievement goal in Jason is specified as $!at$ where at is an atomic proposition.

Finally, actions may include user-defined actions that have some effects on the environment, implemented as a Java class in Jason, or library actions like `.send`, used to send a KQML message (Finin et al., 1994) to another agent, `.findall` used to store in a list all the solutions to a predicate call, as customary in Prolog, `.random` to generate a random number, `.max` to identify the maximum value in a list.

Plans are used to define the course of action for the agent to achieve its goals. A plan looks like

$$te : ctxt \leftarrow h.$$

where te is a trigger event that triggers the execution of the plan; $ctxt$ is the context that indicates the conditions that must be met to consider the plan applicable; and h is the body consisting of a sequence of steps to be executed. The trigger event may be the addition (resp. deletion) of a belief b , and the addition (resp. deletion) of a goal g . A plan is relevant for a triggering event if the event can be unified with the plan’s head. A plan is considered applicable if a condition $ctxt$ holds as a logical consequence of the agent’s belief. Note that the environment and other agents, via communication, can add new beliefs into one agent’s mind, hence triggering the execution of applicable plans. The body of a plan is made up of actions (a), belief updates ($+b$, $-b$), and achievement goals ($!g$).

2.2 The Q-Learning RL Algorithm and Safe RL

Q-Learning is a reinforcement learning algorithm that is used to learn the optimal action selection policy for a given environment. It falls under the category of model-free reinforcement learning methods, meaning that it does not require prior knowledge of the envi-

ronment’s dynamics.

In Q-Learning, an agent interacts with an environment by taking actions and receiving rewards. The goal is for the agent to learn a policy that maximises the cumulative reward over time. At each time step t , the agent selects an action a_t based on the current state s_t . Upon taking action a_t , the agent transitions to a new state s_{t+1} and receives a reward r_{t+1} .

The Q-value, denoted as $Q(s, a)$, represents the expected cumulative reward the agent will receive by taking action a in state s and following the optimal policy thereafter.

The Q-value of a state-action pair (s, a) is updated iteratively using the Bellman equation:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

where $Q(s, a)$ is the Q-value of taking action a in state s , α is the learning rate, controlling the impact of new information, r is the immediate reward obtained after taking action a in state s , γ is the discount factor, determining the importance of future rewards, s' is the next state after taking action a , and a' is the possible actions in state s' . The algorithm implementing the iteration on the Bellman equation is named SARSA, for State–Action–Reward–State–Action.

The Q-table is a data structure that stores the Q-values for all possible state-action pairs. Each row of the Q-table corresponds to a state, and each column corresponds to a possible action. Thus, the Q-table serves as a lookup table for the agent to determine the best action to take in a given state.

Unfortunately, what is learnt by the agent via the Q-learning algorithm may be suboptimal, in some cases even wrong, or very dangerous in the long run although very rewarding in the short run. In other words, it may be unsafe.

According to García and Fernández (García and Fernández, 2015)

“Safe Reinforcement Learning can be defined as the process of learning policies that maximise the expectation of return in problems in which it is important to ensure reasonable system performance and/or respect safety constraints during the learning and/or deployment processes.”

In their survey, they consider two ways to inject safety into the RL process: by modifying the optimality criterion to introduce the concept of risk and by modifying the exploration process to avoid exploratory actions that can lead the learning system to undesirable or catastrophic situations. W.r.t. this second category of approach, they further identify two ways to achieve the goal: (i) through the incorporation of external knowledge, (i.a) providing initial knowledge, (i.b) deriving a policy from a finite set of

demonstrations, and (i.c) providing teach advice, and (ii) through the use of a risk-directed exploration.

In this paper, we are interested in improving the safety of the RL exploration process by taking advantage of external knowledge. Thanks to the Jason framework we adopt, knowledge can be represented in a symbolic way, making the approach transparent and interpretable, and possibly taught by a teacher agent, making it very realistic and close to how humans learn.

3 SafeRLJ: IMPLEMENTING SAFE RL AGENTS IN JASON

A natural way to port the notion of Q-value into an AgentSpeak(L) learning agent is as a set of beliefs. Specifically, as shown in Figure 1, each Q-value can be assigned to a term $st_act(S, A, V)$ that establishes a relationship between states, actions, and their expected reward.

The Bellman equation can be implemented in AgentSpeak(L) to iteratively update the Q-value beliefs as a result of the execution of actions in the environment, that is implemented to send a belief to the agent, with the reward due to the action’s execution, and with the new state resulting from it. To store the hyperparameters, the following beliefs are used: $learn_rate(\alpha)$ (learning rate), $disc_factor(\gamma)$ (discount factor), $epsilon(\epsilon)$ (used to drive the learning process).

LA_Plan.1 in the learning agent’s code is triggered by the addition of the $st(S)$ belief. This belief determines the current state of the agent and its addition triggers a new update iteration of the associated Q-value.

```
(LA_Plan.1) +st(S) :  $\top$   $\leftarrow$ 
.random(R); .findall((V,A), st_act(S,A,V), L);
!select_act(R,L,Action); +my_act(Action);
execute_act(Action,S).
```

Considering the state S under evaluation, **LA_Plan.1** first generates a random value between 0 and 1 ($.random$ action). Then, thanks to $.findall$, it retrieves all the tuples (V, A) associated with state S from the belief base, where A is an action that can be performed in S and V is the currently associated value (*i.e.*, the expected reward of performing A in S). The $!select_act$ achievement goal is executed and when, due to its completion, the $Action$ variable has been unified with one value, the $my_act(Action)$ belief is added to the belief base. $Action$ is hence executed in the Java environment associated with the Jason MAS thanks to the user-defined $execute_act(Action, S)$

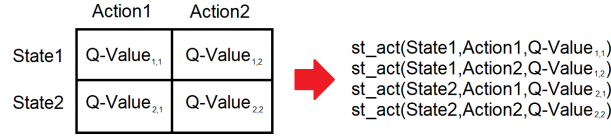


Figure 1: Mapping of the Q-Learning state-action matrix into a set of AgentSpeak(L) beliefs.

Jason action.

Upon completion of **LA_Plan_1**'s execution, the environment adds a belief in the agent's belief base, that is $new_st(S)$. To handle such a belief, the agent has two mutually exclusive plans.

(LA_Plan_2) $+new_st(S_2, Rew) :$
 $st(S_1) \wedge my_act(A) \wedge st_act(S_1, A_1, V_1) \wedge$
 $\neg ext_st_act(S_1, A_1, -) \wedge learn_rate(\alpha) \wedge disc_factor(\gamma) \leftarrow$
 $.findall(NV, st_act(S_2, -, NV), L); max(L, Value);$
 $NV_1 = V_1 + \alpha * (Rew + \gamma * Value - V_1); -my_act(A_1);$
 $-st(S_1); -st_act(S_1, A_1, V_1); +st_act(S_1, A_1, NV_1); +st(S_2).$

LA_Plan_2 is triggered when an action has been completed and a resulting state is available, along with its immediate reward (*i.e.*, the reward obtained by performing the action). This plan can only be applied if there was no external knowledge on the reward given to the couple (S_1, A_1) (condition $\neg ext_st_act(S_1, A_1, -)$ in the plan context). The plan takes into account the hyper-parameters discussed previously, gets the highest expected reward that can be obtained in the new state S_2 (the state reached by applying the action on the environment), and a new expected reward is computed for state S_1 according to Bellman equation. Finally, the agent's belief base is consistently updated by removing the beliefs that are no longer useful and by adding the new state as the current state.

In case $ext_st_act(S_1, A_1, GV)$ is instead a logical consequence of the agent's beliefs, no 'learning by experience' takes place and the new state-action-reward triple stored in the Q-table is $st_act(S_1, A_1, GV)$, hence integrating given external knowledge instead of learned knowledge.

(LA_Plan_3) $+new_st(S_2, Rew) :$
 $st(S_1) \wedge my_act(A) \wedge st_act(S_1, A_1, V_1)$
 $\wedge ext_st_act(S_1, A_1, GV) \leftarrow$
 $-my_act(A_1); -st(S_1); -st_act(S_1, A_1, V_1);$
 $+st_act(S_1, A_1, GV); +st(S_2).$

Note that, the $+st(S_2)$ belief addition will trigger **LA_Plan_1** again, and hence it will activate further iterations which will cause the agent to learn the policy bringing the highest reward (standard Q-Learning).

The Jason code described in this section implements the exploration stage. The exploitation stage just requires that the learned Q-values are used to decide what to do next, give the current state. During exploitation, no Jason plans are needed by the agent,

apart those for converting the symbolic representation of the selected action, into a call to a real action to be performed in the environment.

Beliefs of type $ext_st_act(S_1, A_1, GV)$, needed during the exploration stage, can be present in the learner agent's mind since the very beginning due to a design choice by the developer, or, more interestingly, can be shared with the learner via communication with one or more teacher agents. Communication is natively supported by Jason, and we experimented with this teacher-learnt approach.

We implemented a very simple teacher agent with an initial goal $!teach(List_Of_St_Act_Val_Triples)$ and two plans triggered by the addition of this goal to the goal queue.

(TA_Plan_1) $+!teach([s(S, A, V)|Tail]) : \top \leftarrow$
 $.send(Learner, tell, ext_st_act(S, A, V)); !teach(Tail).$

(TA_Plan_2) $+!teach([]) : \top \leftarrow$
 $.print("teaching completed").$

$List_Of_St_Act_Val_Triples$ is implemented as a Prolog list and may either be the empty list $[]$, managed by **TA_Plan_2**, or a non empty list with $s(S, A, V)$ as head, and $Tail$ that is the remainder of the list. In **TA_Plan_1** the teacher agent sends the belief $ext_st_act(S, A, V)$ to the learner agent, generically represented by the logical variable $Learner$ that should be substituted with the learner's name in the code, and then executes $!teach(Tail)$ to implement recursion over the list's tail.

The behaviour of the teacher shows how easily safe learning by communication can be implemented in Jason. It can become more and more complex, with dynamic generation of what to teach, based on complex logical reasoning, further communication with other teacher agents, and with the learner agent itself.

4 THE AMONG US CASE STUDY

As a motivating example and case study of this work, we tackle the 'Among Us' game (InnerSloth LLC, 2018), which is complex enough to represent a challenging testbed for our approach.

4.1 Rules of the Game

Among Us is a game centred around the concepts of trust and cooperation, where various player types collaborate within a shared environment, each with distinct objectives. None of the players can achieve their mission alone, so it is crucial for them to identify teammates and collaborate with them to accomplish their shared goal before the opposing team does.

The game is set in a futuristic space station where players take on the roles of crewmates or impostors. There are a certain number of rooms on the spaceship and several tasks to be performed in each room. When the game starts, each player is assigned a room as their initial room. The players then start looking for tasks to perform in that room and move when they decide that there is nothing left to do in their current room. Players can travel between rooms and interact with other players to help or hinder them, depending on the team the player believes that the other player belongs to. The game ends when one of the specified ending conditions is met.

Types of Players and Their Goals. Crewmates are tasked with completing a series of tasks scattered throughout the spaceship, while impostors aim to sabotage the crew's efforts and eliminate crewmates one by one. The fact that impostors look and act just like crewmates makes it difficult for the crew to identify them. This leads to a game of cat and mouse, as crewmates use their wits and instincts to uncover the impostors and eject them before it is too late. When the game starts, each crewmate only knows the colours of the other players and addresses them by their colours. The crewmate observes their behaviour to classify them as allies or enemies and ultimately accuses them of being impostors if they are deemed untrustworthy. They seek the support of other crewmates during the voting phase.

Voting Phase. A voting phase is initiated whenever a crewmate identifies another player as untrustworthy. During the voting phase, each of the two involved players tries to convince the rest of the group that the other player is the impostor. There is also a feature called the "advantage of speaking first", which benefits the first player to report the other as an impostor. This feature can be crucial to the outcome of the vote: typically, the reporting player needs the agreement of at least one other crewmate to have the reported player expelled. However, if the reporting player is an impostor, another crewmate must expose the lie and gain the consent of at least a third crewmate. Winning a dispute brings a player closer to victory for their team.

If the crewmate wins the dispute, the impostor will be expelled and the crewmates will have one fewer enemy player. However, if the impostor wins the dispute, the completion of tasks will slow down due to the crewmate's expulsion.

Game-Ending Conditions. As the game progresses, the number of players decreases along with the number of tasks to complete. There are three possible scenarios in which the game can end:

1. All the crewmates are eliminated and expelled from the spaceship. In this scenario, impostors win and take over the spaceship.

2. All the machines on the spaceship are fixed, the central computer is fully operational again and capable of recognising who the impostors are. The impostors are then expelled from the spaceship by the central computer, and the crewmates win the game.

3. All impostors are unmasked by the crewmates and expelled from the spaceship before all the machines are fixed. This scenario is the rarest and occurs because the crewmates have exposed each impostor before completing all of their tasks. Once again, this results in a win for the crewmates.

4.2 How Symbolic Agents Play Among Us

In this section, we report how BDI agents can be used to tackle the Among Us game. Note that we focus only on the most relevant aspects; the complete implementation of the agents can be found in the supplementary material.

We start with the BDI agents. However, instead of reporting the AgentSpeak(L) plans, in order to improve readability, we report the agents' behaviour via UML sequence diagrams.

The first kind of agent we consider is the crewmate one. As we can see, the task of the crewmate is quite straightforward. The agent checks whether there are any tasks to be completed in the room where it is in and, upon receiving such perception from the environment, the agent reacts consequently. Specifically, if there are indeed tasks to complete, the agent will complete them (this can be achieved by performing the actual task exploiting a specific environment action); otherwise, in case no more tasks are available in the room, the agent simply moves to another room.

A more challenging agent is the impostor one, which has a less straightforward behaviour than its crewmate counterpart.

Killer impostors aim to kill crewmates. That is, this agent looks into the room and if a crewmate is detected, the impostor agent kills them. Note that this is

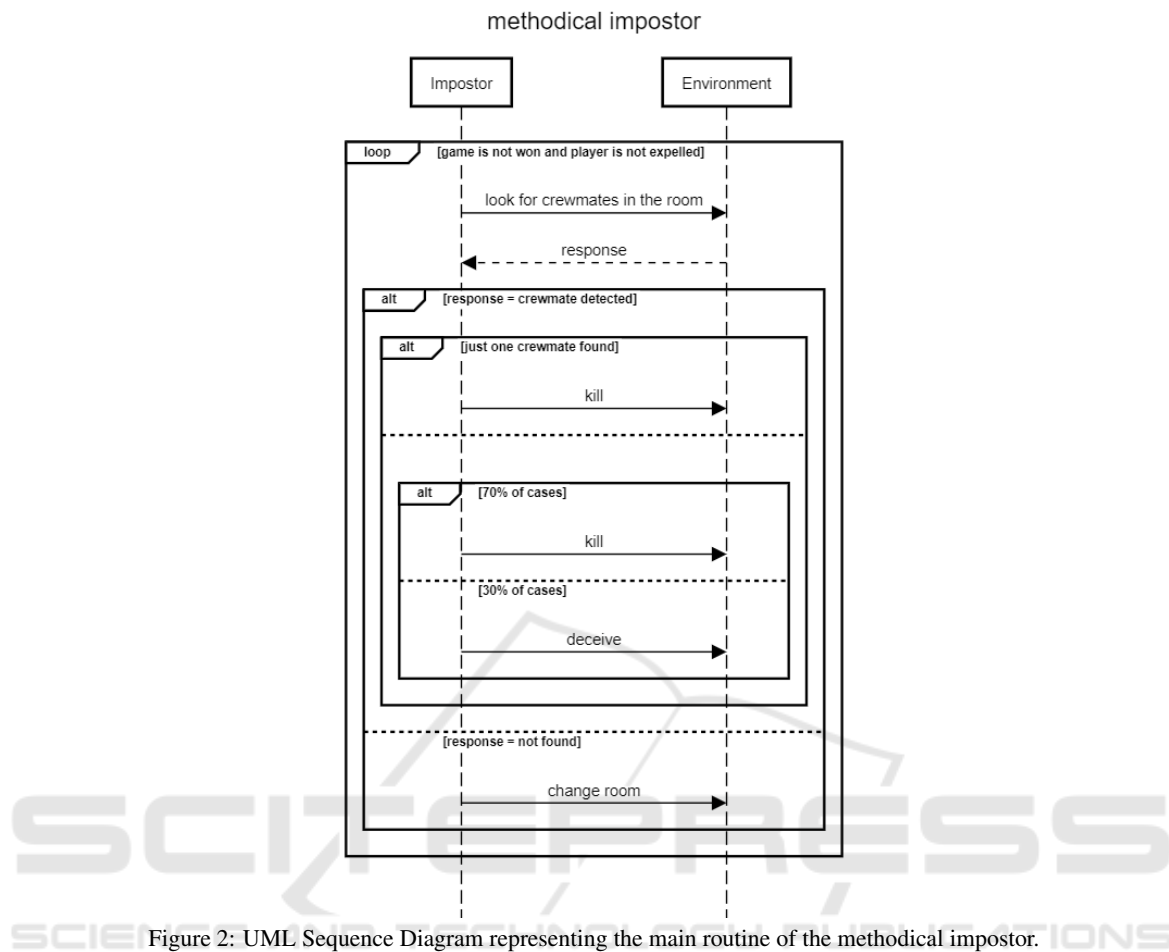


Figure 2: UML Sequence Diagram representing the main routine of the methodical impostor.

done without checking whether there are other crewmates in the room who may witness the murder.

Standard impostors change their action depending on the presence of crewmates in the room. Decide, with a fixed probability, whether to kill a crewmate or not, in the presence of possible witnesses. Methodical impostors instead have a more complex behaviour, shown in Figure 2; in fact, they take into consideration the presence of crewmates in the room. However, differently from before, such agents distinguish between seeing only one or multiple crewmates. In the former case, the agent safely kills the solitary crewmate, while in the latter case, the agent decides with a fixed probability whether to kill one of the crewmates in the room or not.

It is worth noting that these behaviours have been implemented as BDI plans and represent only a subset of the agents' behaviour.

4.3 How Sub-Symbolic Agents Play Among Us

Differently from their symbolic counterpart, sub-symbolic agents do not have plans, but can learn their own policies. To do so, they need to experiment with the environment to get back feedback on which actions to perform in which scenarios. A Java Environment class has been implemented in Jason to compute rewards and next states, in a way that is coherent with the Among Us rules of the game. The interaction between the Environment and the RL agents implemented in Jason has been described in Section 3: during their exploration stage, the RL agents execute actions on the Environment, and the Environment produces $new_st(S, Reward)$ percepts that enter the agent's belief base and trigger either **LA_Plan_2** or **LA_Plan_3**. A teacher agent may be used, with an initial goal that may look like (as an example for teaching the impostor RL agent) $!teach([s(found2orMore, kill, 0), s(found1, kill, 2)])$ suggesting that when two or more agents are in the

same room as the impostor, killing is a bad choice, as some crewmate might identify the impostor and report it. Killing should instead be done when only one crewmate is in the room.

5 EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we report the experiments we carried out on the Among Us case study. First, we show the results obtained by employing BDI agents, then, by employing RL agents, and finally, by combining the two. For space constraints, we focus exclusively on the impostor agents, the reason lies in their being more challenging and complex than their crewmate counterpart. We also focus on the analysis of homogeneous/heterogeneous teams, as we are interested in checking if there are advantages of mixing BDI and RL agents. The safe learning mechanism is not considered here: we run experiments to make sure that learner agents always learn policies coherent with what the teacher teaches them, and this happens. In this section, we let RL agents learn by experience only, without any external hint.

5.1 BDI Experiments

The experiments reported in this section aim at understanding which impostor strategy and which crewmate skill level would make the game as balanced (and hence as significant) as possible.

This section outlines our empirical approach to identifying optimal impostor strategies to maximise win ratios. To facilitate frequent impostor gameplay and precise performance tracking, we intentionally reduced the skill level of crewmates, specifically their task completion probability.

The first strategy analysed is the “killer impostor”, who prioritises eliminating crewmates over other strategies like deception. Figure 3 shows that killer impostors achieve a high win ratio with their direct approach. In games in which the crewmates win, the survivor count is notably low, demonstrating the effectiveness of the killer impostor in reducing the number of crewmates. In contrast, impostor victories often see a high survival rate, with nearly 60% of wins occurring without impostor losses. However, their indiscriminate killings sometimes lead to early expulsion from the game.

Our second experiment (Figure 4) examines “standard impostors”, who balance killing crewmates with faking task completion. These impostors achieve a lower win ratio, likely due to suboptimal decision

making in choosing between deception and elimination. Standard impostors also leave fewer tasks incomplete compared to killer impostors, indicating a potential inefficiency in prioritising deception when a kill would not compromise their cover.

To address this inefficiency, we introduce the “methodical impostor”, which combines the speed of killer impostors with the strategic discretion of standard impostors. Considering the number of crewmates present, methodical impostors optimise their actions, leading to a surprisingly higher win ratio (Figure 5).

Furthermore, the methodical impostor’s use of deception results in more surviving crewmates in games won by their team. Although this may extend the game’s duration, it contributes to a more balanced and strategic game-play experience. In light of these findings, **we select the methodical impostor as the optimal strategy.** Subsequently, we adjust the skill level of the crewmates to ensure balanced game play. Through multiple experiments, we determine that **a crewmate skill level of approximately 0.52 yields a win rate close to 50% for the crewmates**, as shown in Figure 7. Methodical impostors and crewmates with 0.52 skill level play in a balanced way and will be used in the RL-BDI experiments.

5.2 RL-BDI Experiments

In the following experiments, crewmates are always implemented as BDI agents crewmates with skill level **0.52**. What changes is the configuration of teams of impostors.

Team Comprising RL Impostors Agents Only.

When RL agents act as the sole impostors, their performance is poor, and RL-only teams lose almost all games, as shown in Figure 8 (RL-only impostor teams win ratio: 17.5%). Analysis reveals a key behaviour: RL impostors avoid eliminations in the presence of multiple crewmates, prioritising their own survival over the team’s success. This self-serving, reactive nature undermines the collaborative and strategic requirements of Among Us, where teamwork and long-term planning are crucial to victory. Given these limitations, we investigate hybrid team compositions that combine RL impostors with methodical BDI impostors to assess whether RL behaviour can contribute positively in a mixed-strategy context.

Half of the Team Comprising RL Impostor Agents.

Replacing 2 RL impostors with 2 methodical BDI impostors significantly improves the win ratio (Figure 9: 2RL impostors + 2 methodical BDI impos-

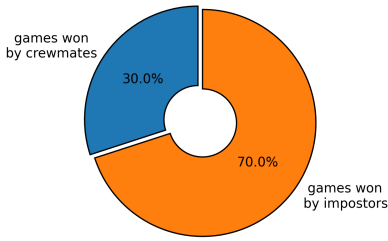


Figure 3: Win ratio of **killer impostors** versus **poorly skilled** crewmates in BDI-only setting.

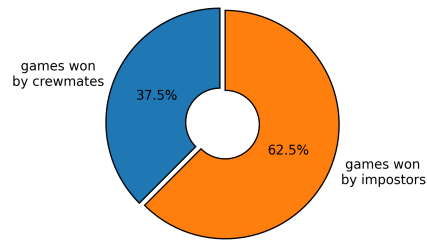


Figure 4: Win ratio of **standard impostors** versus **poorly skilled** crewmates in BDI-only setting.

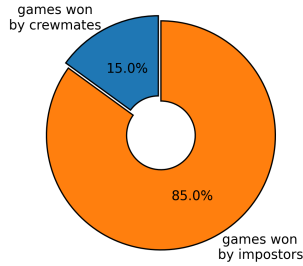


Figure 5: Win ratio of **methodical impostors** versus **poorly skilled** crewmates in BDI-only setting.

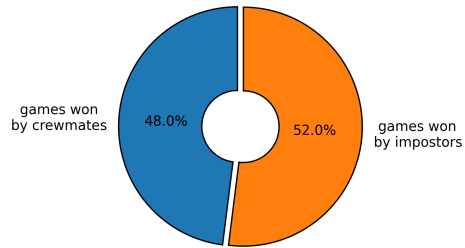


Figure 6: Win ratio of **methodical impostors** versus crewmates with skill level **0.52** in BDI-only setting.

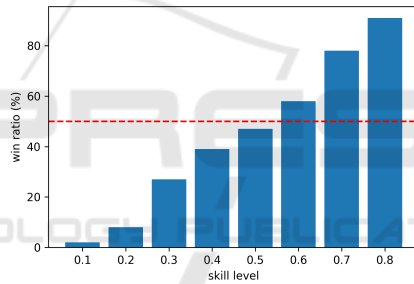
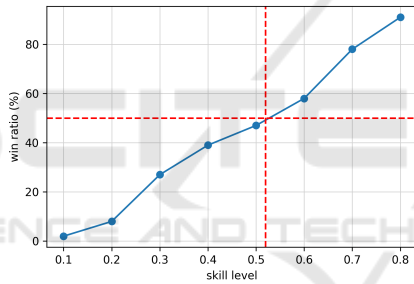


Figure 7: Win rate of crewmates against methodical impostors as a function of crewmate’s skill level in BDI-only setting. **Outcome:** crewmates skill level of **0.52** makes games balanced and will be used in RL-BDI experiments.

tors team win ratio reaches 44.4% that is better than the 17.5% win ratio of RL-only impostors team, but still lower than 52% win ratio for BDI-only impostor team). This indicates that the methodical BDI impostors can positively influence the “selfish” RL impostors, whose performance initially appeared unsuitable for the game’s collaborative context.

Analysis of game outcomes revealed key differences in the dynamics of the early game.

RL Impostor Teams: Pure RL impostor teams rarely achieve victory, as early crewmate eliminations are rare. Crewmates often cluster in initial rooms due to random assignments, and RL impostors prioritise faking tasks in the presence of others, delaying kills.

Mixed RL and BDI Impostor Teams: Introducing two BDI impostors significantly changes early-game strategy. BDI agents disrupt crewmate clustering, prompting RL impostors to initiate kills earlier, resulting in a more effective team dynamic.

These conclusions are based on the average evolution of the game observed in terms of tasks completed, crewmate survival, and overall performance (Figure 11).

A Quarter of the Team Comprising RL Impostor Agents. Although the 50-50 hybrid team outperforms the pure RL team, the pure BDI team shown in Figure 6 remains superior, with 52% win ratio against the 44.4% shown in Figure 9. To explore whether a different configuration could exceed the pure BDI team’s performance, we tested a team with three BDI impostors and one RL impostor, building on the improvement seen from the 4 RL to the 2 RL + 2 BDI configuration.

As shown in Figure 10, this set-up achieves 55% win ratio, that is a better result than the one achieved by the pure BDI team. Further analysis attributes this improvement to: (i) The presence of three BDI im-

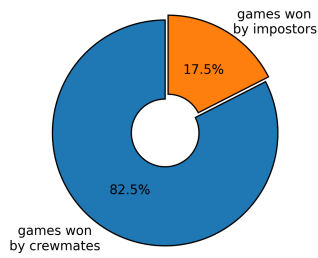


Figure 8: Win ratio when the impostor team consists **solely of RL impostors** and BDI crewmates have skill level **0.52**.

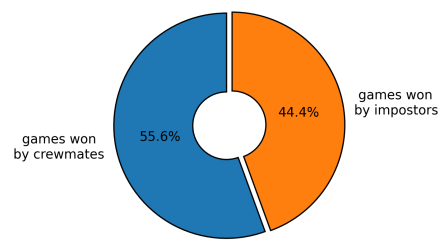


Figure 9: Win ratio when the impostor team consists of **2 RL and 2 BDI methodical impostors** and BDI crewmates have skill level **0.52**.

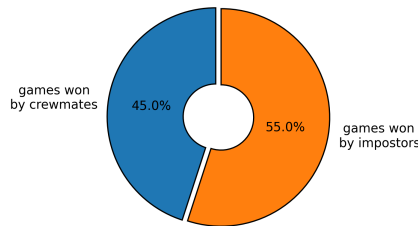


Figure 10: Win ratio when the impostor team consists of **1 RL impostor and 3 BDI methodical impostors** and BDI crewmates have skill level **0.52**.

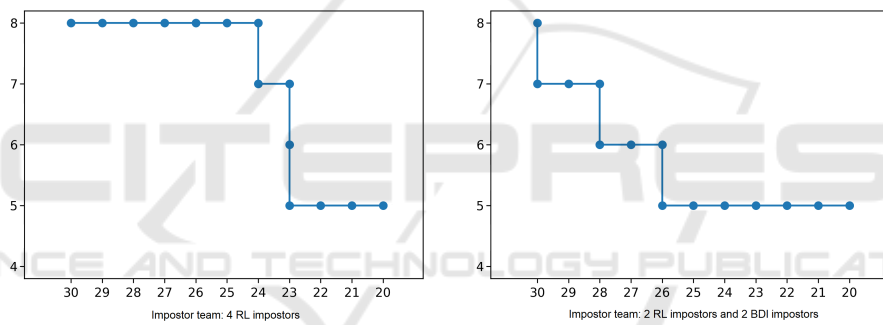


Figure 11: Initial evolution of the game in the two different scenarios.



Figure 12: Win ratio w.r.t. number of RL impostors.

postors accelerates the elimination of crewmates early in the game, prompting the RL impostor to begin eliminating crewmates earlier. (ii) The behaviour of the RL impostor, theoretically making it impossible to be caught, as it only kills when unobserved. This grants the impostor team resilience to expulsions, as

the RL impostor acts as an “immortal player”. Although this experiment yielded positive results, previous experiments provided valuable insights leading to this conclusion. Figure 12 summarises the performance of each team configuration, with the red line denoting the performance of the pure BDI team.

Table 1: General information on recent proposals for integrating BDI and RL agents.

Ref.	Year	Language	RL Algorithm	Implementation of RL
(Badica et al., 2015)	2015	Jason 1.4.2	TDL	Jason Plan
(Badica et al., 2017)	2017	Jason 1.4.2	SARSA	Jason Plan
(Wan et al., 2018)	2018	Jason	SARSA	Jason Plan
(Bosello and Ricci, 2019)	2019	JaCaMo 0.7	SARSA	Jason Action
(Pulawski et al., 2021)	2021	Jason-RL	Iterative SARSA	As in (Bosello and Ricci, 2019)
(Persiani and Hellström, 2022)	2022	Python	DeepRL predict.	DeepRL in Keras
(Ichida and Meneguzzi, 2023)	2023	Python AS	Q-Learning FA	Outside
SafeRLJ	2024	Jason 3.1	SARSA	Jason Plan

Table 2: Features, examples, and licensing of recent proposals for integrating BDI and RL agents.

Ref.	BDI Integration	Safe	Hybrid Planning	Hybrid MAS	Example	Code	License
(Badica et al., 2015)	No	No	No	No	Toy	No	
(Badica et al., 2017)	No	No	No	No	Toy	(Felton, 2017)	GPL-3.0
(Wan et al., 2018)	No	No	Yes	No	Toy	No	
(Bosello and Ricci, 2019)	Yes	No	No	No	Compl.	(Bosello, 2019)	LGPL-3.0
(Pulawski et al., 2021)	As in (Bosello and Ricci, 2019)	No	No	No	Toy	No	
(Persiani and Hellström, 2022)	No	No	No	No	Toy	(Persiani, 2022)	MIT
(Ichida and Meneguzzi, 2023)	Yes	No	Yes	No	Compl.	No	
SafeRLJ	No	Yes	No	Yes	Compl.	(Not given, 2024)	MIT

Performance improvement with hybrid teams depends on the ratio of BDI to RL impostors; RL impostors are beneficial additions, but their effectiveness decreases if their number is too high.

6 COMPARISON WITH THE RELATED WORK

The integration of the BDI model with RL has been explored in fewer than 20 papers. Despite the limited number, several of these works have significantly advanced decision making in various applications. A recent survey on the topic, specifically Section 3.3, can be found in (Erduran, 2023). Based on this survey, we conducted a more systematic comparison of the RL and BDI works published over the last decade, summarised in Tables 1 and 2. Unlike Erduran’s analysis, our review delves into the details of the code, facilitated by the collaboration of authors of key articles (Badica et al., 2015; Badica et al., 2017; Bosello and Ricci, 2019; Persiani and Hellström, 2022; Ichida and Meneguzzi, 2023), who generously shared structured and detailed information about their proposals.

The idea of integrating RL and BDI dates back to the beginning of the millennium. (Norling, 2004) was one of the first authors to integrate the SARSA algorithm into BDI agents implemented using JACK Intelligent Agents, now <https://aosgrp.com.au/jack/>. Her goal was to have a rough approximation to the way humans learn to make decisions in an area in which they have expertise. (Lokuge and Alahakoon, 2005b; Lokuge and Alahakoon, 2005a) proposed a

hybrid BDI model that integrates RL to improve decision making in dynamic vessel berthing applications, while (Airiau et al., 2009) extended BDI agents with learning capabilities using Binary Decision Trees. Other works followed (Qi and Bo-ying, 2009; Lee and Son, 2009; Tan et al., 2011), but it is only from the research in (Badica et al., 2015) that we find approaches similar to ours, from which we could take inspiration.

Table 1 and Table 2 show the language in which the BDI framework is implemented (column **Lang.**), the RL algorithm implemented (**RL**) and where it is implemented (column **Impl. of RL**, stating whether the algorithm is implemented via a standard plan or action in the BDI framework or outside the BDI framework). The column **BDI int.** states whether the standard cycle implemented by the BDI interpreter was changed to accommodate RL, while the column **Safe** stores information about the support given to safe RL, as described by (García and Fernández, 2015). **Hy. Plans** is yes if hybrid plans are exploited in the proposal, where by hybrid plans we mean the coexistence of standard BDI plans and plans aimed at implementing the RL mechanism in the same agent’s code; **Hy. MAS** refers to the implementation of MAS in which pure BDI agents interact with pure RL agents. The column **Example** provides a qualitative evaluation of the complexity of the examples developed using the proposed approach. In most cases, the ‘toy example’ answer was given by the authors themselves. Finally, **Code** and **Lic.** report on the online availability of code and on its licence, respectively.

The unique features of SafeRLJ are the safety support and the experiments with pure BDI and pure RL agents that work together to achieve a common goal.

No previous proposal addressed these aspects. Apart from this, the works closer to SafeRLJ are (Badica et al., 2017; Wan et al., 2018) because they implement SARSA inside standard Jason plans, and in a way that is very similar to ours. However, the examples developed are very simple. Indeed, the dimension of the state-action matrix for the ‘Among Us’ case study is 243 cells for crewmates and 90 cells for impostors. It involves only the four actions of moving up, down, left, right in simple grid worlds with dimensions 4x3, 6x5 and 4x4 in (Badica et al., 2015), (Badica et al., 2017) and (Wan et al., 2018) respectively.

Concerning the scenario complexity, (Bosello and Ricci, 2019) run exhaustive experiments including a Grid world, the CartPole, MountainCar, and Car Intersection with 1080 floats state (reduced then to 20 trough averaging in preprocessing) and 3 actions. (Ichida and Meneguzzi, 2023) developed their MAS using Python-AgentSpeak (<https://github.com/niklasf/python-agentspeak>), and run experiments in a chatbot for a travel planning scenario, where the embedding of the action space has 276 dimensions and the embedding of the state / belief base has 100 dimensions. However, the RL algorithm is implemented outside the pure Jason agent, and both approaches modify the standard BDI interpreter, making the comparison difficult.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a study conducted on the integration of symbolic and sub-symbolic agents at two different levels, single- and multi-agent. We focused on BDI agents as symbolic representatives and RL agents as sub-symbolic counterparts. We introduced a straightforward, but still effective, approach to make sub-symbolic decisions safe, according to the ‘Safe RL’ paradigm based on external knowledge taught by a teacher. A complex video game scenario inspired by the well-known ‘Among Us’ game served as testbench for experimenting the effectiveness of homogeneous teams consisting of BDI or RL agents only, w.r.t. heterogeneous teams, where BDI and RL agents play together. Our results demonstrate the benefits of combining the two types of agents, compared to utilising singular techniques in isolation. The comparison with related work shows that the previous approach did not take safety and the possibility of mixing BDI and RL agents in the same application into account. Future directions include exploring safe RL and connecting it with the models of trust widely studied in the MAS community. In fact, the learner agent

might decide in a dynamic way whether to follow the teacher’s advice or not, based on the trust on the teacher. Since the SARSA algorithm is easy to understand and implement, but it comes with many limitations, we plan to take inspiration from the related literature and integrate more efficient learning methods in SafeRLJ, while ensuring the possibility of exploiting symbolic knowledge in the learning process.

REFERENCES

- Airiau, S., Padgham, L., Sardiña, S., and Sen, S. (2009). Enhancing the adaptation of BDI agents using learning techniques. *Int. J. Agent Technol. Syst.*, 1(2):1–18.
- Badica, A., Badica, C., Ivanovic, M., and Mitrovic, D. (2015). An approach of temporal difference learning using agent-oriented programming. In *20th International Conference on Control Systems and Computer Science, CSCS 2015, Bucharest, Romania, May 27-29, 2015*, pages 735–742. IEEE.
- Badica, C., Becheru, A., and Felton, S. (2017). Integration of jason reinforcement learning agents into an interactive application. In Jebelean, T., Negru, V., Petcu, D., Zaharie, D., Ida, T., and Watt, S. M., editors, *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017*, pages 361–368. IEEE Computer Society.
- Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. J. Wiley.
- Bosello, M. (2019). Companion code of Bosello et al. 2019. Accessed on December 31, 2024.
- Bosello, M. and Ricci, A. (2019). From programming agents to educating agents - A jason-based framework for integrating learning in the development of cognitive agents. In Dennis, L. A., Bordini, R. H., and Lespérance, Y., editors, *Engineering Multi-Agent Systems - 7th International Workshop, EMAS 2019, Montreal, QC, Canada, May 13-14, 2019, Revised Selected Papers*, volume 12058 of *Lecture Notes in Computer Science*, pages 175–194. Springer.
- Bratman, M. (1987). *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, Cambridge.
- Calegari, R., Ciatto, G., Mascardi, V., and Omicini, A. (2021). Logic-based technologies for multi-agent systems: a systematic literature review. *Auton. Agents Multi Agent Syst.*, 35(1):1.
- Calegari, R., Ciatto, G., and Omicini, A. (2020). On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32.
- Ciatto, G., Sabbatini, F., Agiollo, A., Magnini, M., and Omicini, A. (2024). Symbolic knowledge extraction and injection with sub-symbolic predictors: A systematic literature review. *ACM Comput. Surv.*, 56(6).

- de Silva, L., Meneguzzi, F., and Logan, B. (2020). BDI agent architectures: A survey. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4914–4921. ijcai.org.
- Erduran, Ö. I. (2023). Machine learning for cognitive BDI agents: A compact survey. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 15th International Conference on Agents and Artificial Intelligence, ICAART 2023, Volume 1, Lisbon, Portugal, February 22-24, 2023*, pages 257–268. SCITEPRESS.
- Felton, S. (2017). Companion code of Badica et al. 2017. Accessed on December 31, 2024.
- Finin, T. W., Fritzson, R., McKay, D. P., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, USA, November 29 - December 2, 1994*, pages 456–463. ACM.
- García, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480.
- Ichida, A. Y. and Meneguzzi, F. (2023). Modeling a conversational agent using BDI framework. In Hong, J., Lanperne, M., Park, J. W., Cerný, T., and Shahriar, H., editors, *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023, Tallinn, Estonia, March 27-31, 2023*, pages 856–863. ACM.
- Ilkou, E. and Koutraki, M. (2020). Symbolic vs sub-symbolic AI methods: Friends or enemies? In Conrad, S. and Tiddi, I., editors, *Proceedings of the CIKM 2020 Workshops co-located with 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), Galway, Ireland, October 19-23, 2020*, volume 2699 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- InnerSloth LLC (2018). Among us. <https://www.innersloth.com/gameAmongUs>. Video game.
- Lee, S. and Son, Y.-J. (2009). Dynamic learning in human decision behavior for evacuation scenarios under bdi framework. In *Proceedings of the 2009 INFORMS Simulation Society Research Workshop. INFORMS Simulation Society: Catonsville, MD*, pages 96–100.
- Lokuge, P. and Alahakoon, D. (2005a). Handling multiple events in hybrid BDI agents with reinforcement learning: A container application. In Chen, C., Filipe, J., Seruca, I., and Cordeiro, J., editors, *ICEIS 2005, Proceedings of the Seventh International Conference on Enterprise Information Systems, Miami, USA, May 25-28, 2005*, pages 83–90.
- Lokuge, P. and Alahakoon, D. (2005b). Reinforcement learning in neuro BDI agents for achieving agent's intentions in vessel berthing applications. In *19th International Conference on Advanced Information Networking and Applications (AINA 2005), 28-30 March 2005, Taipei, Taiwan*, pages 681–686. IEEE Computer Society.
- Norling, E. (2004). Folk psychology for human modelling: Extending the BDI paradigm. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 202–209. IEEE Computer Society.
- Not given (2024). Companion code of this paper, 2024. The code has been uploaded as supplemental material of this submission.
- Persiani, M. (2022). Companion code of Persiani et al. 2022. Accessed on December 31, 2024.
- Persiani, M. and Hellström, T. (2022). The mirror agent model: A bayesian architecture for interpretable agent behavior. In Calvaresi, D., Najjar, A., Winikoff, M., and Främling, K., editors, *Explainable and Transparent AI and Multi-Agent Systems - 4th International Workshop, EXTRAAMAS 2022, Virtual Event, May 9-10, 2022, Revised Selected Papers*, volume 13283 of *Lecture Notes in Computer Science*, pages 111–123. Springer.
- Pulawski, S., Dam, H. K., and Ghose, A. (2021). Bdi-dojo: developing robust BDI agents in evolving adversarial environments. In El-Araby, E., Kalogeraki, V., Pianini, D., Lassabe, F., Porter, B., Ghahremani, S., Nunes, I., Bakhouya, M., and Tomforde, S., editors, *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2021, Companion Volume, Washington, DC, USA, September 27 - Oct. 1, 2021*, pages 257–262. IEEE.
- Qi, G. and Bo-ying, W. (2009). Study and application of reinforcement learning in cooperative strategy of the robot soccer based on bdi model. *International Journal of Advanced Robotic Systems*, 6(2):15.
- Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In *7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer.
- Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In Lesser, V. R. and Gasser, L., editors, *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, pages 312–319. The MIT Press.
- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.
- Tan, A., Ong, Y., and Tapanuj, A. (2011). A hybrid agent architecture integrating desire, intention and reinforcement learning. *Expert Syst. Appl.*, 38(7):8477–8487.
- Wan, Q., Liu, W., Xu, L., and Guo, J. (2018). Extending the BDI model with q-learning in uncertain environment. In *Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence, ACAI 2018, Sanya, China, December 21-23, 2018*, pages 33:1–33:6. ACM.
- Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems, Second Edition*. Wiley.