Safe Multi-Agent Reinforcement Learning with Quantitatively Verified Constraints

JOSHUA PAUL RILEY

DOCTOR OF PHILOSOPHY

UNIVERSITY OF YORK

COMPUTER SCIENCE

DECEMBER 2022

Abstract

Multi-agent reinforcement learning is a machine learning technique that involves multiple agents attempting to solve sequential decision-making problems. This learning is driven by objectives and failures modelled as positive numerical rewards and negative numerical punishments, respectively. These multi-agent systems explore shared environments in order to find the highest cumulative reward for the sequential decision-making problem. Multi-agent reinforcement learning within autonomous systems has become a prominent research area with many examples of success and potential applications. However, the safety-critical nature of many of these potential applications is currently underexplored—and under-supported. Reinforcement learning, being a stochastic process, is unpredictable, meaning there are no assurances that these systems will not harm themselves, other expensive equipment, or humans. This thesis introduces Assured Multi-Agent Reinforcement Learning (AMARL) to mitigate these issues. This approach constrains the actions of learning systems during and after a learning process. Unlike previous multi-agent reinforcement learning methods, AMARL synthesises constraints through the formal verification of abstracted multi-agent Markov decision processes that model the environment's functional and safety aspects. Learned policies guided by these constraints are guaranteed to satisfy strict functional and safety requirements and are Pareto-optimal with respect to a set of optimisation objectives. Two AMARL extensions are also introduced in the thesis. Firstly, the thesis presents a Partial Policy Reuse method that allows the use of previously learned knowledge to reduce AMARL learning time significantly when initial models are inaccurate. Secondly, an Adaptive Constraints method is introduced to enable agents to adapt to environmental changes by constraining their learning through a procedure that follows the styling of monitoring, analysis, planning, and execution during runtime. AMARL and its extensions are evaluated within three case studies from different navigation-based domains and shown to produce policies that meet strict safety and functional requirements.

Acknowledgements

My deepest gratitude to my supervisors, Professor Radu Calinescu and Dr Colin Paterson, for their invaluable support and advice that has helped shape this thesis and my abilities as a researcher throughout this PhD. Their patience and understanding were, in no small part, a driving force behind the work involved in this project, which would have severely lacked without their reliable feedback and willingness to help. I also want to thank Dr Daniel Kudenko for his advice as a supervisor during the early development of the PhD and the helpful feedback he gave towards its advancement.

I would also like to thank my examiners, Dr Dimitar Kazakov and Professor Vasile Palade, for their comments and feedback, which have helped strengthen this thesis.

I am also very grateful to the Defence Science and Technology Laboratory for sponsoring this project. In particular, I would like to thank Dr Alec Banks for the time he invested in this project and the helpful feedback he generously gave during many conferences and meetings.

Furthermore, I want to thank the members of the Assuring Autonomy International Programme for their interesting questions and helpful comments on the work presented for this project. Next, I want to mention Dr Thomas Hartley for his words of encouragement and professional advice throughout the past eight years of my academic journey.

I also thank my dear friends for their motivational words and wishes, especially those from the University of Wolverhampton, the University of Birmingham, and York. While I specifically tip my hat to my friends Lee Clift and Dan Herring, who have pursued PhDs in Computer Science and Engineering alongside me and have shared this experience.

Lastly, I would like to extend my heartfelt gratitude to my family. In particular, I thank my parents, Dawn and Paul, who have shown unshakable belief in my efforts; my wife, Irada, who has been a pillar of support and understanding; and finally, my mother-in-law, Dr Kamala Abrahimova, for her consistent words of encouragement and faith.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work.

SIGNED: [illegible] DATE: 28/12/2022

Material contained within this thesis has appeared in the following publications and artwork.

Publications

- Riley, J., (2021). Assured Multi-Agent Reinforcement Learning Using Quantitative Verification. In Doctoral Consortium on Agents and Artificial Intelligence (DCAART2021). Best PhD project award.

- Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A. (2021). Reinforcement Learning with Quantitative Verification for Assured Multi-Agent Policies. In Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, pp.237-245.

- Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A., (2021). Utilising Assured Multi-Agent Reinforcement Learning within Safety-Critical Scenarios. Procedia Computer Science, 192, pp.1061-1070.

- Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A., (2022). Assured Deep Multi-Agent Reinforcement Learning for Safe Robotic Systems. International Conference on Agents and Artificial Intelligence. Springer. pp.158-180.
- Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A., (2022). Assured Multi-agent Reinforcement Learning with Robust Agent-Interaction Adaptability. Intelligent Decision Technologies. Springer. pp.87-97.

Artwork

- Riley, J., 2019. Safe Multi-Agent Reinforcement Learning Towards the Engineering of Safe Robotic Teams. Cranfield Online Research Data (CORD).
- Riley, J., 2021. Assured Multi-Agent Reinforcement Learning. Cranfield Online Research Data (CORD).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous systems have become widely invested in, with consistent advances from worldwide research communities, showcasing developments in autonomous land, aerial, and marine-based systems, as well as different variants of robotic systems, from robotic manipulation to human-robot interaction. Many of the proposed uses of these autonomous systems find multiple autonomous agents (an agent being an artificial decision-maker) working together towards a shared goal. Such uses include teams of mobile robots reducing patient discomfort in hospitals, promoting safe travel behaviours between autonomous cars, aiding in search and rescue attempts, and providing assistance to soldiers and non-combatants in combat zones, amongst many other examples [63, 66, 74, 102, 118, 119]. However, while these systems are utilised to complete tasks in which they must reason and make decisions, they are also often limited by problem-based constraints such as time or hardware limitations. Furthermore, these systems often utilise learning-based artificial intelligence (AI) techniques due to their complex use circumstances. A system utilising AI learning techniques is generally described as a system that can learn rational behaviours, developing optimal behavioural strategies (policies) when given knowledge of a problem through suitable inputs [126].

Learning techniques are often utilised in single-agent and multi-agent systems due to the complex problems and environments in which these systems are expected to work. The reason behind their frequent use can be found primarily in the design of these systems. Learning techniques allow rational behaviours to be obtained without the need for hard-coded behaviours and designed functions that are often impractical for the designer to achieve due to the complexities behind these systems [126].

Learning techniques can also find the *optimal* sequence of behaviours subject to stakeholder-specified criteria, with a sequence of behaviours being a set of actions available to an agent over time. By making use of learning techniques, optimal system behaviours can be obtained with incomplete prior knowledge of the problem or environment.

These learning techniques, commonly deep reinforcement learning algorithms or traditional reinforcement learning algorithms such as Q-Learning, have been applied to and highly researched in regards to autonomous vehicles [18], game AI [131], robotics for security and defence [58], robotics in medical applications [140], and other scenarios where robotics can be used to remove humans from potential harm.

However, despite the potential of such learning techniques, there is a significant challenge in using them in real-world practical scenarios. This challenge is found in promoting trust in these systems by demonstrating that they can reliably behave safely during and after learning, avoiding adverse outcomes [46], explainable AI, a branch of research that attempts to allow machine learning to explain and justify its choices are an example of how trust is becoming a large concern with AI [89, 100]. The main issues arise due to the difficulties involved in expressing safety constraints as optimisation objectives, which are both strict enough to guide behaviour away from adverse outcomes and sufficiently relaxed to facilitate behaviours required for mission success. Single-agent safe reinforcement learning is an ongoing research area with many open problems, though promising approaches have been proposed and investigated [46, 55]. Despite advancements in single-agent safe reinforcement learning, multi-agent safe reinforcement learning is fast-growing yet relatively underdeveloped, with many areas requiring further investigation.

In response to these areas worthy of investigation, this thesis explores how recent advancements in single-agent safe reinforcement learning can be adapted and applied to multi-agent safe reinforcement learning to allow multiple agents to work collaboratively in shared environments, flexibly producing efficient behaviours and solutions while adhering to strict safety requirements.

1.1 Motivations of Safe Reinforcement Learning

AI-based learning, collectively known as *machine learning*, has been researched for several decades, with various forms of learning techniques being developed [136, 148]. This thesis focuses on a branch of machine learning techniques known as reinforcement learning [126], and more specifically, reinforcement learning involving multiple agents, known as multi-agent reinforcement learning (MARL) [99, 145].

Reinforcement learning [126] is an optimisation technique that consists of, as a minimum, an agent, an environment, and a goal. Reinforcement learning will use an agent to interact repeatedly with an environment by making decisions (taking actions) within the said environment. The agent will receive feedback on the utility of its decisions regarding the completion of its goal. With reinforcement learning derived from aspects of behavioural psychology, this feedback will be in the form of a reward or punishment, depending on whether the decision made is beneficial or detrimental to the agent. The agent retains knowledge of this feedback and then uses it to influence its future selection of actions. A reinforcement learning agent aims to obtain the largest cumulative reward possible and minimise cumulative punishment, i.e., to find the best possible sequence of actions to reach its goal. The methods in which these sequences of actions are found comprise a mixture of randomly selecting actions to discover more efficient options and choosing the actions with the highest amount of cumulative reward, with the agent being less inclined to choose random actions over time [16].

There are several decades of research focusing on the benefits and advancement of reinforcement learning and MARL, with both being widely implemented into games [133] and mobile robotics [62], amongst other examples [99]. There is practical potential for reinforcement learning and MARL, particularly with the advancements of deep reinforcement learning making use of neural networks, which greatly extends the practicality of reinforcement learning and explains the consistent growth of interest in this branch of machine learning [101].

However, there are also key challenges that prevent the wide adoption of both reinforcement learning and MARL in real-world settings. These challenges are particularly prevalent in mission-critical and safety-critical scenarios, where the safety of humans, sensitive equipment, or the agents themselves are as important or more important than the completion of goals [23, 46]. Within these scenarios, there will be safety requirements that the agent must meet to be deemed reliably 'safe'. However, even if an agent meets safety requirements, unpredictable behaviours can make these systems difficult to trust, further limiting their use [130].

In order to overcome these safety and trust issues in systems utilising reinforcement learning, a relatively novel branch of research has emerged, known as safe reinforcement learning [46, 55]. Safe reinforcement learning has multiple proposed directions for working with various systems and environments, though many of the current solutions lack formal assurances of safety and functional requirements being met [46]. Recently, the foundations of reinforcement learning, which utilises quantitative verification to assure that these safety and functional requirements will be met, have introduced a level of trust and have shown promise for future development [87].

Safe MARL is an even more recent addition to machine learning research and, as such, lacks the same level of development as safe reinforcement learning [55]. However, it shows potential to draw from the firmer foundations of safe reinforcement learning as it advances. As such, this thesis hypothesises that the limitations found in safe reinforcement learning that have recently been mitigated through formal verification techniques can also be mitigated within safe MARL. Specifically, by using quantitative verification [72], it can be assured that the behaviours of multiple reinforcement-learning agents working within a shared environment will meet strict safety requirements. Furthermore, it can also be assured in parallel that functional requirements will also be met. These assurances will allow autonomous systems of varying agent sizes and physical capabilities to employ learning techniques while also supplying the system stakeholders with confidence that safety and mission requirements will be met.

1.2 Thesis Contributions

This thesis introduces and details a two-stage, plugin-styled approach for safe MARL, introducing safety and performance assurances in the form of high-level quantitatively verified constraints. This approach is named Assured Multi-Agent Reinforcement Learning (AMARL). AMARL is introduced in Chapter 4, and is designed to be used with a broad range of tools, techniques, systems, and environments, allowing a standardised, flexible approach to safe MARL, which is limited in current research. Furthermore, AMARL offers a degree of trustworthiness in the stochastic autonomous learning systems that it is utilised on, either by providing formal proof of safety and functional requirements being met—or by showing that the desired requirements cannot be met and therefore indicating that the system should not be deployed.

The main contributions of the work supplied within this thesis are as follows:

- The AMARL approach is described with details and examples of how a multi-agent problem can be abstracted to create a high-level model and how safety and functional requirements can be specified for the said model. The thesis introduces methods that use these high-level models to synthesise safe abstract policies and leverage these safe abstract policies to constrain low-level actions.
- An extension to AMARL, named AMARL-Partial Policy Reuse (AMARL-PPR). AMARL-PPR allows the retrieval and reuse of partial policies that would otherwise not be useable due to previously utilised assured constraints that no longer hold. This offline-based extension reduces the learning time of future learning runs within similar domains by exploiting this previously learned knowledge.
- A two-stage extension to AMARL that is named AMARL-Adaptive Constraints (AMARL-AC). AMARL-AC detects high-level inaccuracies or changes within an environment compared to the abstracted Markov decision process created from the environment and then revises this abstracted model based on the ground truth, allowing new safe constraints to be produced while the learning continues.
- A new algorithm that takes as input a multi-dimensional matrix containing abstracted information about an MDP and outputs an abstract model of an environment encompassing multiple agents for navigation problems. A quantitative verification tool with a simplified input system can analyse this abstract model. This algorithm reduces the knowledge and experience needed to work with quantitative verification tools.
- An extensive evaluation of AMARL and its extensions, and three new case studies to aid in this evaluation. This evaluation showcases the benefits of a plugin-

styled approach in favour of the longevity of AMARL's relevance. The experiments presented in the thesis show AMARL being utilised in varying environments, differing system sizes, using learning algorithms from the main categories of MARL techniques, and with systems comprising both homogeneous and heterogeneous capabilities. Three case studies were developed to aid in this evaluation focusing on navigation-based problems, which are prevalent in reinforcement learning and MARL research. The first of these case study focuses on radiation avoidance in a ROS-based nuclear power plant patrolling simulator [110]. The second is a more detailed simulator focused on an infiltration problem in the Unity games engine and utilises the MARL plugin MLAgents [67]. This highly detailed simulator is increasingly used within the MARL research community. The third is a grid-based world built within the Unity games engine for navigation tasks, based on a previous safe reinforcement learning domain [87].

1.3 Thesis Structure

The remainder of the thesis is structured as follows.

Chapter 2 introduces the key components that underpin the AMARL approach. Firstly, Markov decision processes (MDPs), which represent the foundation of reinforcement learning processes, are introduced. Secondly, the concepts of reinforcement learning and MARL are established. Thirdly, quantitative verification and the tools which allow formal guarantees to be established are explored. Lastly, abstract Markov decision processes (AMDP), which allow high-level details of a problem environment to be analysed without the complexity of a full MDP, are introduced.

Chapter 3 offers an overview of related works in safe reinforcement learning, which underpin critical works in safe MARL. These works are categorised into methods that modify the exploration behaviours and methods that guide the optimisation behaviours of learning agents and facilitate comparisons with AMARL and its extensions.

Chapter 4 introduces our Assured Multi-Agent Reinforcement Learning, a multi-stage plugin-styled approach for safe multi-agent reinforcement learning using quantitatively verified constraints.

Chapter 5 introduces Assured Multi-Agent Reinforcement Learning with Partial Policy Reuse. This extension identifies similarities between domains and allows the partially learned knowledge to be reused in a new learning run.

Chapter 6 introduces Assured Multi-Agent Reinforcement Learning with Adaptable Constraints. This two-stage AMARL extension allows the automated re-constraint of agents during run time when the MARL system identifies inconsistencies between the AMDP and the problem domain.

Chapter 7 summarises the contributions of this thesis and the limitations of its contributions and proposes directions for future research development within the thesis.

Chapter 2

Background of Foundational Concepts

This chapter describes the key components underlying AMARL and its extensions, AMARL-PPR and AMARL-AC. These descriptions are necessary to facilitate a comprehensive understanding of the remaining contents of the thesis. Section 2.1 contains an introduction to Markov decision processes; these processes form the basis of reinforcement learning and multi-agent reinforcement learning techniques. Section 2.2 introduces abstracted Markov Decision Processes, a higher-level form of the Markov decision process used for efficient quantitative verification of their properties. Section 2.3 details the core elements of introductory reinforcement learning, which underpin multi-agent reinforcement learning. Section 2.4 contains the fundamentals of multi-agent reinforcement learning, which facilitates efficient problem-solving in distributed systems. Finally, 2.5 introduces quantitative verification, an established technique that assures safety and functionality through formal mathematical proofs. These fundamentals underpin the rest of the work within the thesis. Section 2.6 contains a summary of the key information detailed in this chapter.

2.1 Markov Decision Processes

The use of Markov decision processes (MDP) [44] is a fundamental method in representing a sequential decision-making problem, such as the ones which are often solved using a form of reinforcement learning. As such, MDPs are utilised in chapters 4, 5, and 6, within the AMARL approach and its extensions.

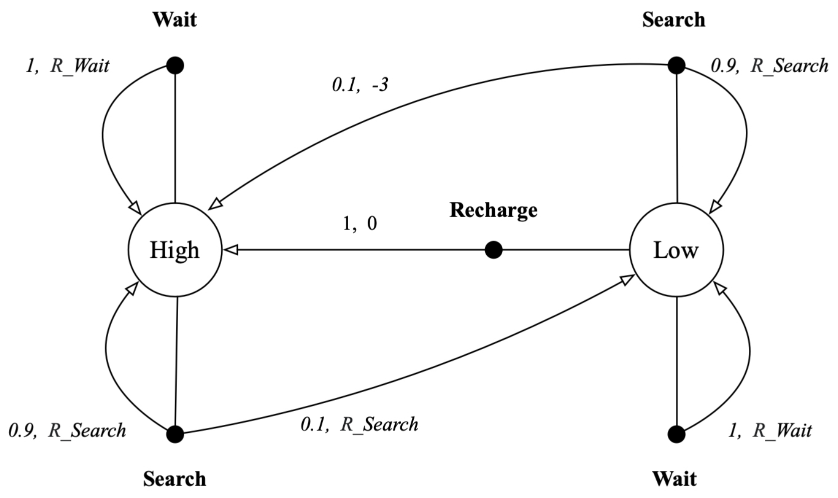


Figure 2.1: Transition Graph example of an MDP for a recycling robot system. Adapted from [109].

A Markov decision process, otherwise known as a controlled Markov chain, is a mathematical formalisation for describing and capturing information about systems that contain stochastic processes. MDPs capture a set of states that describe the potential situations an agent could be within, actions that allow the agent to transition between states and a reward associated with taking said action within a state. Furthermore, with MDPs being utilised for stochastic processes, the probabilities of reaching a state s' when taking action a in the initial state s and receiving a set reward r by doing so are also described.

The purpose of the MDP is to solve sequential decision problems under uncertainty. So the performance of an agent's controller within the MDP must be evaluated to guide the agent's controller to more efficient behaviours. An evaluation can be achieved by observing the reward obtained by the agent whilst navigating through the MDP. This reward signals how effective or ineffective an action is towards meeting an objective, with ineffective actions being given negative rewards, otherwise known as punishments. Therefore, the MDP is solved once a controller is obtained, allowing behaviour that receives the greatest possible positive reward. This behaviour, which involves actions taken in each state over time, is known collectively as a policy.

To help illustrate the MDP framework, Figure 2.1 is presented, which shows the transition graph of a recycling robot that searches for and collects empty bottles. The system has two states, represented by white circles, which relate to the remaining battery that the robot has, which can either be high or low. When the robot's battery is in the state named High, it has two actions: go searching for bottles, or wait for people to bring them to it. When the robot's battery is in the state named Low, it has three actions: search for bottles, wait for bottles, or recharge at a recharging station. These actions are shown by small black circles that protrude from their relative states.

This graph includes a transition function, the probability of an action leading to the next state, and the robot's expected reward for reaching it. For example, suppose the robot is in a low battery state and decides to search for more bottles. In that case, it has a probability of 0.9 of returning to the low battery state and receiving the expected reward from finding bottles. However, there is a probability of 0.1 that the robot will deplete its battery entirely and must be rescued and taken to a recharging station. This transition gives the agent a reward of negative three, as the robot being rescued is far from an optimal outcome.

The following definition of an MDP is adopted [111].

Definition 2.1 (Markov Decision Process)**.**

A Markov decision process is formally described as a tuple (S, A, T, R) where,

- S is a finite set of states.
- A is a finite set of actions.
- $T : S \times A \times S \rightarrow [0, 1]$ is a state transition function, where for all $s, s' \in S$ with any available action $a \in A$ in state s, $T(s, a, s')$ gives the probability of action a in state s resulting in a transition to state s'.
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a reward function where $R(s, a, s') = r$ is the reward received from transition from state s to state s' after taking action a.

The Markovian principle [52] is the governing law that underpins all Markov chains and MDPs. The principle states that all Markovian processes are memoryless, meaning that an action a chosen in-state s, which causes the transition to state s', is chosen with no regard to what occurred in previous states. The governing law is shown in Equation 2.1, where $f(s, s')$ is the probability of the next state being s' given the current state s, where t represents the unit of time. When comparing Equation 2.1 to a non-Markovian governing rule Equation 2.2, it is visible that the Markovian governing rule lacks any reference to states other than s and s'.

$$Prob\{S(t+1) = s'|S(t) = s\} = f(s, s') \tag{2.1}$$

$$Prob\{S(t+1) = s'|S(t) = s, S(t-1) = s_{t-1}\} = f(s, s_{t-1}, s') \tag{2.2}$$

A policy, being the control mechanism that directs the agent within the MDP, is an n-tuple where each element corresponds to an action that should be selected in a specific state. For example, denoting a policy as π, the ith element of π, being $\pi(i)$, is the action that would be selected in the ith state when following policy π. There are two forms of policy, deterministic and stochastic; throughout this thesis, the word "policy" will refer to a deterministic policy. A deterministic policy assigns only one action for each state which must be chosen as directed by an action-selection policy defined as $\pi : S \rightarrow A$; these actions will not change over time.

An *optimal* policy, denoted as π^*, is called optimal as it results in the highest possible cumulative reward. This means that the choices made in all states will be the most beneficial to reaching the system's objectives.

Solving an MDP, which entails finding a policy that returns the highest possible cumulative reward, is achieved using value functions [44]. A value function returns the utility of state s when considering the future states at s_{t+1} that can be accessed from a followed policy and continuing in this fashion. In contrast, a reward function returns the immediate utility/reward of transitioning to a state without recognising future states.

A state value function enables a perceived utility to be assigned to an agent transitioning to a state based on the expected cumulative reward when following a policy such that $V^\pi : S \rightarrow \mathbb{R}$. A policy and a relevant performance criterion are two components of a value function. The performance criterion is used to search for policies that return the optimal sequence of immediate rewards and allow the formation of a value function. Depending on the criteria selected, the value function will be shaped accordingly; the finite criterion can be used in an episodic scenario, also known as a finite horizon problem.

$$E[r_0 + r_1 + r_2 + ... + r_{N-1}|s_0] \tag{2.3}$$

This enables all rewards r_n at each time step to be obtained according to a policy π starting at an initial state s_0. However, a discount criterion is often used when faced with a large state space that may or may not be infinite, otherwise known as an infinite horizon problem.

$$E[\gamma r_0 + \gamma^1 r_1 + \gamma^2 r_2 + ... + \gamma^i r_i|s_0] \tag{2.4}$$

Here a discount factor γ, which is $\gamma \in [0,1]$, is used to emphasise rewards obtained from states that are transitioned to in later time steps. With γ degrading over each time step, the transitions taken earlier will be perceived to be more valuable than those in later time steps; this is due to the general principle that rewards that are more readily accessible are more important in the immediate situation than those further away in time. A smaller discount value will associate more value with immediate rewards, while a larger discount value will value distant rewards more. With a discount value of 1, Criterion 2.4 reduces down to Criterion 2.3

Value functions can also be incorporated with a criterion, such as the finite criterion in Equation 2.5 and the discounted criterion in Equation 2.6. Where $V^\pi(s)$ is the utility of policy π from state s, and r_{t+i} is the reward obtained at time step $t+i$.

$$V^\pi(s) = \sum_{i=0}^{\infty} r_{t+1}|s_t = s \tag{2.5}$$

$$V^\pi(s) = \sum_{i=0}^{\infty} \gamma^i r_{t+1}|s_t = s \tag{2.6}$$

Similar to state value functions are state-action value functions, otherwise known as Q-functions. These differ from state value functions by determining the utility value of taking action a in-state s while following a policy; this value is commonly known as a Q-Value and is as follows $Q^\pi : S \times A \rightarrow \mathbb{R}$. Q-functions can be utilised within infinite horizon problems as shown in Equation 2.7, and infinite horizon problems shown in Equation 2.8.

$$Q^\pi(s,a) = \sum_{i=0}^{\infty} r_{t+1}|s_t = s, a_t = a \tag{2.7}$$

$$Q^{\pi}(s,a) = \sum_{i=0}^{\infty} \gamma^{i} r_{t+1} | s_t = s, a_t = a \tag{2.8}$$

With value functions, it is possible to use dynamic programming algorithms [125] to solve MDPs in which all dynamics are fully known. Where the dynamics of the reward functions or transition functions are unknown or partially unknown, reinforcement learning algorithms [126] can be used to learn these dynamics and solve the partially unknown MDP gradually.

2.2 Abstract Markov Decision Process

In modelling the sequential decision-making problems using MDPs, the consideration of all the possible states that a system may encounter is required. For complex environments in which there are many combinatorial effects, the number of states grows exponentially. This leads to problems of scaling computational effort. In order to tackle this problem, it is common to use abstraction to remove layers of complexity and reduce the size of the state space. By abstracting the MDP, it is possible to retain important information while making the model easier to work with. The outcome of this abstraction process is known as an abstract MDP (AMDP) and is used within chapters 4, 5, and 6.

At a foundational level, many learning and planning algorithms use MDPs; while this use is widely accepted, it leads to severe limitations of use in terms of larger, complex, real-world problems [77]. This limitation is brought from a well-known problem known as the state-space explosion problem [34]; this occurs when the introduction of larger environments and problems causes exponential growth of the state-space, to which MDPs are prone. This state-space explosion problem makes working with MDPs very challenging, especially with tools that use exhaustive processes, such as quantitative verification [34], making these tools impractically computationally expensive. Lastly, to use quantitative verification on an MDP, full knowledge of the MDP must be held. This requirement is often not feasible, as shown by techniques used to solve MDPs with unknown properties, such as reinforcement learning being developed.

In order to subvert these issues, *abstract* MDPs (AMDP) are commonly used in safety engineering [51, 77]. An AMDP is an MDP that has been reduced in size through

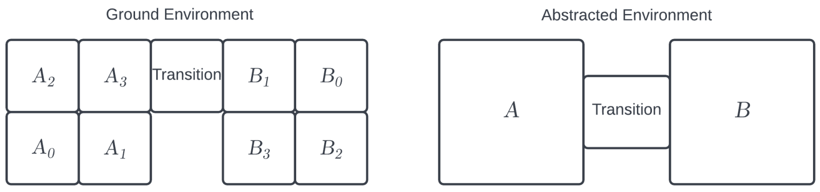


Figure 2.2: Ground Environment and Abstracted Environment of Two Rooms.

a chosen state aggregation method, of which there are many [77]. This state aggregation and what are known as *options* [127], which are high-level closed-loop policies of actions over time, allows for significantly reduced state spaces and the construction of MDPs with limited knowledge. State-aggregation and *options* allow greater state-space reduction due to the abstraction of multiple actions into a simple definition. For example, Figure 2.2 shows a ground view of an environment with two rooms and an abstracted view of this same environment. If an agent must travel from Room A to Room B, in the ground view, the MDP must consider each transition in each state which will consist of moving North, South, and potentially moving East or West. While in the abstracted view, this can be refined to one transition, which could be defined as A *transitionTo* B.

Definition 2.2. Abstract Markov Decision Process

An AMDP is formally defined as a tuple $< \overline{S}, \overline{A}, \overline{T}, \overline{R} >$ where,

- $\overline{S} = \overline{s}(S)$,
- $\overline{A} = \overline{a}(A)$,
- $\overline{T}(\overline{s}, \overline{a}, \overline{s}^{'}) = \sum_{s \in \overline{s}} w_s \sum_{s^{'} \in \overline{s}^{'}} T(s, \overline{a}, s^{'})$,
- $\overline{R}(\overline{s}, \overline{a}) = \sum_{s \in \overline{s}} w_s | R(s, \overline{a})$,

With $\overline{s}(S)$ being an abstraction function of the state space defined by S, and w_s being a weight of state s which is associated with the frequency of this state in the abstracted state [84].

AMDPs remain relevant when considering reinforcement learning and multi-agent systems and have been used for these exact purposes [127]. Regarding this thesis, AMDPs are utilised to facilitate efficient quantitative verification of models in chapters 4, 5, and 6.

2.3 Reinforcement Learning

Reinforcement learning [126] is a sub-area of machine learning used to find optimal solutions for MDPs when the reward and transition functions are unknown. Researchers have focused on reinforcement learning for decades, with multiple forms of reinforcement learning being devised and ending with a plethora of learning algorithms, policy convergence optimisation techniques, and agent exploration techniques. For simplicity and clarity, the rest of this section will only focus on what has become the standard description of reinforcement learning and the most significant learning method in the development of reinforcement learning for AI agents, known as temporal difference learning. A comprehensive view of the primary reinforcement learning methods can be viewed in recently updated sources [4, 126].

There are two main components when discussing reinforcement learning, and these combine to make what is known as an agent-environment interface. First, an agent is a learner and decision-maker in the scenario, and an agent can take many forms. A mobile robot or a non-player character in a video game are two clear examples. The environment generally refers to everything outside of the agent; what constitutes outside of an agent depends on the scenario. However, the environment can commonly contain the rooms or surroundings through which an agent has to transition and the physical components of an agent, such as motors and sensors. The agent-environment interface continuously interacts throughout the learning process, with the agent making decisions and performing actions within the environment and the environment responding to these actions and presenting new situations to said, agent. In terms of an MDP, the agent, when in an initial state, will have action choices that correspond to the transitions present in said state. As the agent transitions, governed by transition probabilities, the environment will present a state to the agent. As a state is presented to the agent, a reward or punishment is given from the environment depending on the state's perceived value. These rewards are numerical, and the agent attempts to maximise these rewards over time.

As the agent moves around the MDP, the reward received from performing action

Algorithm 1 ϵ-greedy action selection [126]

```
1: $n \leftarrow$ random uniform number between 0 and 1;
2: if $n < \epsilon$ then
3:     $A \leftarrow$ random action from the action space;
4: else
5:     $A \leftarrow \max Q(s,.)$;
6: end if
7: return selected action $A$;
```

a in state s is stored as Q-values in a tabular structure known as a Q-table. Q-tables allow reward values to be accessed and manipulated with the use of state action pairs $Q(s,a)$. The way in which the agent chooses to move around the MDP is known as an action selection policy, which directs which actions will be selected in a state. The action selection policy can take many forms; the most commonly introduced is known as the ϵ-greedy approach. The pseudo-code depicting this can be seen in Algorithm 1. The ϵ-greedy approach has two outcomes in terms of action selection behaviour, the first involving taking the action which will result in the highest reward (greedy) and the second involving taking random actions irrespective of reward. The frequency with which these two outcomes happen depends on the value of ϵ, which is between 0 and 1. Action selection policies are ways of answering one of the foundational problems in reinforcement learning, this problem being known as the exploration/exploitation problem. As a learning agent makes decisions within the environment, it can exploit the knowledge of the environment it already has to take what is currently the most efficient action (exploitation) or take actions that are less known or unexplored to gain more information about the environment, potentially finding more efficient options (exploration).

In episodic reinforcement learning, which the rest of this thesis is concerned with, there is a clear start, and end position to a problem and one instance of the problem is called a learning episode. In episodic learning, the agent's goal is to reach a terminal state, which means the problem has been solved. As the agent explores and updates its Q-values, these updates will correspond to how much the action benefits the agent in reaching this terminal state. When this terminal state has been reached,

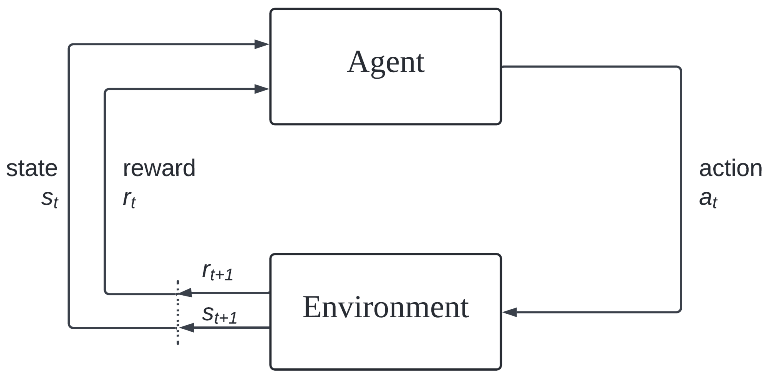


Figure 2.3: Reinforcement Learning Agent-Environment Interface. Adapted from [126].

the environment and the agent are reset to the initial settings. This start-stop episodic setting is ideal for reinforcement learning because the learning process requires repeat interaction with states and actions to find an optimal/near-optimal solution. Due to this recurring nature of reinforcement learning and related time requirements, it is common for learning to be halted at a predefined episode number. It can take thousands of learning runs to find solutions to relatively simple problems. Therefore sub-optimal solutions are often declared as efficient enough to halt learning in efforts to reduce time-based costs.

Temporal Difference update algorithms are one of the most common forms of update algorithms. These algorithms utilise Monte-Carlo sampling and combine this with the Bellman equation to update Q-values each time an action is taken within a state, allowing knowledge to be gained throughout the state space. Arguably the most well-known temporal difference algorithm is the Q-learning algorithm [138].

2.3.1 Q-learning

Q-learning is an off-policy temporal difference learning algorithm meaning it deviates from the current policy it possesses to find more efficient actions. Q-learning utilises the ϵ-greedy approach to make these decisions regarding exploration and exploitation, with the ϵ value degrading steadily over the episodes until it reaches 0, in which case the decisions are entirely greedy. It utilises a Q-table and updates this table based on the difference between the current value of a state-action pair and the maximum

Algorithm 2 Q-learning update algorithm. [126, 138]

```
1: for each episode do
2:   Initialise state $s$
3:   while $s$ is not terminal do
4:     Select action $a$ for state $s$ using action selection policy
5:     Perform a and observe reward $r$ and new state $s_{t+1}$
6:     $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
7:     $s \leftarrow s_{t+1}$
8:   end while
9: end for
```

expected reward from the potential future state. The values in the Q-table are updated with each transition through the MDP that the agent makes depending on the selected state-action pair. The Q-values are updated using the Q-learning update function, which can be seen in Equation 2.9. Where $Q(s_t, a_t)$ is the Q-value of the state-action pair at time t, $\alpha \in [0, 1]$ is a learning rate and discounts the learning process to avoid shortsightedness in value updates, $\gamma \in [0, 1]$ discounts the importance of the future expected reward in comparison to the immediate reward of making a state transition, and r_{t+1} is the immediate reward from moving to state s' from state s using action a.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.9}$$

The Q-learning algorithm, depicted in Algorithm 2, showcases how this update takes place at every transition that occurs within the MDP and, as proven theoretically, with infinite time and a decaying learning rate, will converge to an optimal policy.

2.3.2 Deep Q-Learning

As the complexity of environments increases and the process of creating appropriate state-space models becomes unpractical for traditional reinforcement learning techniques, it has become commonplace to employ deep reinforcement learning techniques, a combination of traditional reinforcement learning and artificial neural networks [80].

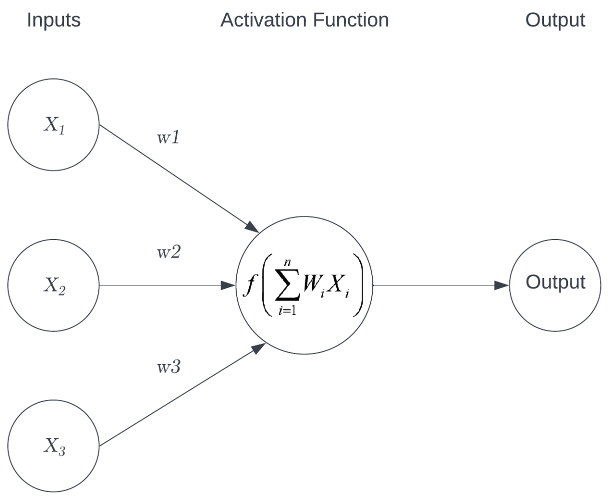


Figure 2.4: Information Processing within an Individual Neuron. Adapted from [19].

Neural networks are inspired by the interconnection of neurons in a biological brain and consist of many simple computational units connected into layered structures to provide complex mappings of inputs to outputs. These simple computational units are called artificial neurons, transforming one or more inputs into a single output value. A single process that takes place within a neuron can be seen in Figure 2.4, where the inputs are a vector of x_n values, which are then multiplied by a weight, such as w_i which adjusts the strength of the inputs. These adjusted input values are summed together and passed to an activation function. An activation function is utilised to determine what future neurons are inactivated, partly activated, or fully activated. It does this by converting, for example, the input values to a value between 0 and 1. With 0 being a signal not to activate, 1 being a signal to activate as strongly as possible, and varying values of strength in between. The signal gained from the activation function can then be passed out of the neuron to either another neuron or as a final output. There are several forms of activation function, each with their own strengths, weaknesses, and preferred usage, with a neural networks prediction accuracy being determined by the activation function chosen, such as binary step functions, Sigmoid functions, and softmax functions [122].

The general structure of a neural network, as seen in Figure 2.5, where each node

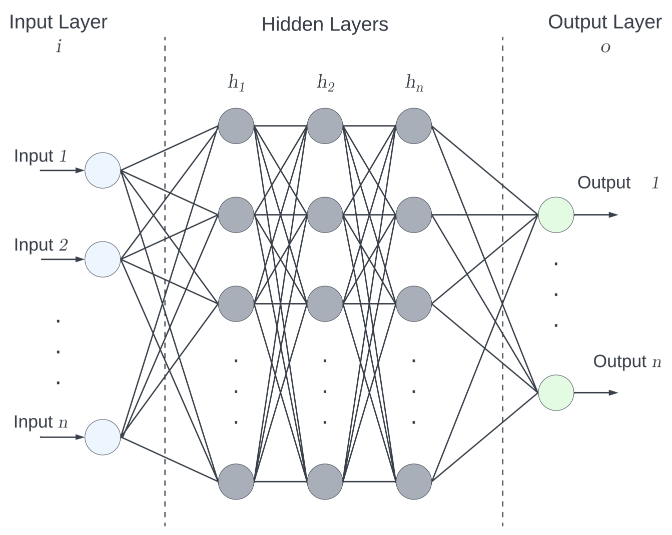


Figure 2.5: Neural Network [19].

contains the process described above, consists of three types of nodes. Input nodes are nodes that take in the raw input and send an activation signal to the hidden nodes, hidden nodes take in the signal from the input layer and pass their output to either another hidden node or the output node, and the output node takes the output of the last hidden node as input and produces a prediction based on the input data. The signals sent from nodes to other nodes are adjusted based on the importance of this transition in the network, which is determined by the value of the corresponding weights, originally arbitrarily set.

The process of training a neural network is an iterative cycle of forward propagation and backward propagation. First, a set of input variables are passed into the network, allowing the network to produce a prediction based on the initial values of the weights; this is known as forward propagation. The amount of error between the network's prediction and the expected result is determined using this measurement. Next, the weights are adjusted to minimise errors by proceeding backwards through the network. This process is known as backward propagation. After the weights have been adjusted, the process is repeated until the network makes reliably accurate predictions.

While it is possible to use traditional reinforcement learning techniques in complex environments, it can be challenging to incorporate them efficiently. For example, Q-

Learning requires Q-tables with values for each state and action pairing, which can become numerous in complex environments. Neural networks, on the other hand, do not require such a cumbersome data structure. With such a benefit of neural networks being established, it is clear to see the advantages of combining neural networks with Q-learning, known as Deep Q-learning [60]. In the Deep Q-learning function, an approximation is used to estimate the optimal Q-values; this differs from standard Q-learning and its value iteration method. Deep Q-learning consists of a neural network with multiple hidden layers; the inputs to the neural network are the states that an agent is in, and the outputs will be the predicted Q-values of each action that's available in said state. Two neural networks are required for deep Q-learning to converge: a training network and a target network. The training network functions as a normal neural network does, adjusting its weights after each iteration. A target network, however, is only updated every n iteration. The reason for this delay is to bring stability to the learning process, the error of the predictions made by the training network is compared to the values of the target network, and the weights are updated based on this error. The target network is updated with the same weights as the training network, but the delay in updating negates the effect of randomness on the network. The final feature of Deep Q-learning is using a batch-replay buffer, which stores data from previous encounters that can be drawn for training; training the network on a batch of experiences can identify patterns and avoid overfitting to a specific experience.

The way in which target values are formulated is through the temporal difference update function seen in Equation 2.10; where r_t is the value of the state transitioned to, and $Q(s_{t+1}, a)$ is received by feeding the current state into the target network.

$$\text{Target}(s, a) = r_t + \gamma \max_a Q(s_{t+1,a}) \tag{2.10}$$

The use of deep learning, such as deep Q-learning, has allowed reinforcement learning to be used highly efficiently to solve complex problems where traditional reinforcement learning would be impractical or even impossible. One such problem in which traditional reinforcement learning can be used in simple environments but suffers from the extreme state-space explosion are problems which involve multiple agents operating within the same environment, either cooperatively or competitively.

Reinforcement learning is a core component of this thesis and can be seen throughout. In order to evaluate AMARL and its extensions, forms of Q-learning are utilised in chapters 4, 5, and 6, and neural networks are employed in chapters 4 and 6.

2.4 Multi-Agent Reinforcement Learning

A multi-agent system (MAS) [139] is known as a collection of autonomous agents that operate in a shared environment, with the ability to make observations of the environment and take actions that influence said environment. While it is possible to preprogram behaviour for these agents, they often must learn behaviours accustomed to the complex and sometimes changing environments they inhabit. This is achieved using multi-agent reinforcement learning (MARL) [24]. The benefits of MARL are paramount due to its distributed nature allowing it to succeed in a myriad of problems with many tasks that need to be completed concurrently and with increased robustness. Also, the potential applications of MARL are sizeable, with many real-world problems being able to be framed as multi-agent problems, which would be too complex to preprogram.

Due to the diversity of use of MAS and MARL, key features have been distinguished to identify different kinds of MAS problems [145]. The *first feature* identifies the relationship of the MAS, whether the system is purely cooperative and working towards a shared goal, purely competitive with conflicting goals, or whether the system contains a mixture of cooperative and competitive goals. The remainder of the work in this thesis is focused solely on cooperative systems, and as such, any mention of a MAS or MARL system should be identified as cooperative. The *second feature* identifies the similarity and unity between agents' functions and abilities, with homogeneous systems containing identical agents and heterogeneous containing one or more agents with different functions or abilities. This thesis contains work with both homogeneous and heterogeneous systems and will state when each is being utilised. The *third feature* identifies how task distributions are handled within the system, one of these being centralised, where a single agent makes decisions for all agents in the system, and decentralised, where each agent is responsible for its own task assignment, as well as many other variants of these two methods. This thesis uses decentralised systems in terms of task assignment, but in all cases where communication is needed, it is assumed that communication is reliable.

Similarly to single-agent reinforcement learning, MARL is used to solve sequential decision-making problems but with two or more agents. In these multi-agent sequential decision-making problems, the current system state and the current system's rewards are directly influenced by the actions of all other agents within the system. One of the simplest ways to view a multi-agent sequential decision-making problem is to have each agent ignore the presence of all other agents and simply view the agent's interactions as part of the system, meaning the agent only makes observations of its own received rewards. Here the multiple agents interacting with the environment make the problem non-Moravian from the single agent's point of view. In this setting, there are no formal proofs of converging to an optimal set of agent policies [128], though practically independent learners are thought to be more applicable in a board sense and have shown to learn effective behaviours [88]. Due to these non-stationary issues in independent learning, frameworks have been created to allow MARL to take place with theoretical proof of convergence. One such framework which facilitates cooperation is known as a Markov/stochastic game [79].

Definition 2.3.

2.4.1 Markov Games

A Markov game captures the interactions of multiple agents and allows the observation of all agent rewards at each time step in the system. These games have been investigated for decades, with several algorithms developed for MARL within Markov Games [145]. A Markov game has a similar relationship dynamic with the system as an MDP, as seen in Figure 2.6, where an agent performs an action within a system and is then presented with a new state and a reward for the transition. However, in a Markov game, all actions from each agent are used to determine what new state is presented to the agents. As each state within the stochastic system requires actions to be selected from each agent, every state transition within a stochastic system results in a new Markov game being presented to the agents. The agents then *play* this game and are presented with a new system state depending on the agents' actions. This process is commonly known as *repeated games*, as the games are played repeatedly as the agents change the state of the system, though the games played change depending on

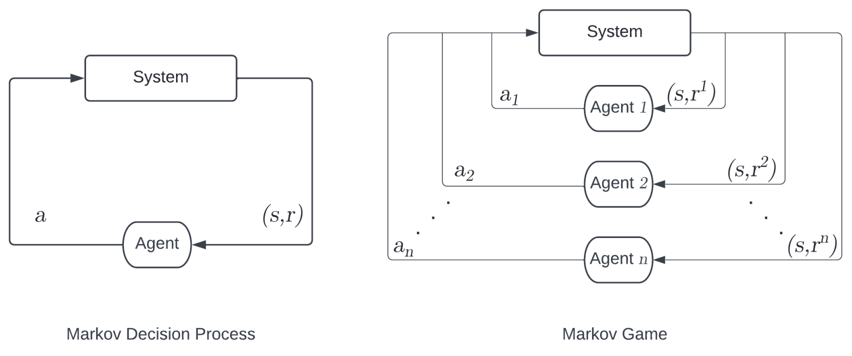


Figure 2.6: Description of the System-Agent Relationship of an MDP and a Markov Game. Adapted from [145].

this state. One goal of the agents within a Markov game is to maximise their expected reward from taking action in the Markov game. In a collaborative game, agents aim to find a Nash-equilibrium [137], where no player has any incentive to change their chosen action, meaning no agent can receive more reward given the actions other agents have taken. Another goal is to find a Pareto-optimal [28] set of actions where no agent can improve its expected reward without decreasing the reward of another agent within the game. These two goals determine the efficiency and completeness of which a game has been solved.

When formally defining a Markov game [22], it is defined as a tuple $< S, N, A, T, R >$ where

- S is a finite set of states,
- N is a finite set of n players/agents,
- $A = A_1 \times ... \times A_n$, where A_i is a finite set of actions available to player i,
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function where $p(q, a, \widehat{q})$ is the probability of transitioning from state q to state $\widehat{q}$ after joint action a and,
- $R = r_1 ... r_n : S \times A \rightarrow \mathbb{R}$ is a reward function for player i.

Varying methods of MARL are utilised throughout this thesis. Chapters 4, 5, and 6 show a multi-agent system utilising independent learning agents. While in chapter 4 a multi-agent system that uses a learning technique that utilises Markov games can be found.

2.5 Quantitative Verification

Due to probability impacting all real-world problems and systems, methods of analysing the stochasticity within these systems have been developed. One such method for stochastic model checking is known as quantitative verification [72]. However, due to the problems that this thesis addresses being of a stochastic nature, the discussion of model-checking techniques will be limited to stochastic model checking, also known as probabilistic model checking. Quantitative verification is a core component of AMARL and its extensions and, as such, features heavily in chapters 4, 5, and 6.

Quantitative verification is a formal verification technique used to verify a mathematical model that captures the behaviours of systems in the form of Markov chains, MDPS, and Markov games. This formal verification technique aims to verify a mathematical model's correctness, reliability, optimality, safety, and other non-functional properties. Some quantitative properties include the probability of the system reaching a terminal state, the cost of performing a behaviour within the system, and how much time would be required for this behaviour to be completed. These properties are formally defined using probabilistic temporal logic and, along with an automated model checker, can efficiently be used to analyse the model of the system.

Quantitative verification uses exhaustive analysis of the model's state space; while this can be a computationally expensive process, it guarantees that the verification results will be accurate. Due to the non-determinism of MDPs, the quantitative verification must resolve this by generating policies through the MDP, also known as adversaries [43], which allows the MDP to be analysed due to the degree of reachability each adversary has, meaning if all adversaries reach a state, then it has a probability of 1 of occurring, while if only half of the adversaries reach this state, it will have a probability of 0.5 of occurring. Quantitative verification has been used in a wide range of problem areas and to a high degree of effectiveness, including unmanned vehicles [50], cloud infrastructure [27], and in single-agent safe reinforcement learning [87].

2.5.1 Probabilistic Computation Tree Logic

As described, quantitative properties are described using probabilistic temporal logic, which allows complex properties to be described, such as ‘the probability that a system will reach the goal state must be at least 0.90’ or ‘what is the total cost of proceeding to the goal state’. In order to verify quantitative properties, probabilistic computation tree logic was developed (PCTL) [43], which is an extension of computation tree logic (CTL) [33].

Part of PCTL and CTL success is in the intuitiveness of its use, building from temporal logic operators [34], which allow differing properties to be analysed. Two operators that focus on the paths within a computation tree are the A operator, which will allow us to determine if a property holds along all paths within the computation tree, and the E operator, which will allow us to determine if a property holds along some of the paths within the tree. However, there are also operators which allow us to determine if certain properties hold within the states along the paths of the tree. For example, the X operator (next time) allows us to determine if a property holds within the second stage of the path. The F operator (future) will allow us to determine if a property will hold at some state in the path, the G operator (global) allows us to determine if a property holds over all states on a path, and U (until) allows us to determine if a property holds until a state on a path given that another property holds on all subsequent states in said path. In addition, PCTL adds a third path operator, the P operator (probabilistic), which allows us to determine the probability of the system evolution occurring in a specific way. Finally, PCTL also allows using the R operator (reward) to analyse the rewards in an MDP and how they develop through a path.

As can be seen, there are two distinct forms of operators, those detailing the path within the tree and those detailing the states on those paths. These relate to the formulas, which can be seen below in the PCTL syntax.

Definition 2.4. PCTL syntax.

The syntax of PCTL is as follows, where Φ is a state formulae, and ϕ is a path formulae:

$$\Phi ::= \text{ true } \mid \text{ a } \in \text{ AP } \mid \Phi \wedge \Phi \mid \neg \Phi \mid \text{ P } \bowtie_p (\phi)$$
$$\phi ::= \text{ X } \Phi \mid \Phi \cup^{\leq k} \Phi \mid \Phi \cup \Phi$$

Where AP is a set of statements that are either true or false in each state of the MDP, known as an atomic proposition, $\bowtie \in \{<, \leq, \geq, >\}$, $p\ in$ $[0,1]$, $k \in \mathbb{N}$, and the reward state formulae which allows the use of the R operator, is defined as:

$$\phi ::= \text{R}_{\bowtie r}[\text{I}^{=k}] \mid \text{R}_{\bowtie r}[\text{C}^{\leq k}] \mid \text{R}_{\bowtie r}[\text{F}\Phi]$$

Where $r \in \mathbb{R}_{\geq 0}$, I is the instantaneous reward which is the expected reward at step k under the constraints of $\bowtie_r$, C is the cumulative reward which is the expected cumulative reward until step k when following path ω under the constraints of $\bowtie_r$, and F is the expected cumulative reward before reaching a state that satisfies Φ, which is also under the constraints of $\bowtie_r$. PCTL also have the following semantics adapted from [86].

Definition 2.5. PCTL Semantics.

The semantics are defined using the operator $\models$, which is a relational satisfaction operator. For example, if M is a Markov model and state s satisfies the formula Φ then $M, s \models \Phi$ is equal to true. With this, we can define the satisfaction relations as shown.

$$M, s \models true \text{ for all } s \in S$$
$$M, s \models a \text{ iff } a \in L(s)$$
$$M, s \models \neg\phi \text{ iff } M, s \not\models \phi$$
$$M, s \models \phi_1 \wedge \phi_2 \text{ iff } M, s \models \phi_1, \text{ and } M, s \models \phi 2$$
$$M, s \models \phi_1 \vee \phi_2 \text{ iff } M, s \models \phi_1, \text{ or } M, s \models \phi 2$$
$$M, s \models \phi_1 \rightarrow \phi_2 \text{ iff } M, s \models \phi_2, \text{ whenever } M, s \models \phi 1$$

Each state is given a set of satisfied atomic propositions through the labelling function $L(s)$. Also, when defining PCTL operator semantics, the above notation holds with the addition of $\omega \in \text{Path}(s)$ representing the infinite state paths.

$$M, s \models \text{A}\phi \text{ iff } \omega \models \text{ for all } \omega \in \text{ Path}(s)$$

$$M, s \models \text{E}\phi \text{ iff } \omega \models \text{ for some } \omega \in \text{ Path}(s)$$

$$M, s \models \text{P}_{\rhd\lhd p}[\phi] \text{ iff } Pr_s\{\omega \in \text{ Path}(s) | \omega \models \phi\} \rhd\lhd \text{ p}$$

$$M, \omega \models \text{X}\Phi \text{ iff } \omega(1) \models \Phi$$

$$M, \omega \models \text{F}\Phi \text{ iff } \exists k \geq 0 \text{ such that } \omega(k) \models \Phi$$

$$M, \omega \models \text{G}\Phi \text{ iff } \forall i \geq 0 \text{ such that } \omega(i) \models \Phi$$

$$M, \Phi_1 \text{ U } \Phi_2 \text{ iff } \exists k \geq 0 \text{ such that } \omega(k) \models \Phi_2 \text{ and } \forall i < K.\omega(i) \models \Phi_1$$

Here the probability of ϕ being satisfied is denoted by Pr_s, with $\omega(iandk)$ denoting a state within path ω.

Using the grammar that PCTL facilitates, it is possible to define properties and verify that constraints hold through a model by translating natural language requirements into PCTL statements. Examples of PCTL statements with their corresponding natural language form can be seen in table 2.1.

Table 2.1: PCTL Statements of Natural Language Properties

Natural language Description	**PCTL Statement**
Probability that more than 5 errors occur is less than 0.1	P<0.1 [F Errors > 5]
What is the expected cumulative amount of battery used in a mobile system?	R=? [C]
The maximum probability that more than ten messages have been lost by time T	Pmax=? [F<=T Lost > 10]

2.5.2 Probabilistic Model Checkers

Due to the complexity of models, it is not possible to utilise manual analysis, and simulation-based model checking struggles with stochastic and non-deterministic models due to the possibility that some computational paths may never be explored and assessed [12]. Therefore, it is necessary to make use of model-checking tools that are capable of analysing stochastic models. Probabilistic model checkers, also known as stochastic model checkers, have been developed to analyse differing stochastic model types and show proficiency under differing scenarios. Furthermore, such model checkers efficiently perform these analysis processes and allow the validity of properties to be formally assured.

Though many model checkers exist and comparisons between these models have been made [3, 65], only two known probabilistic model checkers are relevant to the work presented in this thesis, and this is due to the forms of stochastic models that these tools are capable of analysing, in particular MDPs. PRISM and Storm are the two model checkers known to analyse MDPs and be widely used.

- PRISM [73], which stands for probabilistic symbolic model-checker, is a tool developed jointly by the University of Birmingham and Oxford University as a flexible tool for analysing different stochastic models. Namely discrete-time Markov chains, continuous-time Markov chains, MDPs, probabilistic automata, probabilistic timed automata, partially observable MDPS, and partially observable probabilistic timed automata. PRISM supports several engines which allow stochastic models to be analysed in various ways. Such as the faster *sparse* engine and the more memory-efficient *hybrid* engine. Though it also supports multiple forms of temporal logic, including LTL, CSL, and PCTL. Lastly, PRISM is accessed from a graphical user interface and the command line. It allows models to be described using the PRISM language, which is introduced further within this chapter.

- Storm [36] is a modular model checker developed at RWTH Aachen University to easily allow extensions. Like PRISM, Storm uses numerical and symbolic computations, though unlike PRISM is not capable of supporting discrete event simulation, otherwise known as statistical model checking. However, it can analyse discrete-time Markov chains, continuous-time Markov chains, MDPs, and Markov automata and supports two forms of temporal logic, CSL and PCTL. Storm can be accessed using a C++ and Python API through the command line and allows models to be built in the PRISM language and the JANI specification.

When comparing these two tools, it is possible to reflect on the results of the ninth model checking contest, which took place in 2019 [57]. This competition's results saw

PRISM and Storm outperforming each other in certain elements. For example, PRISM could perform a faster analysis of the test models when running both tools in their default configurations. However, Storm outperformed PRISM when both tools were configured for specific tasks. This outcome shows that the older PRISM model checker, especially when used by less experienced users, remains relevant in the presence of the modern Storm model checker.

Both Storm and PRISM are reasonable tools to be used for the work presented within this thesis, with PRISM being the primary tool chosen due to the efficiency of its analysis when using its default configurations and also the greater number of analysis techniques and stochastic models that are accepted. This flexibility and default efficiency lends well to the broad nature of the intended use of the work presented in this thesis.

Storm and PRISM can take input models described using the PRISM modelling languages, a simple state-based language. A model within the PRISM language consists of two primary components. These are *modules* and *variables*. A module is a collection of local variables, and the values of these local variables constitute the current state of the parent module. A model can contain multiple modules that can interact with each other, and the local state of all modules constitutes the model's state. In order to change the state of the model, the modules within the model must contain commands which are defined within the modules. A command takes the form:

$$[action]\ guard -> prob\_1 : update\_1 + ... + prob\_n : update\_n;$$

Where the action label $[action]$ allows transition rewards to be defined and to aid in module synchronisation, the $guard$ consists of behaviour restriction where the module's state must correspond to the guard specified for the command to be executed. $prob\_1$ until $prob\_n$ are probabilities that sum to 1 and specify the probability of what part the command is executed; if $prob\_1$ is met, then $update\_1$ will be executed.

An example of a stochastic model described using the PRISM language can be seen in Listing 2.1, which shows a simple queue where a customer is either waiting to be served, being served, successfully been served, or failed to be served due to an error and the customer must return to waiting. The only reward structure in this system is to do with customer experience; where a customer is failed to be served will produce a punishment.

```
1          mpd
2
3          module customer_queue
4              s : [0..3] init 0;
5              [waiting] s=0 -> (s'=1);
6              [being_serving] s=1 -> 0.5 : (s'=2) + 0.5 : (s'=3);
7              [service_failed] s=2 -> (s'=0);
8              [served_succeeded] s=3 -> (s'=3);
9          endmodule
10
11         rewards "customer_experience_cost"
12             [service_failed] true : 1
13         endrewards
```

Listing 2.1: PRISM Language: Customer Queue Example

In line one, the model type is declared, in this case, an MDP. Next, lines 3 and 9 declare the beginning and end of a module named $customer\_queue$, which describes the customer's current state and defines four commands, all guarded based on the state where the customer resides. These lines update the customers' current state, with the terminal state being 7. Finally, lines 11 and 13 declare the start and end of a reward structure with an optional name that allows the reward structure to be selected within the PCTL statement. For example, when the command $service\_failed$ is executed, the corresponding command in the reward structure will execute.

With this model, where the aim is to determine the maximum cost of customer experience which could occur, we can define this using the PCTL property below.

$$R\{"customer\_experience\_cost"\}max =? \text{ [F s=3]}$$

Quantitative verification, including PCTL and probabilistic model checkers, stands as a defining characteristic of the AMARL approach and, as such, is discussed in chapters 4, 5, and 6.

2.6 Summary

This chapter has introduced the concepts of Markov decision processes (MDPs), abstract Markov decision processes (AMDPs), reinforcement learning, multi-agent reinforcement learning (MARL), and quantitative verification. These concepts form the foundations of the thesis and, as such, the development of assured MARL and its extension for problems that require adaptability. These foundational techniques and technologies are summarised as follows:

- MDPs are used to model sequential decision-making processes and capture their stochasticity. An MDP contains states, actions, transitions, and rewards, forming a description of the problem. To solve an MDP is to find a state-action pair for every state which returns the maximum possible expected reward. These collections of state-action pairs are known as a policy, and the policy that returns the maximum possible expected return is the optimal policy.

- AMDPs is an MDP whose size and complexity have been significantly reduced through state aggregation. This grouping of states allows actions to be represented as high-level options, such as *move from room A to room B*, rather than looking at low-level state transitions. This simplification of the MDP makes analysing them more efficient.

- Reinforcement Learning is a machine learning technique used to solve MDPs when the reward and transition functions are unknown. Reinforcement learning is utilised in an autonomous agent, which explores an environment to learn about its dynamics and exploits the knowledge it has learned. By utilising temporal difference learning algorithms, the agent can learn which actions will promote further reward as it explores the MDP.

- MARL is the natural progression from reinforcement learning and allows multiple agents in a common environment to learn to accomplish goals. MARL can be accomplished through multiple techniques, the simplest being applying individual reinforcement learning techniques to each agent and having them learn independently. Another is to model the system using the Markov game framework, which allows agents to make decisions that take into account the decisions of other agents.

- Quantitative Verification is a formal verification technique that uses exhaustive analysis with proven mathematical formulas to verify properties within stochastic systems. By modelling the stochastic system as an MDP and specifying properties as Probabilistic Computation Tree Logic (PCTL), it is possible to guarantee whether such properties hold.

Chapter 3

Related Work on Safe Reinforcement Learning

This chapter begins with a definition of both safety and risk to frame the use of these terms throughout the rest of the thesis. These definitions are followed by a discussion of other significant safe MARL approaches that have emerged from strains of research, including the benefits of their application, limitations, and how they compare to the AMARL approach we present in this thesis. Finally, this chapter aims to give a broad overview of safe MARL techniques that will help place AMARL within the surrounding research.

3.1 Defining Safety and Risk

Safety is a widely used term in the literature and can be used to convey different meanings dependent on the target problem in reinforcement learning. Safe MARL also has many definitions that are typically assigned on an ad-hoc basis. We align our work with previous reinforcement learning research that utilises formal methods to produce safe policies. In this thesis, we define safety such that under certain circumstances, an undesirable event will never occur [15]. In the context of robotic systems, undesirable events would be the robotic system and sensitive equipment sustaining damage or humans being allowed to come to harm.

Risk is defined within Risk-Management as the possibility of events that lead to adverse events occurring [21]. Risk is also often quantified by its outcome's cost or impact; In this thesis and other safety-critical works [86], the cost of an adverse effect will be mission failure. In this work, we consider the risk high when it corresponds to an agent within the system taking one or more unsafe actions that have a high probability of incurring mission failure. Conversely, low risk corresponds to a low probability of mission failure with a system that may take unsafe actions of low impact sparingly.

A safe MARL system within safety-critical and mission-critical scenarios is therefore defined as a system that minimises risk while still completing mission objectives to an acceptable degree by meeting certain criteria [46, 86]. Such criteria could be reaching a set number of goals while keeping the probability of a robotic agent succumbing to damage below a certain amount.

3.2 Safe Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL), as with single-agent reinforcement learning, has two components which play an important role in their learning processes. The first of these components is the exploration strategy which is used to explore all of the states in the environment. The second component is the optimisation strategy used to ensure the agents reach their mission goals efficiently. Previous works in safe reinforcement learning [46] have used these components to categorise safe reinforcement learning research into two distinct categories: safe exploration techniques; and safe optimisation techniques. Because of the novelty of safe MARL and the direct relationship between much of safe MARLs techniques with safe reinforcement learning techniques, this section is organised using these two categories.

Each of the two components highlighted above has distinct ways in which they can lead to potentially hazardous actions and unsafe behaviours. Exploration strategies, such as the ϵ*-greedy* strategy [126], sporadically assign random actions to agents during the learning process to find actions with higher utility in under-explored regions of the state space. When using reinforcement learning methods in a safety-critical scenario, where certain actions will lead to potential risk, random action selection strategies can see agents taking actions that should never be taken or repeatedly taking actions that should be used sparingly. In contrast, optimisation strategies are focused towards maximising cumulative rewards with little concern for the side effects of this focus. As such, the system will actively enter states with varying degrees of risk, possibly multiple areas of risk, with no regard for possible adverse outcomes. While some approaches attempt to incorporate safety requirements into the rewards that the system receives [46], the safety requirements are likely to negatively impact the system's efficiency in reaching its goal or to be ineffective. This impact can make it difficult, if not infeasible, to capture multiple conflicting objectives in a reward structure.

Traditional strategies of exploration and optimisation in MARL and reinforcement learning are ill-equipped to deal with safety concerns [46], and multiple safety approaches have been formulated. An overview of these approaches can be seen in Figure 3.1 and described in the remainder of this section.

3.2.1 Exploration

In traditional reinforcement learning and MARL, the learning process begins with limited or no prior knowledge of the environment and is allowed to explore and gather information from all possible states. However, this lack of prior knowledge can lead to learners entering unsafe states without knowing that these states are unsafe. Therefore, to attempt to mitigate these problems, multiple methods, which are described below, have been created to either drive learners away from unsafe states over time or introduce knowledge to avoid unsafe states more consistently. In this section, we present a number of existing approaches to this problem.

3.2.1.1 External Knowledge

External knowledge is any information that is used to aid the MARL system complete its goal in a safe way prior to any learning [46]. This information could be in the form of demonstrations, constraints, and so on. External knowledge is very commonly required within safe reinforcement learning and safe MARL. While the most common kind of External Knowledge is Prior Knowledge, other forms of external knowledge can also lead to safe exploration, including Backup Policies, Teacher Advice, Cautious Simulation and Safe Demonstration.

Prior Knowledge, of some kind, is needed for all of the safety methods shown in Figure 3.1. Many of these methods require us to define the relationship between states, actions and safety before the learning proceeds. A number of methods also require us to define the severity of states and actions with respect to safety properties. As such,

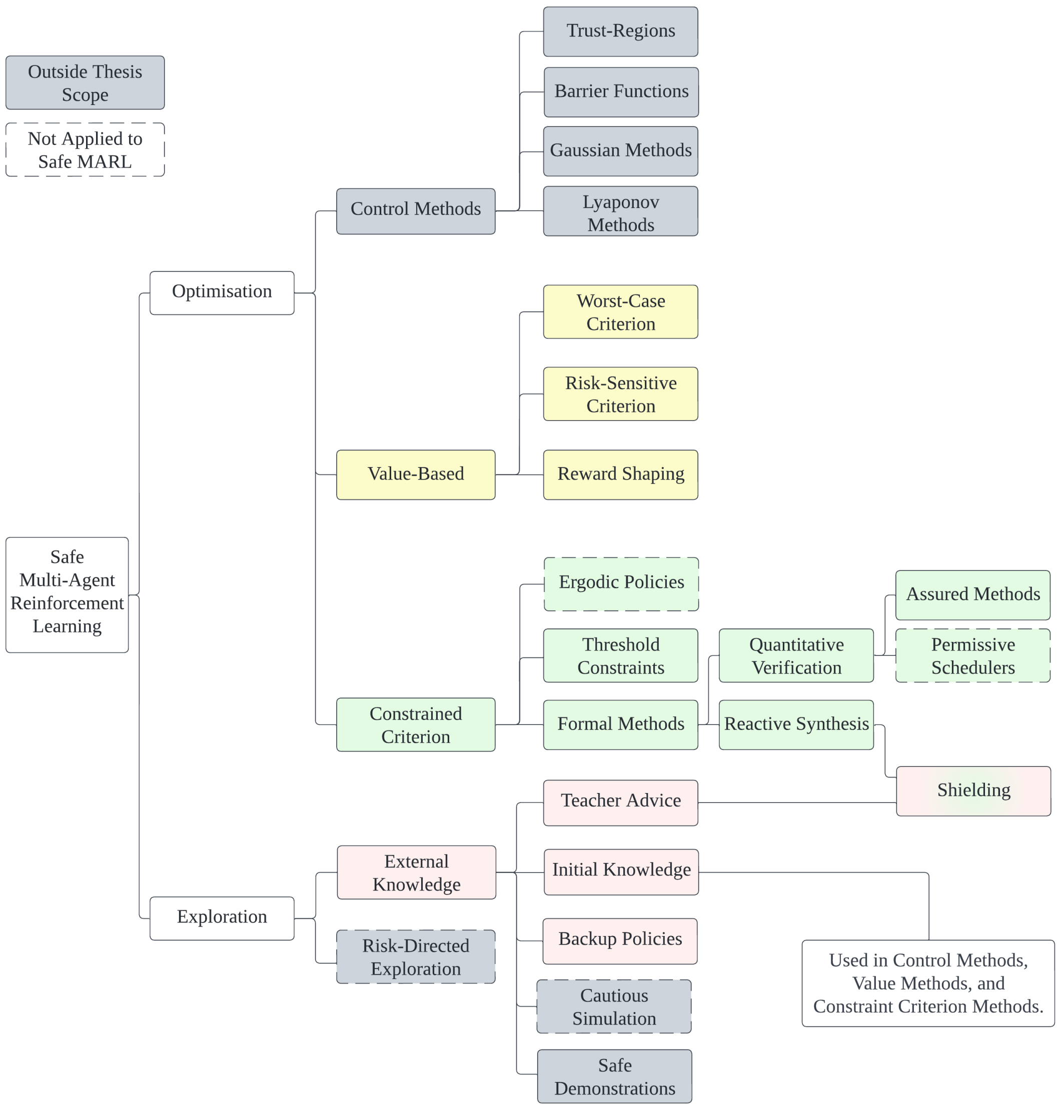


Figure 3.1: Types of Model-based Safe MARL Approaches.

exploration within safe reinforcement learning environments is likely to require prior knowledge of some kind [46].

Back-up Policies are used to verify if actions taken by the agent with a particular state have the potential to lead to unsafe states. If this is the case, we provide an 'escape action' or 'backup action' that the agent near this unsafe state can use to return to a safe state. The original backup policy algorithm has been discussed in previous works [59, 83, 123]. The Safety Handling Exploration with Risk Perception (SHERPA) algorithm utilises a risk perception function to evaluate every possible action the agent may take against some safety threshold, known as a filter. If an agent attempts to take an action which will potentially lead to an unsafe action, then an alternative safe action will be chosen. However, if the action will lead to a safe state, then the algorithm begins to search for a safe backup action for this future state. If all actions in the future state are evaluated, and there are no safe states to store as a backup, then this future state is not transitioned to, and the current state backup action is used.

An accurate perception function and a filter that meets safety requirements guarantee that no unsafe states will be visited as any risks will be detected and mitigated. However, these guarantees only hold under certain conditions. Firstly there is an assumption that for the backup policy method, all unsafe states are able to be both perceivable and identified accurately by this risk perception function which may be non-trivial. Furthermore, depending on the selected safety threshold, the method may be overly cautious, leading to sub-optimal policies, most notably by disallowing actions that lead to states without a backup action. Finally, it can not be guaranteed that taking actions without a backup action will lead to an unsafe state, but, in fact, it could potentially lead to a highly functional state.

The backup policy method can only be used in scenarios in which risk is perceivable and in scenarios that do not require strict probabilistic safety or functional requirements.

Teacher Advice is a method that involves a 'teacher' providing a reinforcement learner with knowledge as it explores the domain space [35]. This advice allows traditional reinforcement learning to take place, with the learner being advised how to behave when confronted with safety concerns. There are two main techniques proposed to incorporate a teacher into the reinforcement learning process: the agent asking the teacher for advice [45], and the teacher providing advice when it feels it is necessary to do so [113]. Teacher advice allows reinforcement learners to explore safely in several ways, either by the teacher supplying a safe set of states within which the agent's behaviour can be optimised or by having suggested actions when the learner moves too close to an unsafe state.

Using teacher advice does not supply safety guarantees but lowers the likelihood of agents entering unsafe states. One can not provide guarantees because the teacher's advice is based on potentially flawed teacher knowledge. When this knowledge is inaccurate, there is potential for learners to be given bad advice [46]. This issue is exasperated further due to the question of when advice should be given or when advice should be requested, potentially leading to a lack of advice. As well as the above limitations, if teacher advice is given without request, then this must be driven by a monitoring process which must remain before and after learning, which may be impractical computationally.

Furthermore, optimality can be significantly reduced depending on how strict and accurate the teacher's advice is, and the flexibility of teacher advice is reliant on the existence of a known prior safe solution. Even with these limitations, teacher advice has been utilised on MARL systems in several scenarios [38].

Cautious Simulation is a method which involves the use of accurate physics simulations and a set of safe example behaviours for a domain within which a domain expert provides guarantees that a leaner will never explore unsafe states [105, 107]. By using these physics simulators to represent the problem domain, states within the simulation can be identified as safe or unsafe based on the output of multiple simulations that run over a set of safe example behaviours. These simulations can create a safety function to label states in the problem domain as safe or unsafe. In addition, a domain expert may add or amend appropriate labels. Reinforcement learning can then be allowed to optimise within the set of states which have been labelled as safe. Unsafe states cannot be labelled as safe given a sufficiently accurate physics simulation and cautious state classification. The role of the domain expert is then to identify any safe states labelled as unsafe by the cautious classification and alter these appropriately.

However, the issue with cautious simulation is that the safety function can be overly cautious, labelling safe states as unsafe, and there is no way of guaranteeing that the domain expert will identify and alter all of these correctly. Therefore, due to safety limitations and overly cautious classification, reinforcement learning can become inefficient, meaning that we can not be guaranteed to produce an optimal solution. Furthermore, producing a sufficiently accurate simulation for open-world environments is non-trivial and the safety function will only be as accurate as the set of simulations employed.

Safe Demonstration is a technique that allows reinforcement learners to develop behaviours, which are often complex, based on demonstrations [1, 142]. This method has been demonstrated using video footage of a human actor illustrating how to accomplish tasks. These demonstrations can be used not only to define the models' dynamics but also as a reward function that will allow reinforcement learners to optimise their behaviour based on these demonstrations. The main limitations of these methods are that the reinforcement learner is limited in how well it can perform by the quality of the human demonstrator from which it defines its reward function. Furthermore, it may not be possible for the demonstration to cover all potential states, and if the reinforcement learner encounters states not addressed within the demonstrations, then the reinforcement learner will not have behaviour to copy. This can potentially lead to unsafe behaviour [46], and hence we are not able to guarantee safety. Finally, the safe demonstration method is limited to scenarios where a safe solution is known prior to learning, as the demonstration must be supplied in advance. It has, however, but has been utilised on MARL systems [98].

Risk-Directed Exploration is a method similar to the constrained criterion introduced later in this chapter, as it utilises a risk metric to determine if a state should be visited. The key difference between the constrained criterion and risk-directed exploration is that risk-directed exploration only drives behaviour towards safer states; it is not a hard constraint [46]. An example of a risk metric is to take the weighted sum of the stochasticity of the action outcomes of a state-action pair and combine this with the normalised expected reward from said state-action pair [76]. Taking these two aspects of the risk metric makes it possible to produce a *risk-adjusted utility* of a state action pair. The reinforcement learner can then learn to optimise using this risk-adjusted utility. This utility will push learners to produce behaviours that visit states with high functionality and low risk.

Due to the behaviour of Risk-Directed Exploration, no guarantees can be made on safety since the risk metrics used in this method are not constraints but incentives. Therefore, it is still possible for unsafe states to be visited more frequently than necessary and since this method has to learn these incentives, in the initial learning stages, unsafe states will likely be visited sporadically [46]. Also, depending on the selected metric, the risk metric can cause functionality to suffer significantly due to the selection of what is deemed acceptable. Lastly, this approach is unsuitable for scenarios requiring strict probabilistic safety and functional requirements due to the limitations mentioned.

3.2.2 Optimisation

Issues with the optimisation process within MARL arise due to the traditional reinforcement learning process being focused only on optimising non-contradicting functional objectives. This narrow focus is, however, incompatible with safe reinforcement learning when safety features become the primary concern. In an attempt to address this, multiple methods have been developed that attempt to mitigate this limitation for practical, safe reinforcement learning and safe MARL. In this section, we highlight a number of commonly used methods.

3.2.2.1 Model-free and Control Theory Setting based Learning Methods

A large portion of safe reinforcement learning and safe MARL research is focused on methods that tackle model-free learning problems that do not use transition probability distributions associated with Markov decision processes. Model-free methods have been derived for control-theoretic settings. With regard to the work presented in this thesis, the most notable use of these methods is for the safe navigation of aerial and land-based autonomous teams. However, most methods based on control theory are not naturally paired with settings involving constrained MDPs, which are a core principle for this thesis. For this reason, we do not discuss control theoretic methods in detail here. A more in-depth description of control theory-based methods, model-free safe reinforcement learning, and safe MARL, can be found in the following survey papers [61, 81]. The methods outlined in these surveys are typically broken up into four categories, which are: Trust-region-optimisation [20, 71, 78, 90]; Control-Barriers [26, 29, 30, 85]; Gaussian processes [17, 25, 41, 134]; and Lyapunov functions [30, 32, 108, 146].

3.2.2.2 Value-Based Methods

Value-based methods attempt to promote safe behaviours by incorporating risk avoidance into the value functions of the reinforcement learning agents while promoting functional behaviour. The key methods included within this area are reward-shaping, worst-case criterion, and risk-sensitive methods.

Reward-Shaping Methods, attempt to balance rewards for potentially conflicting objectives, and it's use has been shown in both safe reinforcement learning [37, 94, 129] and safe MARL [39]. Reward-shaping is typically used to aid learning agents in accomplishing functional goals by altering the rewards received based on the utility of agents' actions and utilising prior knowledge of unsafe states. Shaping a reward function for functional objectives is still required in the realm of safe reinforcement learning and safe MARL because there is an added conflicting goal of avoiding risky scenarios. While this is a natural extension for safe reinforcement learning, it involves a lengthy trial and error process to find a balanced reward structure that guides behaviour towards goals while avoiding risks. In some cases, creating a reward structure that guides behaviour in the desired directions may not be feasible, and, as such, it becomes difficult or even impossible to provide safety guarantees. As well as this time-expensive trial and error limitation, there is a need to have explicit knowledge of which states are unsafe and such information may not be easily obtained. The limitations of reward-shaping have led to the development of further methods, introduced in this section, which move away from answering the question of how rewards can be changed to promote safety.

Worst-Case Criterion involves encoding the worst-case value of an action as the utility of a state-action pair and has been utilised in safe reinforcement learning [124, 141], and safe MARL [144]. This encoding of the worst-case value is counterproductive when using the standard exploitation action selection method found in Q-learning since exploiting this knowledge will result in the learner choosing actions associated with the worst possible outcome. However, if the *MinMax* [47] exploitation method is selected, then the selected actions will have the least harmful possible outcome, given correct prior knowledge. This action selection behaviour has been described as pessimistic behaviour selection, meaning that it acts as if anything that can go wrong *will* go wrong [46], and this behaviour potentially results in a low variance in obtained rewards. With a low variance in obtained reward, it can then be expected that risky scenarios will be less likely to be visited and become part of the final policy, allowing the learner to meet set safety requirements. However, since this criterion is likely overly pessimistic, it results in poor functionality. Lastly, this method attempts to capture safety properties within rewards, much like reward shaping and shares the same limitations.

Risk-Sensitive Criterion, attempts to lower the variance in expected rewards by setting a permissible level of variance and has been used in safe reinforcement learning [14, 42, 49] and safe MARL [40, 112]. In methods such as reward shaping, the value function is used to encapsulate safety properties, given accurate prior knowledge of these properties. This value function then allows an agent to predict, and receive, a reward that it would obtain from performing an action within a state. The idea behind both reward shaping and the worst-case criterion is that a high variance in the reward obtained indicates that the behaviour is more likely to entail moving into unsafe states. In the case of reward shaping, this is because a state action pair that produces a high reward is likely to be associated with greater functionality, while a small or negative reward is likely to be unsafe. The risk-sensitive criterion takes this shared premise and uses a parameter to determine how much variance in the expected reward is acceptable or not [93]. This solution can be used to obtain a low variance in the expected reward. However, it can be expected to produce a lower cumulative reward than is possible, and also has incurred a costly trial and error process, a problem shared with the worst-case criterion [103]. Lastly, since this method utilises instantaneous reward-based variance for decision-making, may not be able to meet all acceptable risk levels for multiple safety properties.

3.2.2.3 Constrained Criterion

Constrained Criterion [69, 94] describes a group of methods that involve the use of constrained MDPs, in which the MDP that disallows agents taking actions that will result in specified criteria exceeding, or falling below, a set threshold. This threshold is typically linked to safety and risk thresholds for safe MARL and is focused on removing, or strategically limiting, unsafe behaviours to guarantee that they are not frivolously pursued both during and after learning. This group of methods is the only group within the scope of this thesis that limits and guides behaviour in this definitive way. The constraint that should be placed on the MDP to guide agent behaviour away from these thresholds can be identified and applied in several ways. The most relevant are reactive systems and constraints guided by quantitative analysis.

Threshold Constraints contain several methods of constraint which do not utilise quantitative analysis or other formal methods of verification. The most natural extension to constraining agent behaviour consists of using hard constraints such that an agent will not be able to move into any state associated with risk, no matter how small of an impact this may have on the system. This extension has substantial limitations, firstly, all unsafe states must be known prior to the learning process, and this may not be practicable. Secondly, the strict constraints may make completing functional goals unlikely or impossible. A much more common use of constraints utilises thresholds that must not be crossed during the duriation of a learning episode [48] or a threshold defined on the percentage, or likelihood, that an outcome will surpass a set value [31]. Constraint-based approaches have been utilised with actor-critic methods [141], and policy gradient methods in model-free environments [132], and with MARL systems [54].

Threshold methods supply no formal guarantees on meeting safety or functional requirements but do make it less likely for agents to visit unsafe states needlessly. The issue with these methods is that prior knowledge must be supplied. For example, all unsafe states within the MDP must be known, and an accurate perception of how unsafe or how likely these states are to result in a negative outcome must also be known. Due to a lack of formal guarantees or functionality, the safety thresholds selected can significantly reduce the functionality of the system, and these techniques cannot be utilised in scenarios where these formal guarantees are required.

Ergodic Policies can be used to constrain a state-space based on whether the states within said state-space are reachable from all other states. All states which are not Ergodic in nature will be made inaccessible by excluding all actions that lead to the state from the learner's available actions [46]. The concept behind Ergodic Policies is that those states which are not Ergodic are unsafe, and by constraining agent behaviour in this way, the learner cannot enter a state from which it can not recover. However, despite this guarantee, Ergodic Policy constraint will likely produce highly sub-optimal policies due to most real-world scenarios not permitting useful Ergodic Policy use [95]. Furthermore, large amounts of rewards may be unattainable by removing the ability to visit states that contain slight risk due to these states being unrecoverable. Lastly, this method can remove states that are not unsafe, as not all states that are unrecoverable are necessarily unsafe [106]. At the time of writing, Ergodic Policies have not been applied to MARL.

Reactive Systems (Formal Methods) are architectural structures that allow separate components to function together resiliently, taking input and responding with an appropriate output [2]. These systems are expected to maintain an ongoing interaction with their environment with behaviours that are often specified using temporal logic to allow a form of correctness checking to take place [82]. There are four main cornerstones to reactive systems [5], being *responsive* (responding promptly), *resilient* (remains responsive in the face of failure), *elastic* (remain responsive in the case of varying scenarios), and *message driven* (utilising asynchronous messages).

Shielding is a process which allows for the synthesis of a *correct-by-construction* reactive system. In this process, teacher knowledge is provided to the learning system to stop safety violations. Correct-by-construction [70] is a process for creating reliable systems at design time. This process is achieved by breaking a system or problem into small components and describing each of these small components in a succinct specification. These small components can then be refined to avoid introducing errors and to identify potential errors closer to the point of their introduction. With the specifications of each component, powerful tools can be used to check for such errors. This correct-by-construction method of creating reactive systems for reinforcement learning-based shields involves creating a two-player game based on a formalisation specified by temporal logic. In these games, the environment (player one) supplies an observation, and the agent (player two) selects an appropriate action. By repeatedly running this two-player game, it is possible to identify state-action pairings within the state space that will not violate safety violations. The set of state-action pairings deemed safe can then be used as a shield, or teacher, to guide the agent so that when it selects an unsafe action, it is replaced with a safe action. Alternatively, unsafe actions may be removed from the agent entirely.

Shielding has been used in safe reinforcement learning [6, 13], and safe MARL [38, 147] and shows that while a shield is operational, safety can be formally assured. However, all levels of risk for all states must be analysed and stored to synthesise a shield. Depending on the size of the problem environment, the synthesis of these shields can be computationally expensive, and the information stored within the shield can become sizeable. Furthermore, for safety to be assured, the shield must be in place both during and after learning. This need is driven by the shielding method utilising a reactive system approach, meaning in order for an agent's actions to be reviewed and potentially replaced by safe actions, there must be constant monitoring [70]. In addition, there is no guarantee that a safe shield will be created from an optimal policy since this procedure is akin to statistical simulation [143]. Finally, as with quantitative verification, discussed next, any inaccuracies within the formalisation of the problem or issues with the shield's construction will result in assurances becoming outdated and invalid.

Quantitative verification (Formal Methods)

Methods which employ quantitative verification allow for efficient searching through all possible paths within an MDP in other to synthesise paths that meet a set of predefined properties where the properties represent safety constraints and functional requirements. These synthesised paths can then be used to constrain the reinforcement learning or MARL agents, allowing these agents to produce learned policies that are guaranteed to meet the defined requirements. As shown in Figure 3.1, there are currently two main methods that utilise quantitative verification, the first is known as Permissive Schedulers, and the second is known as Assured methods.

Permissive Schedulers [68] have been applied to single-agent reinforcement learning but, as of writing, not to MARL. This technique applies quantitative verification to the agent's MDP and synthesises a set of safety policies. These safe policies are synthesised to meet safety and functional requirements, which are formatted using PCTL. One of these safe policies is then used to constrain the agent's behaviour and guide the agent's exploration process within the domain environment. However, while formally guaranteeing the agent meets safety and functional requirements, this constraint negatively impacts the reinforcement learner's main aim of optimising rewards within the environment. Therefore, permissive schedulers, like other constrained methods, allow the agent to learn to maximise the rewards it can receive under the constraints and not in the domain as a whole.

Despite permissive schedulers being one of the first methods that applied quantitative verification to reinforcement learning, producing assured functional and safety guarantees, it holds significant limitations. These limitations come from using quantitative verification directly on the agent's MDP. The first limitation comes from the need to obtain complete knowledge of the MDP in order for permissive schedulers to be used. This is not always possible and is non-trivial. This need for preliminary information is further exacerbated when looking at the use of permissive schedulers used within the cited study [68]. This study used the MDPs state-transition function to facilitate quantitative verification, which is troublesome given that the state-transition function and the reward function are often learned through the reinforcement learning process. Lastly, permissive schedulers have scalability issues when working with large MDPs or multiple agents. This issue is due to the computation time required to analyse the MDP using quantitative verification, which is infeasible for large models due to the state space explosion problem.

Assured Methods have been applied to both single-agent reinforcement learning [11, 87] and within MARL. The work that supports this thesis and what this thesis was built from was the first to use assured methods in MARL[114–117], but since the completion of this thesis, another paper outside of the work that supports this thesis was published that also utilised assured methods for safe MARL [96]. Assured methods, like permissive schedulers, make use of quantitative verification to synthesise safe policies that are then used to constrain the reinforcement learning or MARL system. This quantitative verification is also guided by requirements that are formatted using PCTL. The main variance between permissive schedulers and assured methods is using an abstracted version of an MDP in which similar states are aggregated. The use of an AMDP mitigates the main issues found within permissive schedulers. Using an AMDP within this process allows it to be used on domains where partial knowledge is held, knowledge of states, transitions, rewards, costs, and safety is required, but, unlike permissive schedulers, knowledge of transitional functions is not necessary. The AMDP also allows domains with larger states and transitions to be analysed, and unlike permissive schedulers, it has promoted the use of quantitative verification with MARL systems.

The first limitation of assured methods is shared in all safe MARL work, which is the need for preliminary knowledge. This preliminary knowledge is required to synthesise the constraints that allow assurances of safety and functionality to be made. The second limitation is the size of models and systems that these approaches can handle. Although assured methods allow much bigger state spaces to be analysed, there is still a correlation between the efficiency of quantitative verification and the size of the model it is analysing.

3.3 Comparison to AMARL and Extensions

The methods we present within this thesis, AMARL and its Extensions, AMARL-PPR and AMARL-AC, fall into the category of safe exploitation. This categorisation does not require modification of the exploration procedure, allowing for any exploratory strategy to be used without additional steps being required.

The methods introduced in this chapter can be compared based on four main factors: the assumptions and requirements of use, the safety assurances delivered, the effect on optimality, and how flexible the method is in its use.

3.3.1 Assumptions and Requirements

The AMARL process, as described in chapter 4, relies on an AMDP, meaning it requires accurate prior information on safety and functional properties in the form of abstracted states. Secondly, AMARL requires that safety and functional requirements are defined using the PCTL syntax. In addition, prior knowledge of transition properties and rewards is desirable but optional within AMARL.

Other formal verification methods, such as assured reinforcement learning [87], permissive schedulers [68], and shielding [6], require the developer to know aspects of an MDP or AMDP. Assured reinforcement learning, like AMARL, requires an abstraction of an MDP which retains safety and functional details. Likewise, permissive schedulers require encompassing knowledge of an MDP, including the transition function. While shielding also requires full knowledge of an MDP or an AMDP.

Non-formal methods, such as those that modify the value function of agents, such as reward shaping [94], worst-case criterion [124], risk-sensitive criterion [14], and risk-directed exploration [46], often require knowledge of all risky states. Reward shaping requires prior knowledge of all risks and a reward function that balances risk and functionality and this can be time-consuming, if not infeasible, to achieve. Worst-case and risk-sensitive criteria share the need for prior knowledge of all risky states and a reward function that punishes unsafe actions in order to discourage their selection. Risk-directed exploration also requires knowledge of all risky states and a risk metric to discourage actions. These all have the limitation of needing full knowledge of all risky states, while AMARL only requires the knowledge of risk within an abstracted state space.

The methods that utilise expert knowledge are more varied in their requirements. For example, teacher advice [35] requires a priori safe policy to guide the system and a risk filter to decide when the teacher should advise the system. The cautious simulation method [107] requires a provably accurate physics simulator, which can be difficult and time-consuming to create. In comparison, the safe demonstration method [1] requires an expert that can provide safe demonstrations.

Lastly, AMARL-AC, described in chapter 6, does not require predefined knowledge of where safety and functional properties are, but these safety and functional properties must be perceivable. This limitation is similar to the backup method [59], which also requires risk to be perceivable. Even though AMARL-AC requires less prior abstracted knowledge, perceivable safety and functional properties are a large assumption.

3.3.2 Safety Assurances

AMARL, AMARL-PPR, and AMARL-AC rely on formal verification methods to produce abstract constraining policies that are guaranteed to meet safety and functional requirements. There exist only three other approaches that offer guarantees through formal verification. These are permissible schedulers [68], assured reinforcement learning [87], and shielding [6].

Like AMARL, permissive schedulers utilise quantitative verification to produce constraints that are guaranteed to meet safety and functional requirements. However, permissive schedulers require complete knowledge of the MDP for the problem. This includes the transition properties, which can be difficult to obtain and also suffers from state-explosion limitations regarding quantitative verification, limiting its scope of use.

Assured reinforcement learning also utilises quantitative verification to produce constraints guaranteed to meet safety and functional requirements. This method, like AMARL, also utilises AMDPs though the method focuses on single-agent reinforcement learning. However, AMARL-AC and assured reinforcement learning differ more significantly, with AMARL-AC producing safety and functional guarantees in problems with limited knowledge of states, transitions, safety properties, and functional properties prior to learning.

Shielding is the last approach that utilises formal verification, though, unlike AMARL, assured reinforcement learning, and permissive schedulers, it does not use quantitative verification. Instead, this runs different policies through a constructed version of an MDP or AMDP and selects one policy to act as a shield or a teacher. However, like statistical verification, which does a similar thing, depending on how long it is left to run, it is not guaranteed that this type of verification will find the optimal policy or even a safe policy, even after prolonged verification.

Methods such as teacher advice [35], cautious simulation [107], and safe demonstration [1] are only as safe as the teacher, simulator, or demonstrator is accurate or safe. In contrast, risk-directed exploration [46], reward shaping [94], worst-case criterion, risk-sensitive criterion, and threshold constraints do not hold any formal guarantees but focus on pushing agents towards behaviours that are less likely to visit states needlessly by incorporating safety into value functions or through constraint.

3.3.3 Optimality

AMARL and its extensions utilise PCTL to define both safety and functional requirements. A benefit of AMARL is synthesising a set of Pareto-optimal abstract constraints. Therefore, the domain expert can select a solution which is a compromise between safety and functionality. Like all safe reinforcement learning and safe MARL techniques, there are direct impacts on optimality due to incorporating safety, and this is also true for AMARL during the verification stage. The same is true for assured reinforcement learning [87] and permissive schedulers [68].

Shielding [6] also allows a compromise between safety and functionality. However, due to the lack of quantitative verification, these policies tested and proven to meet functional and safety requirements are not likely to be Pareto-optimal, causing either safety or functionality to take precedence and the other to be unnecessarily sub-optimal under the constraints. As with AMARL, the generated shield is likely to cause limitations on optimality for safety considerations.

Threshold constraints [48] and Ergodic policies [46] also directly constrain the environment differently. Threshold constraints will disallow actions that take risks above a certain threshold. However, as these constraints are not produced through formal verification, there is no bound on how much optimality can be affected. Ergodic policies, however, will disallow any state that is not reachable by any other state. The issue arises when a non-ergodic state is not unsafe and contains a functional value.

Methods utilising expert knowledge, such as teacher advice [46], cautious simulation [107], and demonstration [1], will only be as optimal as the expert knowledge allows it to be, which depending on the scenario, can vary greatly.

While the other methods which adapt value functions [14, 46, 94] will have no guarantee of reaching high functionality. In case of worst-case criterion, [124], is highly likely to have low functionality.

3.3.4 Flexibility

AMARL and AMARL-PPR have no inherent use case limitations and are suited to problems with conflicting requirements. While both permissive schedulers [68] and assured reinforcement learning [87] are either currently limited to, or solely focused on, single-agent reinforcement learning. Shielding [6] shares AMARLs flexibility, however, it has the possibility of struggling with very large state spaces both due to the method of statistical verification employed and the construction of a potentially bloated shield system that has to remain active both during and after learning.

Outside of the formal methods described, there are clear limitations. All of the methods that utilise expert knowledge have noticeable use limitations, with both teacher advice [46] and safe demonstration [1] requiring a safe solution to be known in advance. Cautious simulation [107] has the added limitation of being only applicable to physical systems. In addition, ergodic policies [46] cannot be utilised on problems that are not proveably Ergodic. The remaining methods can not be used in scenarios with strict probabilistic safety or functional requirements.

AMARL-AC, unlike the other methods, can also be applied to scenarios where complete information on transitions, states, and safety and functional properties are unknown. However, this is replaced with an assumption that these safety and functional properties are perceivable. This means AMARL-AC can not be used in scenarios where the system's safety and functional properties are not recognisable and perceivable.

3.4 Summary

This chapter describes existing methods for safe reinforcement learning and safe MARL with methods categorised based on how they attempt to influence the learning process to achieve the defined safety requirements. The first category manipulates the behaviour of the system as it explores the environment [1, 35, 59, 76, 107], whilst the second concerns exploitation as the system attempts to find optimal policies to solve the stated problem [2, 14, 48, 68, 69, 87, 94, 95, 124]. The work we present in this thesis, AMARL and its extensions AMARL-PPR and AMARL-AC, all fall into the exploitation-based category of safe MARL. AMARL and its extensions can explore the environment without modifying a typical exploration technique. Instead, the techniques which are used for optimisation are limited by constraints that assure that safety requirements are met.

All approaches and varying methods share five characteristics by which they can be compared. These are the assumptions and requirements that must be met in order for these methods need to be used, what assurances they offer, how they impact optimally, how flexible they can be in adapting to varying scenarios, and finally, if they have been applied to MARL systems or not [46]. It is within these five characteristics that each method has strengths and limitations. Primarily these methods pursue one area at the disadvantage of another, such as constraining behaviour to produce safety guarantees but removing functionality and therefore negatively affecting optimally. In order to more easily distinguish each method's relationship with these distinct areas, they are detailed within Table 3.1.

Of all the methods detailed within this chapter that relate to AMARL, AMARL-PPR, and AMARL-AC, Assured Reinforcement Learning [87] acts as a foundation method that inspired the initial AMARL approach. Assured Reinforcement Learning and AMARL share many assumptions and requirements, assurances, optimality, and generalisability, with the exception being that Assured Reinforcement Learning does not tackle safe MARL. At the same time, the method that shares the most similarities to Assured Reinforcement Learning and AMARL is the method that utilises Permissive Schedulers [68]. First, since Permissive Schedulers involve applying quantitative verification directly to an MDP, instead of an AMDP, as with AMARL, more detailed knowledge of the MDP is required, such as transitional values, while these are not in AMARL. Secondly, while the assurances and optimally are the same as with Assured methods, the generalisability is greatly diminished due to the need to analyse the entire MDP, which is unpractical and even infeasible with larger models and systems. This lack of generalisability is why Permissive Schedulers are not naturally applicable to MARL systems.

Since AMARL-PPR is an extension to AMARL that allows the partial reuse of learned policies when learning is required to be rerun on an altered domain space, there are no significant differences to be distinguished between AMARL-PPR and AMARL. With the exception that this extension works exclusively with learning techniques that store state-action values in a tabular format. However, the same can not be said for AMARL-AC, which utilises a monitor-analyse-execute process structure. AMARL-AC can be utilised to adapt to changes in a domain during run time and monitor for said changes throughout the learning process. This monitoring for changes requires the MARL system to perceive goals, potential risks, transitions, and states. This need for goals, risks, transitions, and states to be perceivable is a large assumption that must be met to utilise AMARL-AC effectively.

When these assumptions can be met, the assurances and optimality of AMARL-AC should mirror that of AMARL. If assumptions are met, the main positive of AMARL-AC is the increased generalisability from which it benefits. AMARL-AC can be utilised in the same scenarios as AMARL and scenarios with partial knowledge of the goals, risks, abstracted states and abstracted transitions. The closest approach to AMARL-AC regarding its monitor-analyse-plan-execute process structure is Shielding [6]. This similarity comes from Shielding using a reactive system that monitors potential actions, evaluates safety, and then executes changes to help guide a system towards safe behaviours. The main difference between shielding and AMARL-AC is that the shield that is utilised in shielding must be used after learning as halted, while AMARL-AC no longer requires continued monitoring. Secondly, shielding is not currently capable of adapting to such previously unknown conditions.

Table 3.1: Comparison of Safe MARL Approaches

Method	Assumptions & Requirements	Safety Assurances	Optimality	Flexbility
Exploration				
Back-up Policies	Prior Knowledge of Risky States. A filter and perception function to detect risk.	Given an accurate filter it is guaranteed that no unsafe states will be visited.	Filter can cause functionality to suffer.	Not applicable to scenarios that require strict probabilistic safety or functional requirements. APPLIED TO MARL [13].
Teacher Advice	A priori safe policy to guide systems. A risk filter to deselect unsafe actions.	No guarantees but lower probabilities of entering unsafe states.	Restrained by how optimal the teacher advice is and how strict the filter is.	Limited to scenarios where a safe solution is known priori. APPLIED TO MARL [38].
Cautious Simulation	An accurate physics simulator.	Cautious state classification means no unsafe states can be classified safe.	Limited to physical systems.	Limited to Physical systems.
Safe Demonstration	A domain expert available to provide safe demonstration.	No guarantees when states are visited which were not the focus of the demonstrations.	Only as efficient as the demonstrator.	Limited to scenarios where a safe solution is known priori. APPLIED TO MARL [98].
Risk-Directed Exploration	Prior Knowledge of Risky States. A risk metric to discourage unsafe actions.	No guarantees as system's behaviour is guided away from unsafe states but is not explicitly constrained.	Depending on the risk metric functionality can suffer.	Not applicable to scenarios that require strict probabilistic safety or functional requirements.
Exploitation				

Reward-Shaping	Prior knowledge of risks. A reward function that balances risk and functionality is feasible.	No guarantees due to difficulties in designing functions that produce desired behaviours.	Depending on the chosen reward function, functionality can suffer.	Not applicable to scenarios that require strict probabilistic safety or functional requirements. APPLIED TO MARL [39].
Worst-Case Criterion	Prior knowledge of all risky states. Reward function to punish unsafe actions.	Due to pessimistic exploration, a minimum level of safety can be confidently gained.	High probability of states with high functionality being avoided due to pessimistic view on risk.	Not applicable to scenarios that require strict probabilistic safety or functional requirements. APPLIED TO MARL [144].
Risk-Sensitive Criterion	Prior knowledge of all risky states. Reward function to punish unsafe actions.	Reduces variance in obtained rewards but not guarantees to meet safety properties. Not suited for meeting multiple safety properties.	Depending on the chosen risk parameter, functionality can suffer.	Not applicable to scenarios that require strict probabilistic safety or functional requirements. APPLIED TO MARL [40].
Threshold Constraints	Prior knowledge of likelihood of adverse outcomes.	No formal guarantees of safety or functional requirements being met. Less likely to needlessly visit unsafe states.	Depending on the thresholds deemed as acceptable, functionality can suffer.	Not applicable to scenarios that require strict probabilistic safety or functional requirements. APPLIED TO MARL [54].
Ergodic Policies	A safe policy exists that does not require non-ergodic states.	Is not capable of guarantees specific safety requirements. Only the ergodicity of a solution.	Due to the discounting of non-ergodic states, functionality can suffer greatly.	Not suited to problems whose solutions are not provably ergodic.

Shielding	Full knowledge of an MDP to allow sequences of actions to be analysed to create a shield.	Given a correctly constructed shield, guarantees safety and functional requirements.	Optimality limited by constraints but formally guaranteed to reach functional requirements.	No obvious limitations apart from larger MDPs can result in cumbersome shield construction times, and the shield must remain active during learning. APPLIED TO MARL [38].
Permissive Schedulers	Encompassing knowledge of an MDP including transition function.	Formally guarantees extensive safety, functional, and other non-functional requirements.	Optimality limited by constraints but formally guaranteed to reach functional requirements.	Not suitable for problems with a large state-space. Such as in MARL.
ARL	Abstraction of reinforcement learning MDPs retaining safety and functional detail.	Formally guarantees extensive safety, functional, and other non-functional requirements.	Optimality limited by constraints but formally guaranteed to reach functional requirements.	No obvious limitation. Suited to problems with strict and often conflicting safety and functional requirements. Developed for single-agent use.
AMARL	Abstraction of MARL MDPs retaining safety and functional detail.	Formally guarantees extensive safety, functional, and other non-functional requirements.	Optimality limited by constraints but formally guaranteed to reach functional requirements.	No obvious limitation. Suited to problems with strict and often conflicting safety and functional requirements. APPLIED TO MARL.
AMARL-AC	A monitoring system that can detect risks and rewards. Ability for MARL system to translate AMDP constraints to a domain space.	Given an accurate monitoring system and long enough learning durations, formally guarantees extensive safety, functional, and other non-functional requirements.	Optimality limited by constraints but formally guaranteed to reach functional requirements.	Limited to scenarios in which goals and risks are perceivable. Suited to problems with strict and often conflicting safety and functional requirements. APPLIED TO MARL.

Chapter 4

Assured Multi-Agent Reinforcement Learning

This chapter delivers a conceptual overview of Assured Multi-Agent Reinforcement Learning (AMARL), describes its implementation, and reviews its performance in example domains.

4.1 Introduction

In this chapter, the novel, safe MARL approach, AMARL, is introduced. AMARL is a four-stage plugin-styled approach that formally assures that a system utilising MARL will meet strict performance, safety, and other non-functional requirements during and after its learning process.

The incentives behind the creation of AMARL are due to the commonly practised methods of producing MARL policies and the inherent stochasticity of reinforcement learning. In MARL, objectives for multiple agents are defined, and the behaviours of MARL agents are driven towards reaching these objectives through the use of numerical rewards and punishments. These numerical rewards have been used successfully for learning agents when the problem's objectives are 'simple'. However, when objectives have more complexity, it becomes infeasible to design efficient reward functions. The complexity here can be from the problem containing multiple non-functional objectives or conflicting objectives. For example, an agent receiving a reward for reaching a functional objective and punishment for putting itself at risk is unlikely to lead to an agent's behaviour converging to the desired behaviour. This difficulty is due to the agent's inability to perceive the efficiency of its actions other than this single numerical value. Increasing the information within the MARL problem can help with these issues but potentially add to the state-space explosion problem that is already present with every additional agent that is added to a MAS. Furthermore, if possible, any attempt to construct a reward function to balance these objectives would require a costly trial-and-error procedure. This process is commonly called Reward Shaping and involves running a learning process repeatedly with different reward structures until the required behaviours are produced.

AMARL overcomes these issues by allowing traditional reward functions to be used without the need for over-designing to encompass these complex and conflicting objectives. This is accomplished with the use of formal verification techniques that are utilised on an abstracted version of the problem domain, producing what are known as *abstract policies*, which show how the system can reach both functional and safety requirements. These abstract policies are formally proven to meet these functional and safety requirements, so it is possible to identify a set of low-level state-action pairs that meet all requirements. While also allowing all state-action pairs that do not meet these requirements to be removed from the agents' MDP, a process known as 'constraining'. The MARL system is then free to utilise traditional MARL techniques under the guidance of these constraints.

The abstracted version of the problem domain is expressed as an AMDP and is a significant step within the AMARL process as it alleviates some of the issues of state-space explosion. This non-trivial problem can potentially make the planning and analysis of these systems infeasible. With the use of abstraction, all details of the environment which are not strictly relevant to the functional and safety requirements are removed. By removing these unnecessary details, the complexity of the environmental model is drastically reduced, allowing analysis to become more accessible and safety and functional assurances to be produced. An efficient analysis must be available as these assurances are delivered through the formal proofs synthesised through a formal verification technique called quantitative verification. Quantitative verification identifies paths within the AMDP that meet functional and non-functional requirements and produces these paths as safe abstract policies.

The rest of this Chapter focuses on the individual steps within the AMARL approach and provides an introduction to case studies and experimentation involving the AMARL approach.

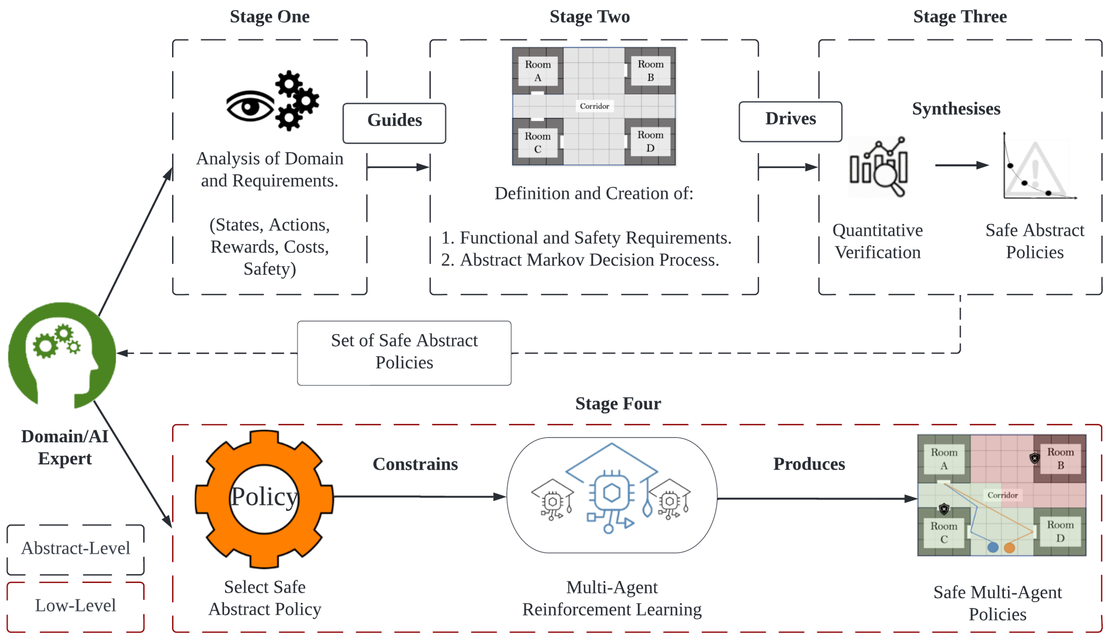

Stage One
Analysis of Domain and Requirements.
(States, Actions, Rewards, Costs, Safety)
Guides
Stage Two
Room A
Room B
Corridor
Room C
Room D
Definition and Creation of:
1. Functional and Safety Requirements.
2. Abstract Markov Decision Process.
Drives
Stage Three
Synthesises
Quantitative Verification
Safe Abstract Policies
Set of Safe Abstract Policies
Domain/AI Expert
Stage Four
Policy
Constrains
Produces
Select Safe Abstract Policy
Multi-Agent Reinforcement Learning
Safe Multi-Agent Policies
Abstract-Level
Low-Level

Figure 4.1: Assured Multi-Agent Reinforcement Learning.

4.2 Approach

The AMARL approach consists of four stages, as seen in Figure 4.1. These stages are kept non-domain or tool specific to emphasise the plugin nature and intended broad usage of the approach.

Prerequisites must be fulfilled before using AMARL, and these prerequisites consist primarily of preliminary knowledge, such as knowledge of states, actions, rewards, costs, and potential safety issues, a common prerequisite of safe MARL methods. While this knowledge does not have to be 'complete', it must contain sufficient information about the environment, which relates directly to safety and functional objectives. With the assumption that this knowledge is accurate regarding the environment and that the MARL system is given ample learning time to converge, the AMARL approach can be effectively utilised to produce learned behaviours that satisfy functional requirements by the time of convergence and safety requirements throughout the entire learning process. If these assumptions are not met, any suggested behaviour constraints will not result in any assurance of safety or functionality in the ensuing MARL stage.

4.2.1 Stage One: Analysis of the Problem Domain

The first stage of the AMARL process involves the analysis of preliminary knowledge to define a set of objectives O=$\{O_1, O_2, ..., O_n\}$, that the MARL system must accomplish through traditional learning techniques involving maximising a reward function or minimising a cost function. Furthermore, these objectives must be reached while adhering to a strict set of safety constraints C=$\{C_1, C_2, ..., C_n\}$, which must be defined by accurately discerning the needs of the problem scenario.

The problem scenario must be analysed to collect this information in order to create an MDP structure. This includes the states, such as geographical states within the environment; the actions that will be available to the agents within these states; and the objectives and costs that can naturally be used within a MARL setting, such as reaching a goal; the resources needed to reach said goal, and any states or actions which include any potential safety concerns. The information collected within this stage is then used to guide stage two of the AMARL approach, which involves creating an AMDP and identifying functional and safety requirements.

4.2.2 Stage Two: Multi-Agent AMDP Construction

The second stage involves using preliminary knowledge gained in stage one to construct an AMDP and define functional and safety requirements.

The definition of functional and safety requirements should be completed before the construction of an AMDP due to how these requirements may alter the structures within the AMDP. These functional requirements can involve, for example, certain objectives, such as reaching a certain amount of goals or retaining a certain level of battery power. At the same time, Safety requirements can involve reducing the probability of agents coming to harm within the environment, such as through collisions.

AMDPs at this stage are simplified models created for efficient quantitative analysis and reduce the complexity of describing the problem scenario in regard to the low-level environment. For example, the low-level environment may consist of Cartesian locations, wherein in each Cartesian location, there are at most four actions (move North, move South, move East, move West). If an environment consists of multiple rooms with hundreds of Cartesian locations, the state-action space would be substantially large, difficult to construct into a model, and inefficient to analyse. By abstracting out the low-level environment, it is possible to drastically reduce a state space by only modelling the transitions and states with meaningful properties, such as transitioning from *roomA* to *roomB*. This simplification allows a model to be constructed with less information and ensures quantitative verification tools can be effectively used. When working with MARL systems, the abstraction of an MDP is particularly important due to the complexity different agents can add to the environment.

In order to facilitate the necessary quantitative verification in stage three, the AMDP should be constructed within a language that is used by the chosen quantitative verification tool. These languages allow the fulfilment of labelling the AMDP with atomic propositions that ultimately allow probabilistic model checking to be utilised. For example, by providing atomic propositions to the states within the AMDP, the model-checking tool can distinguish whether a described goal has been accomplished. Using these probabilistic tools, it is also possible to create structures within these languages to facilitate tracking rewards, costs, and numerical safety features, such as the probability of a system becoming damaged performing a certain behaviour. It is important to identify the objectives and requirements of the problem domain before the construction of the AMDP, as the model must be adapted for effectively tracking these objectives and requirements. Such an example would entail reaching n amount of goal states before the system reaches a terminal state. This can be achieved by shaping the AMDP within the probabilistic tools language to track the number of goals reached, using the atomic propositions to remain aware of these changes within the system.

The rest of this section details an algorithm which was created to automate the creation of AMDPs within a specific language, known as the PRISM language. Though multiple tools accept this language, this algorithm can be adapted to function with other tools that do not use the PRISM language by changing the language's syntax encoded within the algorithm. This algorithm was created to reduce the expertise and time required to create abstract models for quantitative verification, which can be a very time-consuming process. The first algorithm was created, as detailed in Algorithm 3 and its related functions in Algorithm 4, to automate the construction of an AMDP in the PRISM model language given a multi-dimensional array that contains abstracted information from the problem domain and the number of agents that are within the system, and the initial state of the system. The input and output structure

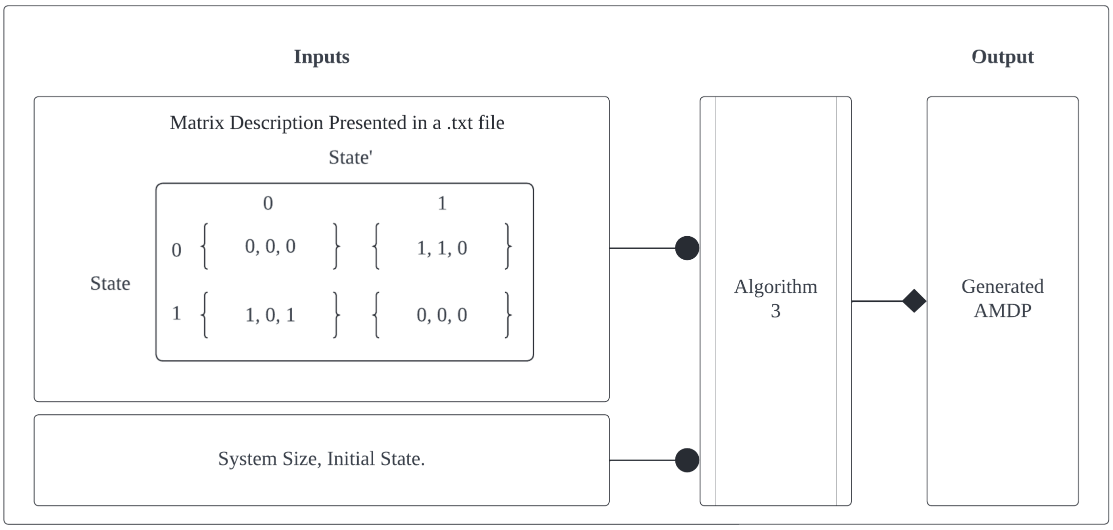


Figure 4.2: Input and Outputs of Algorithm 3. Showing the multi-dimensional array structure for describing an AMDP.

of Algorithm 3 can be seen detailed within Figure 4.2.

When looking at this figure, a multi-dimensional array or matrix structure is shown as an input to Algorithm 3. This is a simplified way of describing the transitions between states within an AMDP. Here states and transitions values of the AMDP are represented by the index values of the multi-dimensional array.

This multi-dimensional array is accessed and manipulated by making use of three values which relate to the indices of the multi-dimensional array. The first value determines which row within the multi-dimensional array is being accessed. As shown in Figure 4.2, the row accessed is a way to represent which state an agent is currently in within. The second value is used to determine which column is being accessed within the multi-dimensional array, where the column that is accessed is used to represent the state that the agent is transitioning to, as shown by *state'* in Figure 4.2. Finally, the third value is used to select which indices should be accessed within the nested arrays that are present in the indices of the multi-dimensional array. The indices within these nested arrays are used to store the properties of an agent transition from its current state to the potential future state. These three indices consist of an integer that acts as a Boolean to state if a possible transition between the current state and the new state is possible, an integer value which shows the potential reward from the transition, and finally, an integer value that shows the cost involved in making this transition.

An example of how to use this Matrix is given. When the Matrix description shown in Figure 4.2, referred to as a Matrix World View in Chapters 5 and 6, is read in by Algorithm 3, it can determine that the abstracted representation of the domain contains two states because there are only two columns, and only two rows, meaning there are only two potential states to transition to. If an agent was currently within state one and wanted to move to state zero, this information can be accessed by using the matrix input values of $[1,0]$. First, it can be determined that a transition is possible from state one to state zero by looking at the first indices of the nested array using the input values of $[1,0,0]$, which will return the value 1, showing that a transition is possible. Secondly, it can be determined that no reward can be obtained from making this transition by using the input values of $[1,0,1]$, which will return the value 0. Lastly, it can be determined that there is a cost to making this transition by using the input values $[1,0,2]$, which will return the value 1. This structure can be adapted to contain many states and transitions by simply increasing the number of rows and columns within the Matrix World View.

The code itself is described with an overview of Algorithm 3. As mentioned previously, the syntax within this algorithm is specific to the PRISM language, which was used in all of the case studies within this thesis. Algorithm 3 begins with the input of the multi-dimensional array as a txt file on line 1 and the conversion of lines of text into an integer matrix through lines 3 to 8. The PRISM model file is created on line 9 and is populated with the initial structures of an MDP model, such as lines 10 and 11, which state for the quantitative analysis tool that the model is an MDP and that the module is named MAS. Lines 12 to 17 use the inputs given, the number of agents and the limit on state visits to further populate the model with a definition of agents within the system and the states within the system. Line 18 defines a *terminal state* as a Boolean, with the aim of the quantitative analysis tool to find a path that sets this Boolean to true. In order to populate the model with all the available transitions, which is the most time-consuming in manual creation, lines 19 to 30 are used. These lines check if a transition is declared within the integer matrix and calls an external function called ‘DefineTransition’ along with the corresponding reward and risk associated with the said transition. In line 18, a set of conditions are selected, which, when met, will result in the terminal condition being met; these conditions are created using lines 31 to 39. Regarding this algorithm, the base conditions are set to a patrolling problem; as such, the conditions are that all states must be visited; however, this can be easily adapted. Finally, the algorithm's closing lines state the MAS module's ending and call on an external function to create the reward structures that will track safety and functional requirements.

Algorithm 4 consists of the external functions made use of within Algorithm 3, named 'DefineTransitions', described in lines 1 to 15, and 'RewardStructure', described in lines 16 to 27. As seen in line 3, each transition is repeated for every agent within the system, with the possibility of further editing to incorporate states that not all agents may visit. Lines 4 to 11 check if any risks or rewards are associated with the transition sent to the function, and if so, then a reward or risk label is added to an array which will be utilised in the function named 'RewardStructure' as well as the actual value of the reward or risk which is stored in a second array. Line 12 uses the given parameters and writes the transition to the PRISM model; these transition definitions are long and often numerous. With the second function, lines 18 to 25 involve the definition of the reward functions for tracking risks and goals and the definition of every transition label resulting in risk and goals being added. This uses both the risk and reward label arrays and the corresponding arrays, which state how much reward or risk should be attributed to these transitions. Finally, on line 26, the final reward structure is closed, and the PRISM model is generated. This model can then be used with a quantitative verification tool or for further editing.

The generated AMDP and the knowledge of functional and safety requirements are then used within stage three to perform quantitative analysis using probabilistic model checking tools, securing formal proofs.

4.2.3 Stage Three: Quantitative Analysis

In the third stage of AMARL, the AMDP and PCTL formatted requirements and objectives are utilised within a probabilistic model checking tool to synthesise *safe abstract policies* that are assured to meet both the objectives and requirements. This quantitative analysis identifies different paths within the model for one or several options which the quantitative analysis tool can synthesise. This is guided based on the reward

Algorithm 3 AMDP Generation

```
Input ← AgentWorldMatrix
for i in Input do
    for x in Input[i] do
        if x == NUMBERCHAR then
            States.append(int(Input[x]))
        end if
    end for
end for
Output ← GENPRISM.nm
Output.Write("mdp")
Output.Write("module MAS")
for i in range NUMBEROFAGENTS do
    Output.write('A' + i + ' : ' + '[0..' + len(States) - 1 + ']' + "init " + INITSTATE + ';')
end for
for i in range len(States) do
    Output.write('visits' + i + ' : [0..' + STATECOUNT + '] init 0' + ';')
end for
Output.write('done : bool init false;')
count ← 0
for i in States do
    TransitionCount ← 0
    TransitionAmount ← len(States[Count]) / 3
    for i in range(TransitionAmount) do
        if TransitionCount == 0 and int(States[Count][0] == 1) then
            DefineTransition(count,Count,States[count][1],States[count][2])
        end if
        TransitionCount++
    end for
    count++
end for
for x in range(len(rooms)) do
    if x == len(rooms) - 1 then
        visitsDoneConditionAr.append(" visits" + str(x) + " >= " + STATECOUNT)
    else
        visitsDoneConditionAr.append(" visits" + str(x) + " >= " + STATECOUNT + " &")
    end if
    donePrint ← donePrint + visitsDoneConditionAr[x]
```

```
38: end for
39: Output.write(“[done]” + donePrint + "-> (done’= true);")
40: Output.write("endmodule")
41: RewardStructures()
```

Algorithm 4 DefineTransition and RewardStructure Functions

```
 1: Function{DefineTransition}{currentState, nextState, reward, risk}
 2: if currentState != nextState then
 3:    for i in range(NUMBEROFAGENTS) do
 4:       if risk != 0 then
 5:          riskLabels.Labels("[visits" + currentState + "_" + nextState + "_" + i + “]”)
 6:          riskList.append(risk)
 7:       end if
 8:       if reward != 0 then
 9:          rewardLabels.append("[visits" + currentState + "_" + nextState + "_" + i +
    "]")
10:          rewardList.append(reward)
11:       end if
12:       Output.write("[visits" + currentState + "_" + nextState + "_" + i + "] "+ "!done &
    !goal" + nextState + " & r" + i + "=" + currentState + " & visits" + nextState + "<" +
    STATECOUNT + " -> 1:(r" + i + "’=" + nextState + ")&(visits" + nextState + "’=visits"
    + nextState + "+1)&(goal" + nextState + "’=true);")
13:    end for
14: end if
15: EndFunction
16: Function{RewardStructure}
17: Output.write(’rewards "risk"’)
18: for x in range(len(riskLabels) do
19:    Output.write(riskLabels[x] + "] true : " + riskList[x] / 10 + ";")
20: end for
21: Output.write("endrewards")
22: Output.write(’rewards "goal"’)
23: for x in range(len(rewardsLabels) do
24:    Output.write(rewardLabels[x] + "] true : " + rewardList[x] / 10 + ";")
25: end for
26: Output.write("endrewards")
27: EndFunction
```

structure defined within the AMDP and the PCTL requirements.

- multi(R"goal"max=? [C<=200], R"risk"min=? [C<=200])

After the quantitative verification tool has identified paths within the given AMDP under the guidance of the PCTL requirements, several Pareto-optimal policies will be synthesised. These safe abstract policies show an order of transitions that are formally assured to meet safety and functional requirement. The domain expert can then remove low-level states and transition options that are not under this safe abstract policy and constrain the MARL system. The domain expert will then be free to select the appropriate policy that meets the requirements. However, these policies are not produced in a format that is readable, and the policy is not sorted over the thousands of possible states. Because of this, Algorithm 5 was created to allow these policies to be used. In order to make use of this algorithm, two files are required, the policy synthesised by the quantitative verification tool and the exported state labels for the model. These numerical labels are required as they supply the algorithm with the initial state that the system starts within and allow the first action to be found in the policy by using these labels. Lines 1 to 10 of the Algorithm open the state labels and begin to search until the value 0 is found; this states that the label next to this value is the initial label for the system. On line 7, this label is stored for use in the second half of the algorithm.

Table 4.1: Original and Revised Policy

Original Policy Form	**Revised Policy Form**
0 0 3 1 visits1_2_1	[A1_1_0]
1 0 4 1 visits1_2_1	[A2_1_2]
2 0 8 1 visits1_4_1	[A1_4_3]

The second half of the algorithm, from lines 11 to 27, searches through the synthesised policy to produce a human-readable version of said policy. Lines 11 and 12 involve reading the original policy and preparing a file for the new human-readable policy. The original policy file is parsed through on line 15 in order to find the next state label; when this is found, the action label is written to the human-readable policy, and the previous state label is stored for further checks. The algorithm ends when the condition specified on line 24 is met, that being that the next state label is equivalent to the previous state label, meaning the policy has been collected.

Algorithm 5 Policy Organisation

```
 1: Input ← InitialState.txt
 2: CheckCounter ← 0
 3: for x in Input do
 4:    LabelReference.append(x.split(:)[0])
 5:    InitalStateCheck.append(x.split(:)[1])
 6:    if InitalStateCheck[CheckCounter] == 0 then
 7:       NextStep ← LabelReference[CheckCounter]
 8:    end if
 9:    CheckCounter++
10: end for
11: InputTwo ← adv1.txt
12: Output ← PolicyOne.txt
13: while done != True do
14:    CheckCounter ← 0
15:    for x in InputTwo do
16:       StateCheck.append(x.split(‘ ’)[0])
17:       if StateCheck[CheckCounter] == NextStep then
18:          Output.write(StateCheck.append(x.split(‘ ’)[2]))
19:          PreviousStep ← NextStep
20:          NextStep ← StateCheck.append(x.split(‘ ’)[1]
21:       end if
22:       CheckCounter++
23:    end for
24:    if NextStep == PreviousStep then
25:       done = True
26:    end if
27: end while
```

This algorithm transforms the original policy of column one in Table 4.1, which is unsorted and unfiltered, to the sorted and filtered version of column two. This allows the policy to be understood by a domain expert.

4.2.4 Stage Four: Constrained MARL

The fourth and final stage of AMARL begins with a set of safe abstract policies, which vary in terms of safety levels and general functionality. These abstracted policies are obtained from stage three through quantitative verification. Due to this quantitative verification, these abstracted policies will fall within the scope of the requirements and objectives. With the potential for multiple safe abstract policies to be available, a selection must be made based on the domain expert's desire to balance functional and non-functional capabilities. This is required as the set of Pareto-optimal abstracted policies will vary on how much one requirement is prioritised over the other.

This stage is the first stage that directly involves the ground-level environment. The ground-level environment is constrained under the rules set by the safe abstract policy. Here, state-action pairings that will go against the guidance of the safe abstract policy are removed from an agent's available action choices. This enables the MARL system to learn to maximise its utility to reach objectives as efficiently as possible while following the constraints that guarantee requirements are abided by. The safe abstract policy identifies which agent is permitted to perform a certain behaviour within the MARL system. This allows flexibility for each agent and becomes increasingly relevant when considering heterogeneous systems. However, while this individualism is a key feature that allows the benefits of MARL to be utilised under this approach, the overall success and safety of the system as a whole is captured.

Under these constraints, the chosen MARL technique should be left to run for sufficient time to converge to a solution. It is common for MARL to be halted before the optimal solution to a problem is found, and instead, a sub-optimal solution is used due to the time-based expense of reinforcement learning. The AMARL approach assures that safety requirements will be met. Therefore, due to constraints, the global optimality of the MARL system is treated as a secondary priority and is expected to be negatively affected by the constraints that assure this safety. The MARL system aims to learn to navigate optimally through the constrained environment, where the environment is constrained as minimally as possible, enabling exploration when it is known to be acceptable under the constraints.

4.3 Evaluation

In order to evaluate AMARL and its flexibility in terms of MARL techniques, the approach to two varieties of navigation-based domains, which will be our case studies, one to showcase traditional MARL and another for deep MARL. However, only navigation-based problems were investigated as this thesis focused on multi-robot systems, which are often involved with navigation tasks.

The first case study is a representation of a multi-agent inspection mission based on real-world problems in nuclear power plants [64] and inspired by the Fukushima Daiichi nuclear power disaster response [97]. Here a MAS must learn to reliably visit points of interest while minimising battery usage and exposure to highly irradiated zones, two conflicting goals which can be counterintuitive to the learning process. As can be seen in Table 4.2, the first case study was used to answer three research questions with six experiments. Experiments 1 and 2 include both a MARL system learning without AMARL to showcase prevalent issues and an experiment utilising AMARL to demonstrate if these prevalent issues are mitigated. Experiments 3 to 5 involved three different MARL algorithms being used alongside AMARL. This is to help demonstrate the plugin style of AMARL in regard to algorithms while also attempting to show if the individual benefits of these algorithms are not negated. The algorithms in question are Independent Q-learners, the game theory-driven Team-Q algorithm, and the direct policy search algorithm known as WoLF-PHC. Experiment 6 contains a heterogeneous system, showing the plugin nature of AMARL in regard to systems and also attempting to show how AMARL can be used to partition tasks in ways which take advantage of the individual abilities of the agents within the heterogeneous system.

The second case study takes the shape of an infiltration and collection-based domain, inspired by current and potential future real-world defence and security problems [7]. Here a MAS utilising a Deep MARL technique must learn to explore a representation of a building with rooms and corridors to collect sensitive equipment while avoiding being in sight of surveillance cameras as much as possible. This case study is utilised in experiment six, which attempts to show that the AMARL process can be utilised with a Deep MARL technique, utilising an AMDP structure to constrain the agents.

All of these experiments were carried out on a computer with an Intel Core i7-

Table 4.2: Research Questions and Relative Experiments.

Experiment	Research Question
Case Study One	
Experiments 1-2	Can AMARL allow a MARL system to reliably meet functional and safety requirements?
Experiments 3-5	Can AMARL allow different algorithms to be utilised and retain their individual benefits?
Experiment 6	Can AMARL be used with heterogeneous system and partition tasks effectively?
Case Study Two	
Experiment 7	Can AMARL be utilised with Deep MARL techniques?

6700HQ 2.60GHz CPU with 32 GB of RAM. In addition, all experiments which are detailed above were run multiple times (five times), as is standard when evaluating stochastic processes, and the final policies which were gained through the learning process were evaluated many times (10,000) in order to ensure that the results are statistically significant [8].

4.3.1 Case Study One: traditional MARL Radiation Avoidance Patrolling Domains

The first set case study in which AMARL is utilised contains three separate domains, as shown in Figure 4.3, which are based on multi-agent inspection and maintenance problems within nuclear power plants. In these scenarios, autonomous robotic systems are tasked with mission-critical objectives corresponding to infrastructure integrity within nuclear power plants. Furthermore, due to the setting, these systems will be exposed to high doses of radiation which at prolonged exposure is fatal to humans and damaging to robotic agents, making these scenarios safety-critical. The robotic systems which are running within the domains introduced must visit all location-based goals, here on called patrol points, at least three times while attempting to maximise battery conservation and limiting time spent within irradiated zones due to potential damage to their hardware. Due to the nature of the problem being investigated within this case study, all domains were constructed within a ROS patrolling simulator [110].

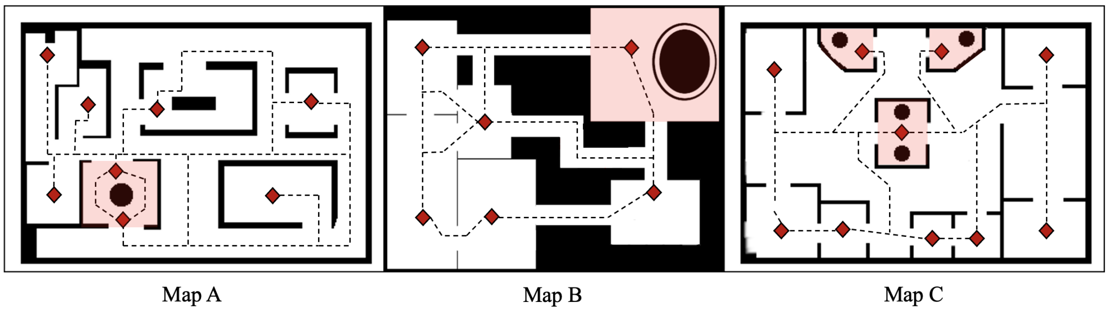


Figure 4.3: Three navigation domains in which n agents must collaboratively visit patrol points (red diamonds) three times while avoiding areas of high radiation (areas in red with black circles) as much as possible.

Map A, as seen from Figure 4.3, is a navigation domain which consists of eight patrol points, two of which are located within an irradiated zone. There are three possible ways for agents within this domain to visit these patrol points, through the two entrances, which are the slightly safer options or to travel through the irradiated zone, increasing time spent within the risky area. This domain is given a two-agent homogeneous system that uses its battery extensively when making choices to navigate between patrol points and a safety requirement of keeping the probability of system damage to 10%.

Map B is a navigation domain consisting of six patrol points with a single patrol point within an irradiated zone. There are two options for the agents within the system to navigate to this patrol point, one which reduces the travel time in this radiation and one which will involve agents spending more time subjected to this radiation. Therefore, this domain is given a two-agent homogeneous system that drains its battery when making choices to navigate between patrol points and a safety requirement of keeping the probability of system damage to 20%.

Map C is a navigation domain that consists of eleven patrol points, with three patrol points within irradiated zones. The top two irradiated zones are less dangerous than the centre zone, so a new system setup is utilised to balance these differences. The system utilised in this domain is a three-agent heterogeneous system comprising two different types of agents. Two lightweight agents are used to move around the larger domain space while using less battery; however, due to the emphasis on lightweight hardware, the agents are less shielded against radiation. Finally, one heavyweight agent is used to contrast the others, using more battery to move its hardware around the domain but holding more protection against radiation. This collaborative system is tasked to utilise its different hardware capabilities to solve the problem efficiently and a safety requirement of keeping the probability of system damage to 20%.

Throughout all these domains, the agents receive a numerical reward when visiting a patrol point based on how long it has been left without an agent visiting it. These rewards are individually given to the agent who visits said patrol point.. However, in order to emphasise patrol points that have been visited less, this reward is reduced by how many times it has already been visited, meaning the agents are pushed to visit the patrol points intelligently. This should promote agent behaviour that does not unnecessarily visit patrol points that no longer require agent interaction and seek out patrol points that have been neglected. This reward structure, which can be seen in Equation 4.1, promotes very efficient battery conservation behaviours and quick goal completion, where $time\_elapsed$ is the time that a patrol point has been left unattended and $times\_visited$ is the times a patrol point has been visited. Agents are given a punishment when an agent is damaged by radiation, which is decided through probability based on the time spent within the irradiated area.

$$f(x) = \begin{cases} time\_elapsed / times\_visited, & \text{if } times\_visited < 3. \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

In these missions, the MAS is expected to learn to complete its mission while using as little battery as possible. In order to evaluate the performance of the MAS, the cumulative battery of the system is used, with cumulative battery meaning all of the agents' remaining batteries are added together at the end of the learning episode. It should be identified that when a system is called efficient, it implies that it has a large amount of unused battery. Furthermore, the system is tasked with remaining below a certain probability of succumbing to damage throughout a learning episode, and this is evaluated using cumulative risk. Since a risk is taken every time an agent within the system enters a potentially dangerous area, the probability associated with these risks is added together to produce an overall probability, or cumulative risk, of the system being damaged throughout the learning episode. Therefore, a system is deemed to be acting safely if the cumulative risk throughout a learning episode never exceeds the acceptable risk level.

4.3.1.1 Experiment 1: Standard MARL

When MARL is applied to a safety-critical and mission-critical scenario, such as the ones Maps A, B, and C introduce, the learning agents can produce efficient behaviours that satisfy functional objectives reliably. However, as the process of MARL is inherently stochastic, it is likely that during the learning process, safety requirements will be repeatedly abused, and there is no guarantee that the final system behaviour will not violate these requirements.

To demonstrate the shortcomings of MARL within safety-critical and mission-critical scenarios and to act as a benchmark, Map A and its MARL system are utilised with independent Q-learners without the guidance of AMARL. This system utilised the same reward structure as shown in Equation 4.1 to guide the learning and was allowed to run for 300 learning episodes. The results from these learning runs can be seen in Figure 4.4 and show the average risk and battery over these 300 learning episodes in a total of five learning runs, with error bars detailing the standard deviation between these five learning runs.

The results that can be seen from these learning runs demonstrate the efficiency of MARL in maximising a simple reward function, with battery levels towards the end of the learning runs being in the high 20s and exceeding beyond this, meaning that the cumulative battery of all agents in the system was above 30% in the final policy. However, without any guidance to drive the system away from unnecessary risk, the cumulative risks that the system took were erratic, unpredictable, and repeatedly violated safety requirements during the learning process. Even with a punishment in the reward structure that attempts to dissuade agents from entering into areas of risk unnecessarily, the system violated safety requirements late into the learning process. This demonstrates the limitations of MARL that attempts to incorporate safety as a product of agents' reward structures; the balance between conflicting objectives is difficult to find and offers little to no confidence in the system's safety.

4.3.1.2 Experiment 2: Initial AMARL Implementation

The AMARL approach is utilised in the same problem domain, Map A. The system is a two-agent homogeneous system utilising independent Q-learning, as in Experiment 1, allowing a comparison between MARL and AMARL.

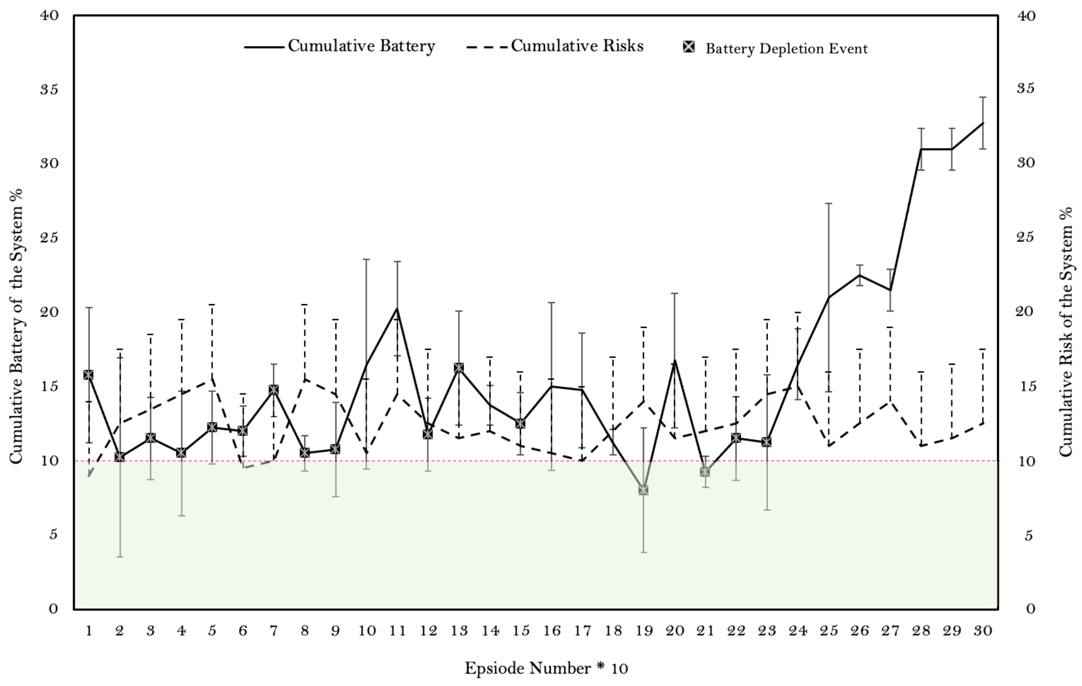


Figure 4.4: Cumulative battery and risk results from five individual learning runs for the Individual Q-learners including two homogeneous agents in Map A without AMARL Guidance.

This experiment was conducted by first following the four stages of AMARL. Starting with stage one, MAP A was analysed regarding states, transitions, rewards, costs, and safety issues. There are eight objectives across seven rooms and a finite number of ways to traverse between different objectives. The *states* within this domain can be derived from the eight objectives if the agent behaviours for transitioning between them are abstracted away. However, each objective has four conditions (not visited, visited once, visited twice, visited three times), meaning these eight abstracted states must also contain four conditions. The *transitions*, when abstracting away the specific agent travel behaviours, are simply transitions between these abstracted states. The *rewards* given to the agents are based on them reaching the abstracted states that have not been visited three times, but since a mission can only be deemed as successful if all abstracted states are visited three times, this reward can be abstracted away. The *cost* of the transitions are related to the battery that is used in the worst-case, as each transition will have a different worst-case battery usage, which will vary depending on the transition made. The *safety* features are related to the probability of risk when entering, leaving or transitioning through possibly irradiated zones which are shown as red-shaded rooms in Figure 4.3.

After this analysis, stage two is followed to create an AMDP, for which aspects of the PRISM model can be found in Listing 4.1. Where the *states* are abstracted down to the eight patrol points and added to the PRISM model from both agent one and two on lines 3 and 4. Also, the conditions of these states are encoded, allowing them to be visited a maximum of three times and for these visits to be tracked, as seen on lines 6 and 7. The *transitions* between these abstracted states are encoded on lines 12 and 13 and simply change the agent's state to the new state and increase the visit count of this state. The *rewards* for reaching the abstracted states are abstracted out entirely. As mentioned previously, the only way for a mission to be completed successfully is for all eight states to be visited three times. Therefore it was possible to set the target end state of the model to encompass all of these objectives as encoded on line 10, removing the need to incorporate these rewards. The battery *costs* of transitioning between the states are able to be encoded and tracked using the reward structure named 'battery' from line 20 onwards. At the same time, the *safety* properties can also be encoded and tracked from another reward structure named 'risk' on line 16 onwards.

```
const N = 3;
module RobotSystem
    r0 : [0..7] init 1;
    r1 : [0..7] init 2;

    visits0 : [0..N] init 0;
    visits1 : [0..N] init 0;
    done : bool init false;

    [] visits0 >= 3 & visits1 >= 3 & visits2 >= 3 & visits3 >= 3 & visits4 >= 3
     & visits5 >= 3 & visits6 >= 3 & visits7 >= 3  & r0 != 3 & r0 != 4 & r1 !=
    3 & r1 != 4-> (done'= true);

    [visits0_1_0] !done & r0=0 & visits1<N -> 1:(r0'=1)&(visits1'=visits1+1);
    [visits0_1_1] !done & r1=0 & visits1<N -> 1:(r1'=1)&(visits1'=visits1+1);

endmodule
rewards "risk"
    [visits0_3_0] true : 0.1;
    [visits0_3_1] true : 0.1;
endrewards
rewards "battery"
    [visits0_1_0] true : 3;
    [visits0_1_1] true : 3;
endrewards
```

Listing 4.1: PRISM Language: Model for a MARL System. Appendix A.

With the AMDP created, stage three can begin, where the PRISM model checker is utilised. PRISM uses PCTL requirements to direct the model checking to identify Pareto-optimal policies that end in mission success and meet all functional and safety requirements. For example, for MAP A, the PCTL command *R'battery'min=? [C<=200], R'risk'min=? [C<=200]* was used. This will cause the probabilistic model checker to find a Pareto front of safe abstract policies that minimise the energy usage of the MARL system and the risk the system takes. Next, Algorithm 5 sorts the policy which is unsorted and unfiltered and displays safe abstract policies in a readable format. With all previous stages complete, stage four was started by the domain expert selecting a

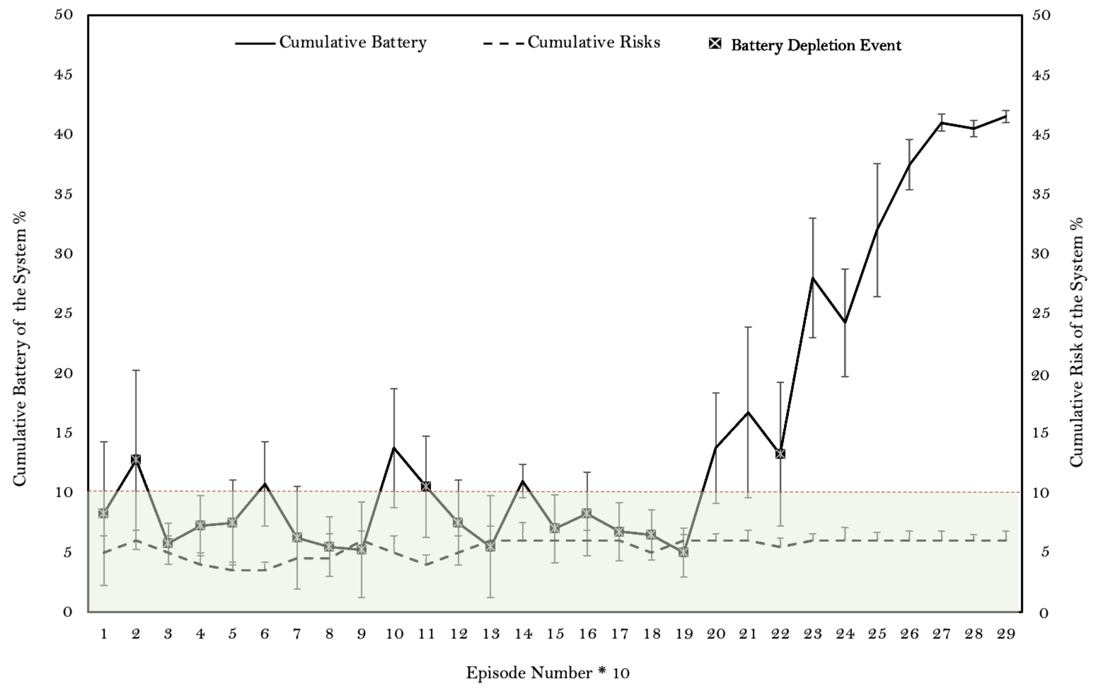


Figure 4.5: Cumulative battery and risk results from five individual learning runs for the Individual Q-learners including two homogeneous agents in Map A with AMARL Guidance.

synthesised safe abstract policy. This policy was used to constrain the MARL system within the lower level domain problem allowing the MARL system to enter into risky scenarios and therefore complete the mission goal, but with the assurances of the safe abstract constraints that the probability of an agent coming under harm never exceeding a set amount.

The results from this initial AMARL implementation can be seen in Figure 4.5, where the average cumulative battery and risk over five learning runs are displayed with their standard deviation. The cumulative risk is consistently around 7%, which is below the 10% safety requirement showing that the formal guarantees of AMARL have held. We can see in Figure 4.6, which contains both the results from MARL and AMARL, that the system's battery is significantly lower than unconstrained MARL until episode 200 where it drastically improved. This is likely due to the constrained nature of AMARL, making it more difficult initially for the system to find as efficient behaviours than without the AMARL constraints. The unconstrained AMARL algorithms

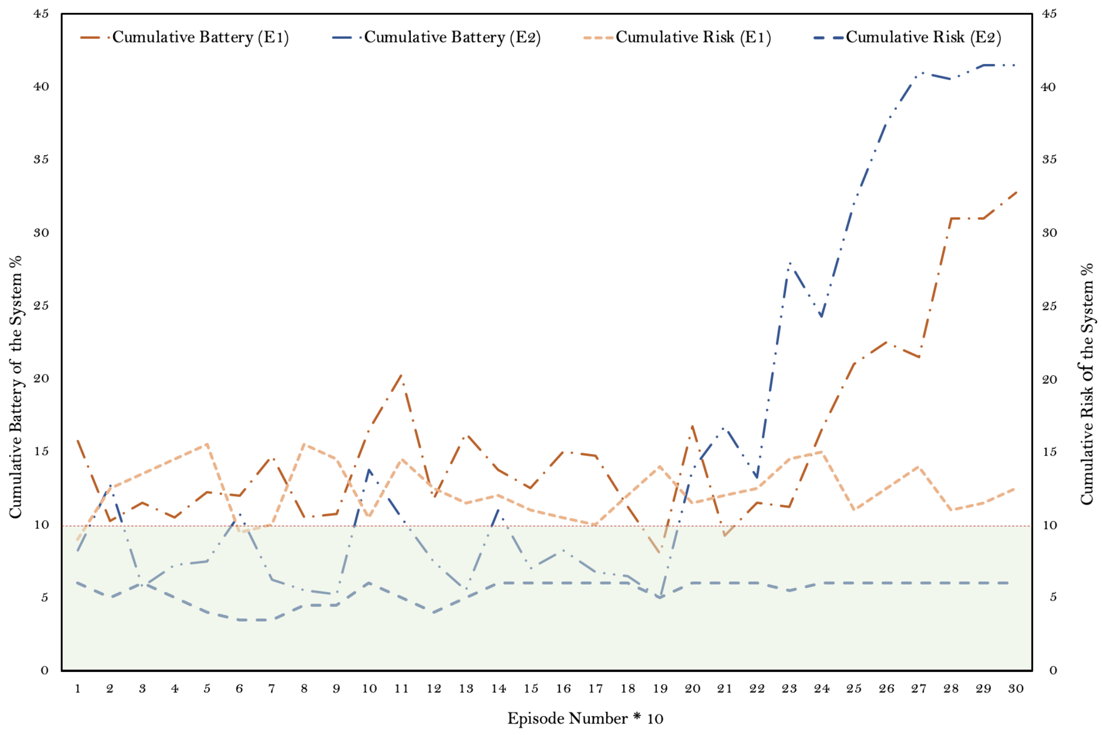


Figure 4.6: Averages from Experiment One and Two over five learning runs.

find behaviours that reliably conserve more battery towards episodes 230 on-wards, though to a lesser extent than the AMARL system. A noticeable difference between the unconstrained and AMARL-constrained results is that the risk levels of the AMARL-constrained system appear to be a relatively flat line through all 300 episodes, never moving over a seven per cent probability of an agent succumbing to damage. This is the main draw of AMARL, the assurances that are delivered. These assurances are that the system will be free to learn a functional policy while never exceeding a certain probability of negative outcomes, and the final policy will meet functional and safety requirements.

4.3.1.3 Experiments 3-5: Differing Algorithm Structures and AMARL

With AMARL shown to function with Independent Q-learners in Experiment 2, the simplest implementation of a MARL system, the evaluation of AMARL continued to demonstrate its ability to be utilised with other MARL techniques. Three algorithms were evaluated using Map B and the system that is utilised within it; these were chosen due to the algorithm's structural differences, utilising game theory and direct policy search techniques, which, as well as altering the implementation, also alters the system's learning behaviours. Different algorithms were used with AMARL to attempt to demonstrate the plugin nature of AMARL and that the approach allows the individual benefits and behaviours of these varying algorithms to be exploited. The first algorithm that is used is the constrained independent Q-learning algorithm that was utilised previously. This is reused to highlight the patterns caused by the unique behaviours of the other two newly introduced algorithms.

The second algorithm is a direct-policy search learning algorithm known as 'Win or Lose Fast (WoLF) - Policy Hill Climb (PHC)' (WoLF-PHC). WoLF-PHC extends Q-learning by utilising a probabilistic policy which follows a probability distribution function, meaning that for every set of actions in a state, there is a probability distribution that dictates how likely each action is to be selected, rather than selecting the action with the highest Q-value as with standard Q-learning. In order to drive efficient learning, WoLF-PHC will allow continued exploration and learning when it is performing well. However, when its performance drops, it will attempt to end the learning episode as quickly as possible.

The third algorithm is a variant of Q-learning which utilises game theory techniques, named Team-Q. Team-Q consists of a Markov game being created every time the agents' within the system must take action. These games can be competitive, but the agents within the domain play cooperative Markov games due to the Map B domain being cooperative. Here the agents receive rewards based on the utility of both agents' actions, meaning behaviours begin to coordinate towards the mission goal, making agents capable of more intelligent decision-making.

With the system in Map B being outfitted with standard temporal difference learning, direct policy search, and game theory techniques, the AMARL approach was utilised, allowing for an investigation into how AMARL can guide these different learning techniques. As with all the Radiation Avoidance Patrolling Domains, the objective is for all patrol points to be visited three times. However, in Map B, the system must keep the probability of damage occurring to the system below 20%. Furthermore, it must do this while attempting to maximise battery conservation. The results from running these experiments with independent Q-learners can be seen in Figure 4.7,

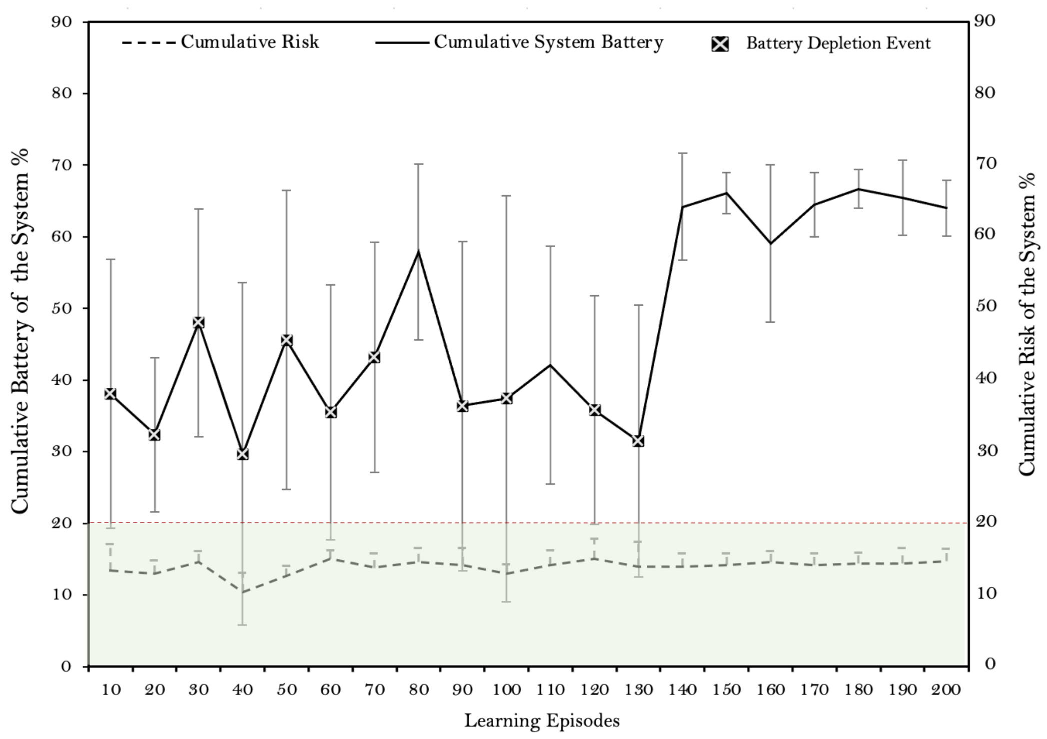


Figure 4.7: Cumulative battery and risk results from five individual learning runs for the Constrained Individual Q-learners including two homogeneous agents in Map B.

WoLF-PHC learners can be seen in Figure 4.8, and Team-Q learners can be seen in Figure 4.9.

The error bars in Figure 4.7 show that the variance between the five learning episodes is large during the beginning and slowly reduces in size until similar policies are found at the end of the learning episodes. The independent Q-learners' results are as one would expect, given that the agents are learning with no regard to each other's actions, and begin with an osculating battery conservation value, with battery conservation being the optimising value. This occurs due to two separate agents attempting to utilise Q-learning in an environment that they are both manipulating, meaning the optimal action choice at a certain time step depends on the previous and current actions of all agents within the system. However, as these agents are operating with a mutual ignorance of each other, this will lead to fluctuating values. Due to the complexity that is involved with two agents learning in a shared environment with this mutual ignorance, it can take a larger number of learning episodes for the agents

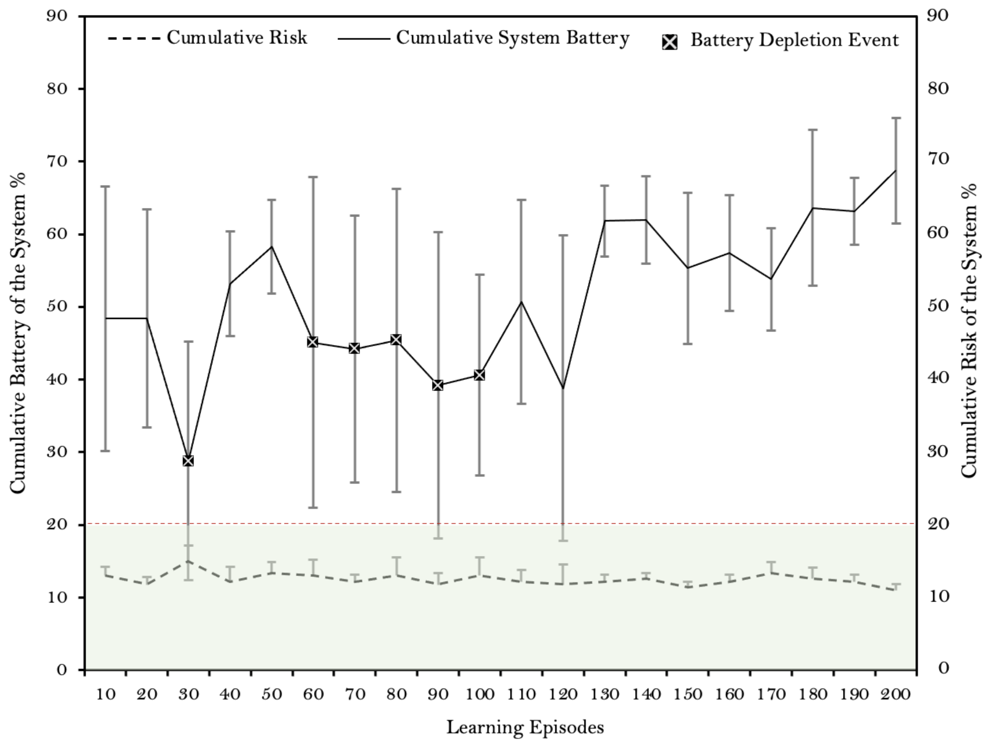


Figure 4.8: Cumulative battery and risk results from five individual learning runs for the Constrained WoLF learners including two homogeneous agents in Map B.

to find efficient policies, as can be seen with a sudden jump in battery conservation at episode 140, but also find successful policies, as can be seen, by the long sequence of learning episodes were battery depletion events occurred. Crucially, however, with the guidance of AMARL, it can be seen that the risk level never exceeded 20% throughout any of the learning runs, and this continued after learning was concluded.

When looking at Figure 4.8, which shows the results of WolF-PHC, the 'Win or Lose Fast' behaviour of the algorithm can be seen from episodes 30 to 100, where the algorithm quickly ended the learning episode when performance decreased in order to find a more efficient behaviour. It can be seen that this algorithm typically found highly efficient behaviours from learning episode 130 and continued to have the greatest performance regarding battery conservation out of the three algorithms. The greater variance between learning episodes that can be seen between episodes 10 to 120 shows the probability distribution sampling techniques that underpin the algorithm and, in an unconstrained scenario, would likely result in frequent high

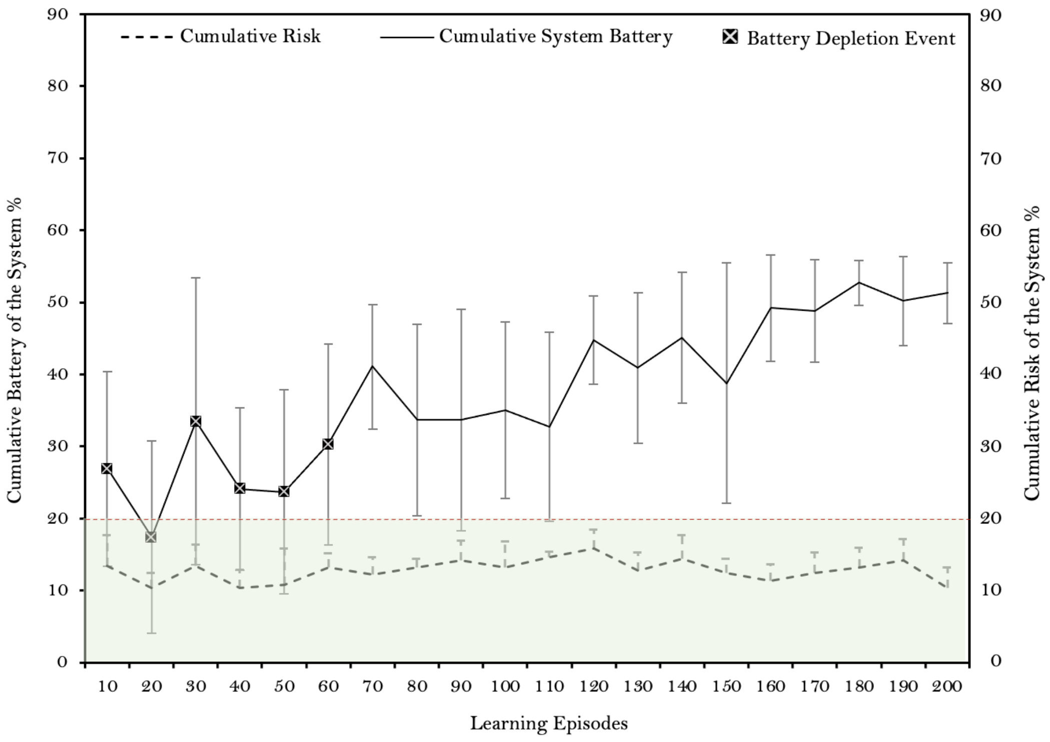


Figure 4.9: Cumulative battery and risk results from five individual learning runs for the Constrained Team-Q learners including two homogeneous agents in Map B.

cumulative risk. However, due to the guidance of AMARL, this algorithm which relies on greater flexibility in terms of action selection exploration, never exceeded the lower than 20% chance of damage requirement.

When looking at Figure 4.9, the game theory-led behaviour of Team-Q can be seen regarding the steady climb in regards to battery conservation as the agents share knowledge of their actions and work directly to achieve the mission goal. This intelligent joint decision-making led to Team-Q reaching consistent mission success quicker than the other two algorithms, which can be seen from episode 70 onwards. However, due to the limitations of Team-Q and the Markov game foundations, the battery conservation levels remained the lowest out of all three algorithms. This is due to the added complexity that these Markov games add to the learning process and the increased time requirements that this adds. Also, a common issue with Markov game-led algorithms is the difficulty of attributing awards to specific agents within the system and specifying which individual action out of the joint actions had the greater utility. These issues likely impacted the efficiency of the learning process and would require longer learning runs to produce the same battery conservation levels as the other two algorithms. However, due to the intelligent joint decision-making technique, Team-Q delivered a final policy with the lowest level of risk. As with the other two algorithms, under the guidance of AMARL, they never exceeded a risk level of 20%.

The results of these experiments which were summarised in Table 4.3, show how AMARL allows the different behaviours and benefits of various reinforcement learning techniques to continue to be utilised. In regards to the variance in battery and risk found in each technique, five learning runs, it can be seen that independent learners, which do not factor in other agents' actions within their learning process, would have the greatest differences, while Team-Q which directly utilises other agents actions in their learning process would have the lowest variance. At the same time, it is shown that Team-Q was the quickest to find an approach that resulted in no battery depletion events and the lowest risk, but due to the larger Markov game structure that is generated at each joint state in the domain, it took longer to policies that produced more reserved cumulative battery. While with WoLF-PHC, due to it being a direct-policy search algorithm, was able to utilise its more flexible probability-based action selection behaviour to locate the highest amount of reserved cumulative battery over the learning periods. However, WoLF-PHC, like independent Q-learners, do not take into account other agents during its learning process, and as such, produced higher variance in its learning runs. But due to the 'win or lose fast' behaviour, it uses it was still able to outperform independent Q-learns in every aspect.

Experiments 3-5 show the plugin nature of AMARL, and despite its constraint-based approach to safe MARL, is still able to be utilised without removing the individual benefits of certain learning techniques. As has been shown in Table 4.3, the safety, efficiency, and amount of variance of the learning system can be altered by the technique it uses, and AMARL allows flexibility of choice.

4.3.1.4 Experiment 6: Heterogeneous Systems and AMARL

After demonstrating different algorithm structures plugged into AMARL, it is very relevant to question how different system types can be plugged into AMARL. One of

Properties	Indi Q-Learners	Team-Q	WoLF-PHC
Final Cumulative Battery	64% (Average)	51% (Average)	68% (Average)
Final Cumulative Risk	15% (Average)	10% (Average)	11% (Average)
Final Battery Depletion	Episode 130	Episode 60	Episode 100
Variance	Highest	Lowest	Moderate

Table 4.3: Summary of Results from Experiments 3-5 including Independent Q-Learners, Team-Q Learners, and WoLF-PHC learners.

the primary benefits of multi-agent systems is the ability for agents to delegate tasks based on individual functionality, utilising their heterogeneity. This heterogeneity is a strength that can be utilised in many different scenarios. As such, Map C and its system are used to investigate how heterogeneous systems can be incorporated into the AMARL approach. This system contains three agents, two of which are lightweight and use less battery, and one which is heavier and therefore uses more battery to move but has increased protection from radiation.

In order to utilise AMARL for the mentioned heterogeneous system, stage one must be followed with further attention given to how the different abilities of the agents will affect costs, safety issues, and the system's capabilities to reach goals. In stage two, simple changes are made to the process of creating an AMDP within the PRISM language or a language from another verification tool. Certain instructions must be added that capture the agents' independent capabilities, such as being able to move to states that other agents cannot, and the reward structures must capture the differing properties of these agents. In the case of the system in Map C, these differing properties are the amount of battery that an agent uses to perform the same task may differ drastically, or the probability of an agent sustaining damage may be less than other agents. These differences can be captured by assigning labels to transitions, such as $[A1\_0\_1]$, where $A1$ is the name of the agent, 0 is the current state, and 1 is the state to be transitioned to. By changing the agent name within the label, it is possible to show that only certain agents can make certain transitions and to assign agent-specific costs and rewards to said transitions.

When the safe abstract policies are synthesised in stage three, the policies are produced that are assured to meet safety and functional properties as specified. However, a benefit of AMARL, as well as the functional and safety assurances, is its ability to

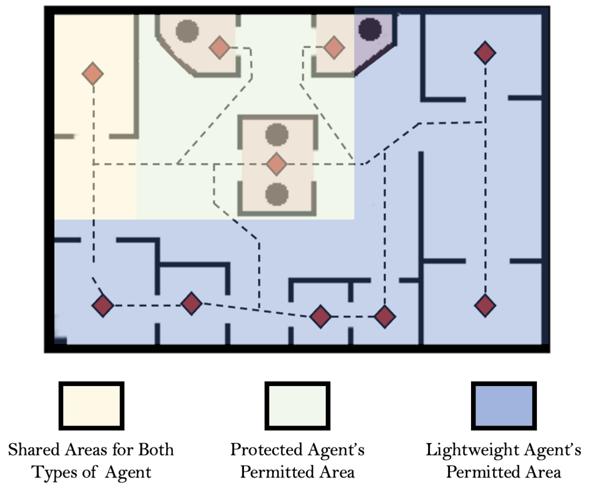

Shared Areas for Both Types of Agent
Protected Agent's Permitted Area
Lightweight Agent's Permitted Area

Figure 4.10: Map C Partitioned for the Heterogeneous System.

intelligently suggest task partitions for multi-agent systems. This is apparent when using AMARL with heterogeneous systems with distinct abilities. This extra benefit is the product of producing Pareto-optimal policies, which naturally partition tasks to agents which will be more efficient at completing them given specific properties. In regards to Map C, the abstract safe policy partitions can be seen in Figure 4.10. Here it shows how the model checker has synthesised a policy that assigns tasks to specific agents based on their strengths and weaknesses. The two lightweight agents were given the task of travelling greater distances which corresponds to less battery consumption, and the protected agent travelling smaller distances but almost exclusively to irradiated zones, reducing the risk to the system.

In Figure 4.11, the results of running the heterogeneous system a total of five times can be seen. The efficient partitioning of tasks can be seen when looking at the battery depletion events in Figure 4.11. After only 20 learning episodes over five learning runs, the three-agent system learned to complete patrols consistently with no agents within the system, fully depleting their batteries. After this, with the protected agent only allowed to move short distances and primarily to irradiated patrol points, the two smaller agents gradually learned to maximise their battery conservation over the remaining 180 learning episodes. Because of the partitioning of risky actions to the protected agent, the amount of risk throughout the learning episodes stayed

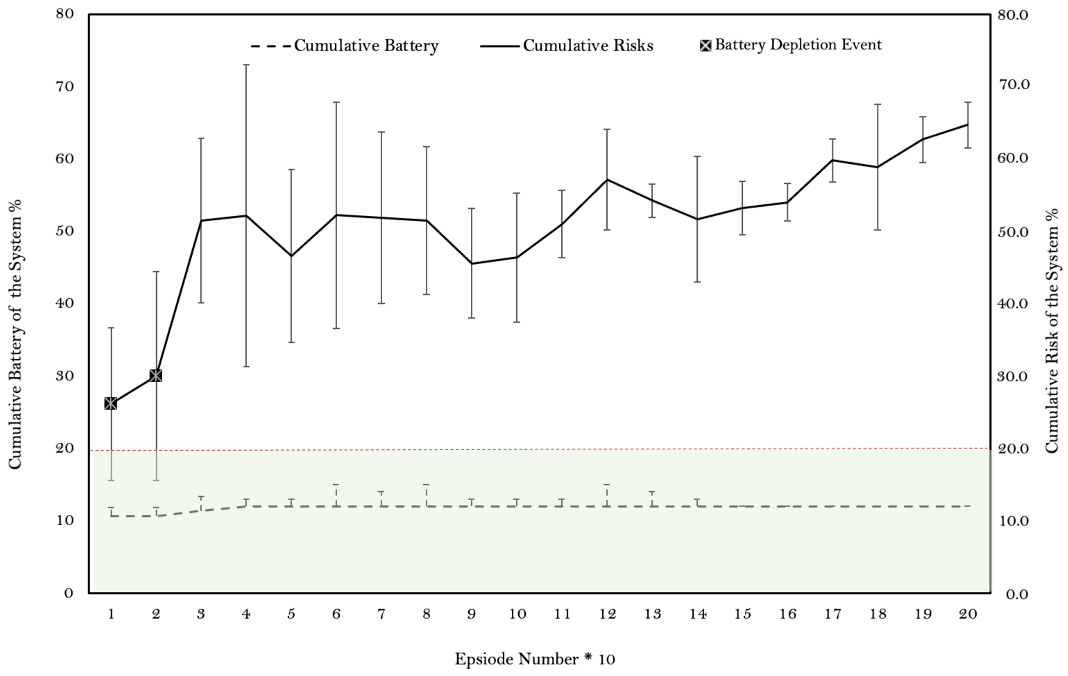


Figure 4.11: Cumulative battery and risk results from five individual learning runs for the Individual Q-learners including three heterogeneous agents in Map C with AMARL Guidance.

mostly between 8% to 15%. However, as the learning progressed from episode 130 onwards, the variance between these learning runs in terms of risk became small to non-existent, remaining at around 11%.

What experiment 6 aimed to achieve was to showcase how AMARL can be used to produce safe abstract policies that allow the individual capabilities of agents within a system to be utilised effectively. Through the safe abstract policy that was produced and then shown in Figure 4.10, and the following five learning runs displayed in Figure 4.11 it has been shown that AMARL can be used to assure functional and safety properties and also create efficient task partitions.

4.3.2 Case Study Two: Deep MARL Infiltration Domain

A substantial amount of MARL research involves using neural networks to facilitate learning [9], otherwise known as deep learning, where the word 'deep' is in reference to the layers within a neural network. In order to demonstrate that AMARL is a valid approach concerning the deep MARL research community, a navigation domain was created within a simulator that warrants deep MARL techniques. This was created using the Unity games engine and a MARL system that utilises the MLAgents plugin [75]. This new domain, as shown in Figure 4.12, is based upon the premise of a team of agents which are given an infiltration and collection mission, which would fall under a defence and security setting. Infiltration and or collection-styled missions are missions that place robotic systems in the role of an adverse force attempting to infiltrate a space. This domain, which consists of nine rooms and various connecting hallways, is given a system of three homogeneous agents, which are required to navigate through the rooms and hallways collecting items, in this case, flags, which are located in every room. The system must collect at least six out of nine flags for the mission to be deemed a success while learning how to navigate correctly through the domain. However, as this is an infiltration domain, the system must remain covert and undetected by the surveillance cameras protecting these flags. As the agents move into view of these surveillance cameras, there are set probabilities of agents being detected by the partially monitored feeds, and the number of instances of coming into view of a camera should be limited. The cumulative risk of an agent being detected should be kept below 30%.

The agents within this system act as independent learners, using the deep reinforcement learning algorithm called Proximal Policy Optimization (PPO) [121]. This algorithm attempts to minimise a cost function while refraining from making considerable changes to the previous policy it held using trust region updates. This allows learning to occur without the agent changing its behaviour rapidly for a reward that may be sub-optimal or dependent on sparse conditions. Furthermore, to promote more persistent exploration, Intrinsic Curiosity [104] is also factored into the learning algorithm. Intrinsic Curiosity is an additional feature that rewards agents for finding actions within states that do not return the expected outcome, pushing agents further into unexplored areas.

Utilising AMARL in this domain requires the analysis of stage one, recognising the states, transitions, goal locations, and the risk involved with moving between these states. The risk involved with making certain transitions vary with different hallways and cameras which are partially operated. Cameras h_2, h_5, h_7, h_8 have a probability of 5% of detecting an agent, cameras h_3, h_4 have a probability of 10%, and camera h_{10}

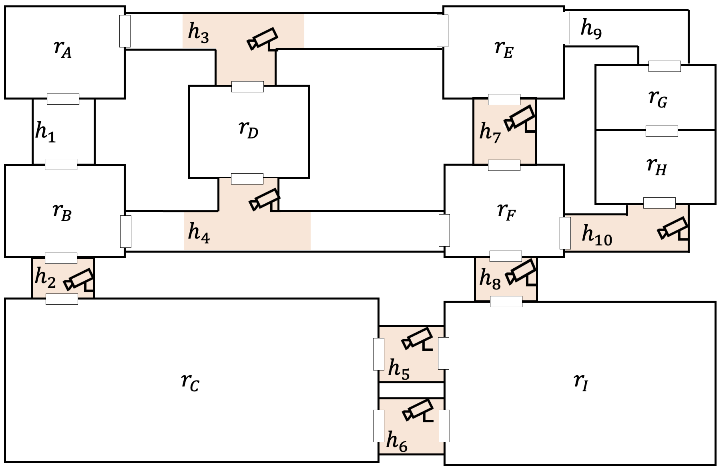


Figure 4.12: An infiltration-themed navigation domain built within the Unity games engine, consisting of three homogeneous agents, surveillance cameras which must be avoided, and nine rooms which contain flags that must be collected.

has a probability of 15%. In stage one, it is possible to treat entire rooms as a single state and each hallway being treated as a single transition, meaning in the domain's initial state, all transitions will result in a goal being reached.

In stage two, using the information gained in stage one, an AMDP was created using the Algorithm 3 by inputting a multi-dimensional matrix which contains nine columns and nine rows, one row for each of the abstracted states within the domain and one row each for the possible transitional relationships between these abstracted states. In this stage the functional and safety requirements are also identified, here the overall probability of an agent being discovered should remain below 30%, and at least six out of nine goals should be reached.

In stage three, this AMDP was then subjected to quantitative verification, where the requirements were to maximise the potential goals retrieved and minimise the encounters with surveillance cameras. The resulting policy, which can be seen in Figure 4.13, allows robot one to pass through hallway h_3 to collect the rewards in rooms r_E, r_G, and r_H, robot two to collect the reward in rooms r_A and r_B, and robot three moving through hallways h_3 and h_5 to collect the rewards in rooms r_C and r_i.

The policy was synthesised based on the availability of rewards without encountering cameras and the probability of an agent being detected depending on the specific camera encountered. As can be seen, based on the policy that has been synthesised,

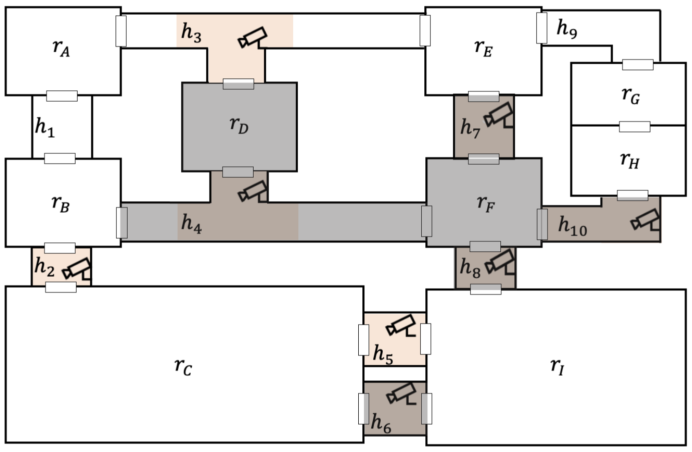


Figure 4.13: Constraining Safe Abstract Policy with Inaccessible Areas in Grey.

the assurances given are that the MARL system will converge to a cumulative reward of seven and a cumulative probability of detection that does not exceed 0.3 or 30%. This means that the domain expert can move forward with confidence if these assurances meet the mission requirements or will know not to proceed if this falls short of said mission requirements. In order to demonstrate these assurances, the three-agent system was constrained in stage four under the above policy and allowed to learn over 1400 episodes, with 6000 steps per learning episode. The agents within this simulator are given the physical ability to move left and right, forward and backwards. As input, they have five laser sensors which protrude from the front of the agents and are rotated around the agent to give the agents a 100-degree view of the surrounding world.

As can be seen from the five individual learning runs that are shown within Figure 4.14, during the early stages of learning, all three agents struggled to reliably move out of rooms r_A, r_B, and r_C this is expected due to the constraints and the agents interfering which each other within this initial area. However, it should also be noted that within the initial stages of deep MARL, it is unlikely to find meaningful behaviours and data due to the neural network having to collect experiences to refine its behaviour using batches. The system then learned to collect reliably from rooms r_I and r_E after 450 learning episodes, and this may be due to the relatively large batch sizes that the system uses to update its neural networks. The agent, which was given flexible constraints to move from rooms r_A, r_E and then to rooms r_G and r_H struggled to explore reliably around hallway h_9, possibly due to lower exploration rates and the batched

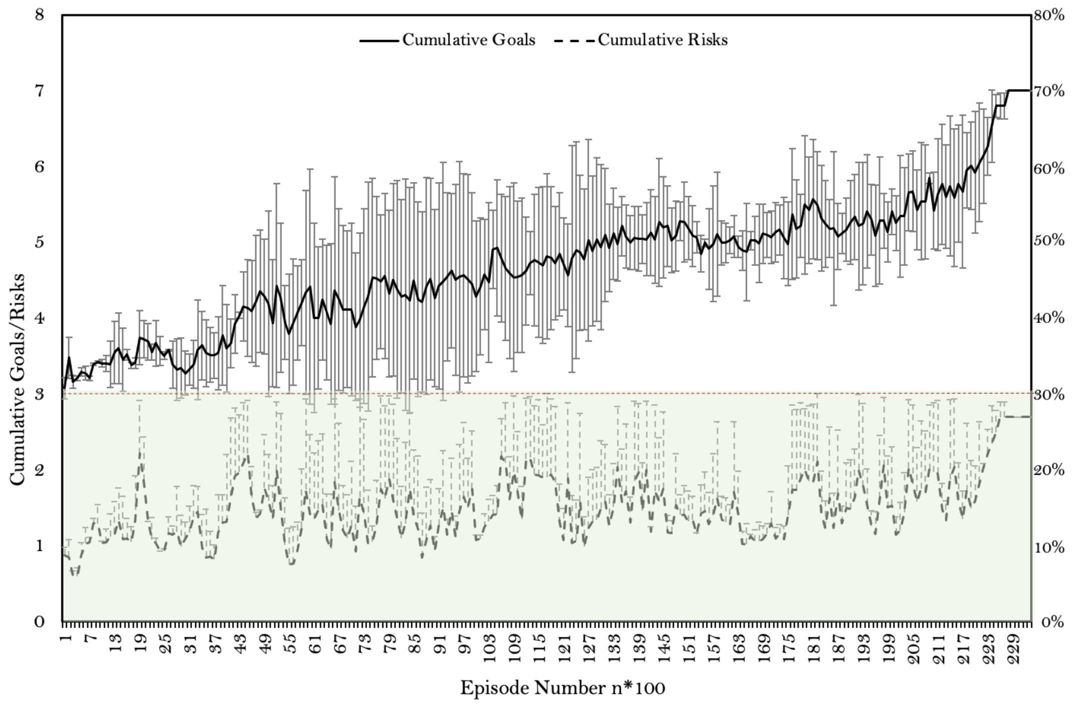


Figure 4.14: Results of Five Learning Runs of Deep AMARL. Where n is the Episode Number Shown on The Graph.

nature of neural networks at the point of encountering said hallway and as such the agent had a slow learning process from episodes 5500 to 17300 before reliably the system began to collect all seven possible goals. As is expected with any reinforcement learning process, the behaviour between episodes is erratic, meaning the expected rewards and risk can vary drastically or minimally. The goals received generally stayed within a margin of one for each episode, meaning the system would collect one less goal or one more than expected at the time of learning. However, the same can not be said for the risk involved. The risk levels changed erratically throughout all learning periods due to the learning process's stochastic nature. Despite the erratic changes, it can be seen that the safety requirements of the mission were never neglected, with the system never exceeding a 30% chance of being caught.

Over five learning runs, the deep MARL system continuously found all seven available goals, reaching its functional requirements while never breaching safety requirements. Demonstrating that the safe abstract policy and its assurances were upheld due to the ADMP being constructed correctly. This experiment shows how

AMARL can be utilised with deep learning techniques by creating an abstracted AMDP from the problem domain.

4.4 Summary

This Chapter on the safe MARL approach known as AMARL displays how it can be utilised to meet safety and functionality requirements with varied MARL techniques and systems. This approach consists of four stages: three stages involve creating and analysing an abstracted version of the problem domain, and one stage involves constraining the low-level problem domain. Before AMARL can be utilised, it must be established that enough preliminary information is known about the environment and the mission needs. While this preliminary does not have to be extensive, it must include all safety and functional requirements information.

1. Stage one involves defining a set of functional objectives and safety constraints that align with the mission requirements. Next, the domain must be analysed to determine which transitions and states are vital for inclusion within an AMDP. Those included need only to contain information that directly relates to these functional or safety requirements, such as transitions that result in moving rooms, but not the states within said rooms.

2. Stage two consists of the formal definition of an AMDP, utilising the decisions made in stage one on how to abstract the states. This AMDP is constructed in the PRISM language and can be accomplished manually or using this Chapter's Algorithm. This algorithm allows the automated construction of a formal AMDP with minimal input. The AMDP also consists of reward structures used to track objectives and safety concerns throughout the model analysis in stage three. This stage allows efficient analysis of the problem domain and also allows analysis of the problem where reward or transition functions are unknown.

3. Stage three involves running quantitative analysis over this AMDP, which is guided using the functional and safety requirements established in stage one within the form of PCTL. This quantitative analysis will synthesise paths through the model that meet both these functional and safety requirements, known as

safe abstract policies. While there is a possibility that no safe abstract policies will be synthesised due to the functional and safety requirements, this is a clear message to not deploy the system.

4. Stage four consists of choosing a safe abstract policy most appropriate for the system's needs and using this safe abstract policy to constrain the MARL system. This removes a lot of complexity of safe MARL by simply allowing MARL to run traditionally under these assured constraints. This allows agents to enter risky states or make risky transitions but only within the guidelines of the constraints.

AMARL was evaluated in a ROS-based patrolling simulator and the Unity games engine. These were used to demonstrate the flexibility of the plugin style approach of AMARL by showing it being utilised with three different types of traditional MARL techniques and a deep MARL technique, as well as homogeneous and heterogeneous systems. These evaluations showed that AMARL could be used to allow safe MARL to occur while being assured to meet or improve upon both functional and safety requirements during and after the learning process.

The first limitation of AMARL revolves around inaccuracies in the initial two stages. The assurances provided by the safe abstract policies that are synthesised by quantitative verification only hold if the AMDP is constructed correctly. Therefore, no guarantees can be made regarding the safety of the MARL system. This issue is addressed in chapters 5 and 6, which attempt to correct issues in the AMDP description manually and autonomously without restarting the lengthy MARL process. The second limitation of AMARL relates to the synthesis of safe abstract policies, as there is no guarantee that any policies will be produced after a time-expensive analysis. However, while this is a limitation, it is also a valid outcome that states that deploying the safe MARL system would not be safe.

Chapter 5

AMARL with Partial Policy Reuse

In Chapter 4, we introduced the abstract multi-agent reinforcement learning approach (AMARL), which allows for the synthesis of safe abstract policies using safety constraints and efficient quantitative verification. However, when an AMDP fails to sufficiently represent the real world, e.g. due to changes over time, the information about the low-level domain that the MARL system learned is rendered obsolete, and guarantees on safe behaviour are invalidated.

A significant issue with addressing inaccuracies in the AMDP is that the time-consuming MARL process must be restarted from scratch. This chapter provides an overview of AMARL with Partial Policy Reuse (AMARL-PPR) and an approach which reduces the time and cost related to rerunning long MARL processes. In addition, AMARL-PPR increases learning speed by reusing aspects of previously optimised or sub-optimised policies when changes to the AMDP are required. This approach also reduces the amount of manual effort necessary to reuse partial policies.

5.1 Introduction

This Chapter introduces the AMARL-PPR process, as shown in Figure 5.1. The proposed process is initiated when the domain expert identifies that the formal assurances of safety and functionality requirements no longer hold.

When we consider the initial AMARL approach as described in Section 4, we see that the formal assurances that the system meets safety and functional requirements are supplied through the synthesises of safe abstract policies using quantitative verifi-

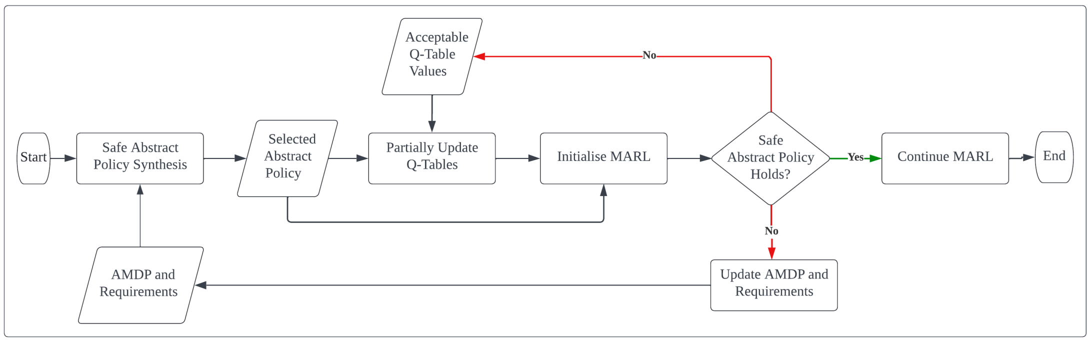

Acceptable Q-Table Values
No
Start
Safe Abstract Policy Synthesis
Selected Abstract Policy
Partially Update Q-Tables
Initialise MARL
Safe Abstract Policy Holds?
Yes
Continue MARL
End
No
AMDP and Requirements
Update AMDP and Requirements

Figure 5.1: Partial Policy Reuse with AMDP and Q-Table Value Reuse.

cation. However, using quantitative verification in this context can be computationally expensive for large models. In order to mitigate this limitation, the problem may be abstracted. This abstraction is driven by safety and functionality properties such that the resulting AMDP is reduced to a size that can be efficiently verified on the available hardware and in a timely manner. In addition, abstracting domain information for the AMDP in this way can provide a high-level view of the problem, thus allowing AMARL to be utilised with partial domain knowledge. With AMARL, any information that does not directly relate to a safety or functional objective can be discarded. When this abstraction is completed correctly, meaning all relevant information is captured for an unchanging domain problem, then the AMARL process produces formal assurances for functionality and safety that will hold true.

The issue with AMARL utilising AMDPs comes from the entirely plausible event of an abstraction being completed incorrectly due to incomplete knowledge or the problem changing over time. This would mean that the formal assurances that are synthesised through quantitative verification will be acquired from an inaccurate model of the problem, resulting in assurances that do not hold in reality. If this is the case, then the safe abstract policies used to guide the constraint of the AI agents may restrict the agents from completing goals or allow them to act recklessly.

When this occurs, AMARL must be utilised again from stage one, and the lengthy learning process must be restarted. When this learning process is restarted, everything the system has learned is discarded as the information from the low-level domain that has been learned may no longer be useful when the new safe abstract policies are synthesised.

AMARL-PPR overcomes these issues while meeting two main aims. The first of these aims is to reduce the amount of information which is discarded between runs. AMARL-PPR leverages previously learned knowledge to reduce the learning time in subsequent learning runs. Domain experts can partially reuse existing policies by analysing changes within the problem domain to determine which areas remain unchanged and which areas are likely to be impacted in terms of safety or functionality. This allows those parts of the policy previously developed for use in these unchanged areas to be reused. Learning is then confined to the areas in which the change has impacted. The second aim is to reduce the amount of manual manipulation and decision-making required by the domain expert when reusing learned information.

The next section describes the AMARL-PPR approach, which takes two safe abstract policies and a matrix data structure storing a previously learned policy as input.

5.2 Approach

AMARL-PPR builds upon the standard AMARL process, initially introduced in Figure 4.1, with policy reuse connected directly to stage four of the original process. An example of how AMARL and AMARL-PPR are interconnected can be seen in Figure 5.2. As the MARL system is learning within the constrained low-level environment or after the learning has concluded, the AMARL-PPR process can be initiated. AMARL-PPR involves six potential steps, beginning with a question posed to the domain expert: "Does the safe abstract policy still hold?". This question requires the expert to evaluate whether the problem domain is still in a condition that satisfies the original modelling assumptions. This will result in one of two outcomes.

The first outcome results when the domain expert identifies that the safe and functional assurances still hold, having analysed the environment to see that no change has occurred that materially impacts the potential functionality or safety of the system. Step 2 (continued MARL) can take place in this case, and the MARL system is free to continue learning.

The second outcome results when the domain expert identifies that the AMDP does not correctly capture the low-level domain and that the safe and functional assurances no longer hold. This triggers the third step (Update AMDP and Requirements), which involves the update of the AMDP to match the current state of the lower-level

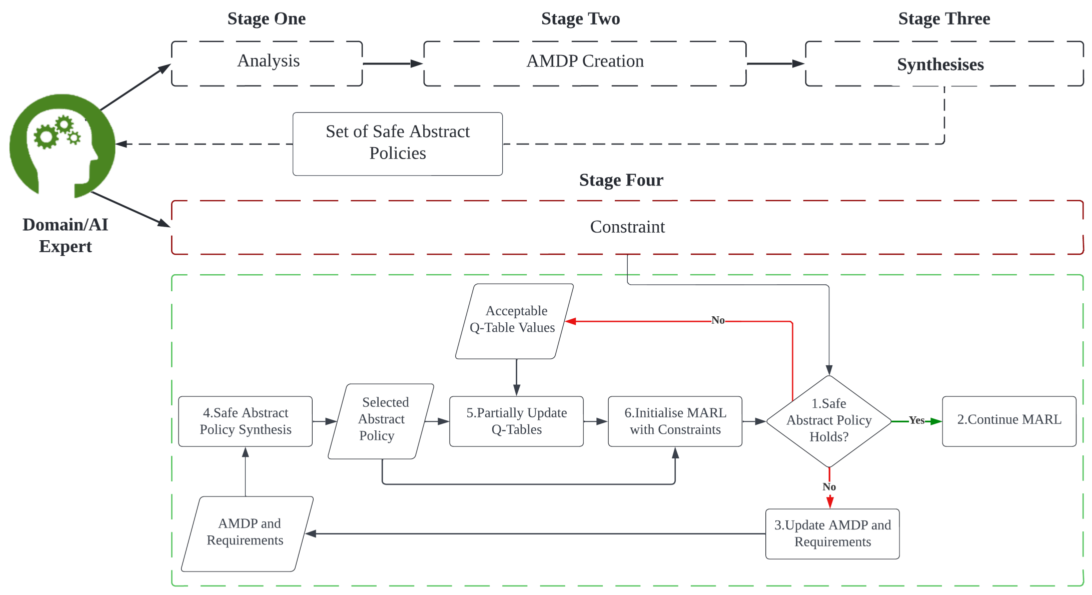

Stage One
Analysis
Stage Two
AMDP Creation
Stage Three
Synthesises
Set of Safe Abstract Policies
Stage Four
Constraint
Domain/AI Expert
Acceptable Q-Table Values
No
4.Safe Abstract Policy Synthesis
Selected Abstract Policy
5.Partially Update Q-Tables
6.Initialise MARL with Constraints
1.Safe Abstract Policy Holds?
Yes
2.Continue MARL
No
AMDP and Requirements
3.Update AMDP and Requirements

Figure 5.2: Partial Policy Reuse incorporated within the AMARL Approach with the policy reuse process shown within the green dashed box

domain and, if required, the safety and functional requirements. This will involve manually changing the AMDP model within the PRISM language and the safety and functional requirements within the PCTL syntax. This allows for a degree of flexibility for the domain expert following the AMARL-PPR process, allowing the mission and safety requirements to be reanalysed and redefined to reflect the current state of the problem they face. At the same time, the domain expert must also be aware of and store the policy information (in the form of Acceptable Q-Table Values) that was learned in previous runs of the AMARL process.

In step 4 (safe Abstract Policy Synthesis), the updated AMDP is analysed with respect to the safety and functional requirements encoded as formal logic using quantitative verification. This results in the synthesis of safe abstract policies which are guaranteed to allow the system to meet safety and functional requirements. Once a safe abstract policy has been selected, it can be directly compared to the previous safe abstract policy, as shown in Figure 5.3. Here the transitions that remain the same between the two safe abstract policies can be used to inform the rest of the approach within step 5. In step 5 (Partially Update Q-Tables), the differences between the new and old safe abstract policies can be used to determine which aspects of the policy learned can be reused and which need to be discarded. This allows for selected aspects of the Q-table values to be copied over for the future MARL system to use as a foundation. Once the partial update of the Q-tables has been completed, then step 6 can be followed, which involves the new safe abstract policy being used to guide the constraint of the lower-level problem domain. Learning can continue with partial knowledge from the old MARL learning run.

At step 6, the MARL is initialised under constraints that are guided by this new safe abstract policy. After step 6, the process reaches the original question posed, allowing the process to be repeated as many times as needed to produce a safe MARL policy that meets safety and functional requirements; however, it may be the case that this process is only needed once. The process can be stopped when the question reliably produces the answer that the safety and functional assurances hold. However, suppose an event occurs where the domain expert notices an issue after a prolonged period. In that case, the process can also be initiated on the condition that the previous safe abstract policy and lower-level policy details are available.

The policy reuse approach is encoded in Algorithm 6 and reduces the manual manipulation required by the Domain Expert. Throughout the description of this algorithm, the tabular data structures are discussed as Q-Tables and Q-values. This algorithm compares the problem domain's original safe abstract policy with the newly defined safe abstract policy. Similarities between these abstract policies indicate there are learned values that the algorithm can retain. Therefore, it is possible to have the Q-values of the agents copied over from the previous Q-Tables into the newly initiated Q-Tables for the different agents within the system. In lines 2 to 26, the algorithm loops through the previous and the new safe abstract policies, which share a format with the example in Figure 5.3. If identical instances exist in these abstract policies, then the algorithm will find them (line 4). Next, the string is split to allow the abstract state, the state transitioned to, and the agent making the transition to be identified. Once these have been identified, the low-level state action pairs from the previous Q-tables are copied to the new Q-tables. This transition is facilitated in lines 8 to 23 and relies on identifying the low-level states that are abstracted in both the initial and transition states. Finally, depending on the agent value collected in line 7, the Q-tables of the related agent is updated.

Algorithm 6 Policy Reuse Algorithm

```
1: Function{PolicyReuse}{AbstractPolicyOld, AbstractPolicyNew}
2: for i in AbstractPolicyOld do
3:   for x in AbstractPolicyNew do
4:     if i == x then
5:       abstractState = x.split('_')[0]
6:       abstractState' = x.split('_')[1]
7:       Agent = x.split('_')[2]
8:       for all low-level (s,a) in abstractState do
9:         if Agent == '1' then
10:          QtableOne(s,a) = QtableOne*(s,a)
11:        end if
12:        if Agent == '2' then
13:          QtableTwo(s,a) = QtableTwo*(s,a)
14:        end if
15:      end for
16:      for all low-level (s,a) in abstractState' do
17:        if Agent == '1' then
18:          QtableOne(s,a) = QtableOne*(s,a)
19:        end if
20:        if Agent == '2' then
21:          QtableTwo(s,a) = QtableTwo*(s,a)
22:        end if
23:      end for
24:    end if
25:  end for
26: end for
27: EndFunction
```

With this algorithm, the domain expert has only to supply the original Q-table, the original and new safe abstract policies, and the learned values that can be automatically copied over. This reduces the amount of manual manipulation that is required, which, depending on the size of the tabular data structure, could be relatively extensive.

Initial Policy	New Policy
R1_start_roomD	R1_start_roomA
R2_start_roomB	R2_start_roomB
R1_roomD_roomE	R1_roomA_roomC
R1_roomE_roomF	R1_roomC_roomE
	R1_roomE_roomF

Figure 5.3: Comparison of safe abstract policies to dictate which Q-values will be reused or discarded.

5.3 Evaluation

In this section, the functionality of AMARL-PPR is evaluated with respect to the speed of the learning process and how much of an impact AMARL-PPR has when compared to AMARL as presented in Chapter 4. In order to facilitate this evaluation, a grid-world-based domain was adapted from previous safe reinforcement learning research [87] using the Unity Games Engine. This grid-world domain, as shown in Figure 5.4, is framed as a simplified search and rescue setting, with a system comprising of two homogeneous agents making use of Independent-Q learning. The agents must search a building in which there are risks to the system safety due to building instability. The objective is to find at least four flags whilst avoiding areas of instability such that the risk of damage occurring remains at, or below, 20%. The domain contains a 21 by 15 grid, where each grid space is defined as being an individual state that one of the agents can occupy. When an agent is within a state that is surrounded by unoccupied grid squares, the agent has four movement actions allowing it to move orthogonally through the space. In such a setting, it is not unforeseeable that the physical characteristics of the domain space could change. In emergency scenarios, walls may collapse, blocking access to rooms or creating new access routes. These physical changes result in changes to the probabilities concerning risk to agents as well as states being added or removed from the reinforcement learning model.

This problem domain was created to allow a research question to be answered, which relates to how AMARL-PPR can be used to make use of otherwise wasted

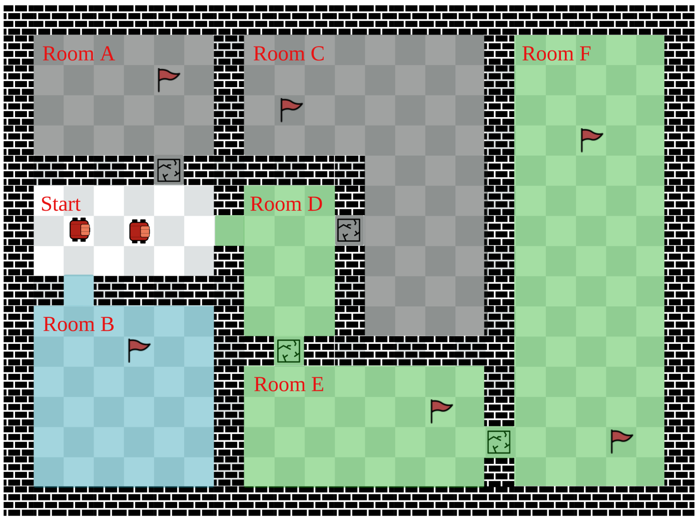


Figure 5.4: Search and Rescue inspired Domain with Objectives (Flags) that must be reached and unstable ground (Shown as cracks within doorway squares) that must be avoided as much as possible. Synthesised safe policy overlaid in green and blue, donating differing agent abstracted actions, with grey showing states that either agent never visits.

learned information and policies from MARL which took place under constraints and assumptions which become invalidated. Depending on the similarities between the original information concerning the problem domain and the updated information of the same problem domain, how much more efficient is AMARL-PPR in comparison to AMARL?

In order to meet its first aim, the AMARL-PPR process should greatly increase the MARL system's learning speed. This aim should be achievable based on the number of learned values within the tabular data structure that can be reused in specific scenarios. Though, it is expected that the efficiency of AMARL-PPR will depend entirely on the similarities between the original information about the problem domain and the new information gained about the problem domain.

We started our evaluation by applying the initial AMARL approach within this domain. First, the problem domain was analysed, identifying seven abstracted states, one per room, with the abstracted transitions relating to an agent moving between rooms. It was then possible to attribute risks and goals to these transitions based on how many flags can be collected per room and how many dangerous areas are

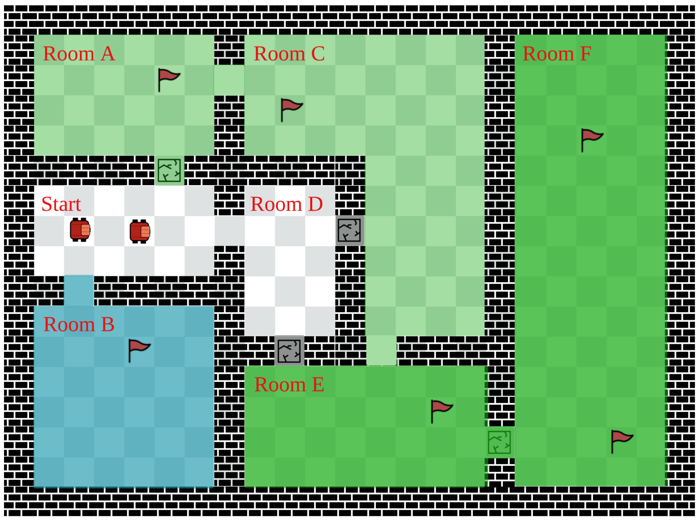


Figure 5.5: domain updated with knowledge of transitions between rooms A and C, and rooms C and E. Policies for both agents overlaid in green and blue with areas where Q-values have been reused shown in a darker shade. Primarily these Q-values are reused in rooms B, E, and F.

passed to make the said transition. The AMDP was then constructed based on this analysis by the domain expert and constructed using the PRISM language. Quantitative verification was then applied to this AMDP. Finally, the safe abstract policy was used to guide constraints for the MARL system and allowed to learn for over 1000 learning episodes, where each episode contained 200 actions. This equals a maximum of 200,000 actions that can be taken over the entire learning period within a domain with a total of 264 states and 1056 possible transitions without considering the multi-agent aspect. The results of the safe abstract policy, chosen for this domain, divide the actions of the two agents as shown in Figure 5.4, where the first agent's abstract policy results in the agent visiting those rooms shown in green whilst the second agent is shown in blue. This task segmentation greatly reduces the complexity of the MARL problem by reducing the amount of interaction between the agents during the learning processes. However, as has previously been shown with AMARL, a compromise has been found such that whilst the minimum permissible number of flags are collected, two rooms are never visited. This ensures that the probability of an agent being damaged is 20% or below.

When looking at the policy shown within Figure 5.4, it is important to note that the safe abstract policy has removed access to rooms A and C entirely. This means that no prior information will be able to be collected for the states and transitions within these two rooms. However, the optimal policies for rooms B, D, E, and F, within their current condition, can be utilised once found.

In order to evaluate AMARL-PPR, three sets of learning runs had to be completed. The first involved the MARL system learning within the initial unchanged domain, allowing an efficient policy to be learned. This learning run forms the first part of the process, collecting information with inaccuracies and requiring a new policy to be produced, rendering the previous time and resources spent on the initial learning run lost if standard AMARL was being utilised. The second and third set of learning runs, containing five learning runs each, act as the experimentation that will allow a direct comparison between AMARL and AMARL-PPR when faced with this occurrence and how efficient AMARL-PPR is in terms of learning time.

Having obtained a policy using AMARL, we turn our attention to policy reuse when changes to the environment are identified. The altered domain we consider is shown in Figure 5.5. Here two new passageways are created, possibly by collapsing walls so that agents may move between rooms A and C and between C and E. Though blockages have also been added to the domain, as shown by shaded grey squares within the transition spaces. In this way, we can observe that it becomes possible for all the flags to be collected without additional risk. We note that while this is an 'ideal' setup, the efficiency and usefulness of AMARL-PPR depend entirely on the similarities between the prior learning environment and the new learning environment. If the similarities are extensive, then the utility of AMARL-PPC will be significant due to similarities in the synthesised safe abstract policies. If there are few similarities or none, the utility will quickly reduce or become non-existent.

The safe abstract policy, which was synthesised for the new domain, is shown in Figure 5.5, and it can be seen to be altered from the original policy in three ways. First, actions within room D are no longer necessary as a blockage has occurred, meaning transitions from room D to rooms C and E are impossible. Second, agent one (green) is now allowed access to rooms A and C, providing an alternative route to room E, which results in more goals being collected for the same degree of risk. This means it is impossible to reuse Q-values for states within the starting room and room D as the desired direction of travel has altered considerably. Third, the Q-values for Rooms A

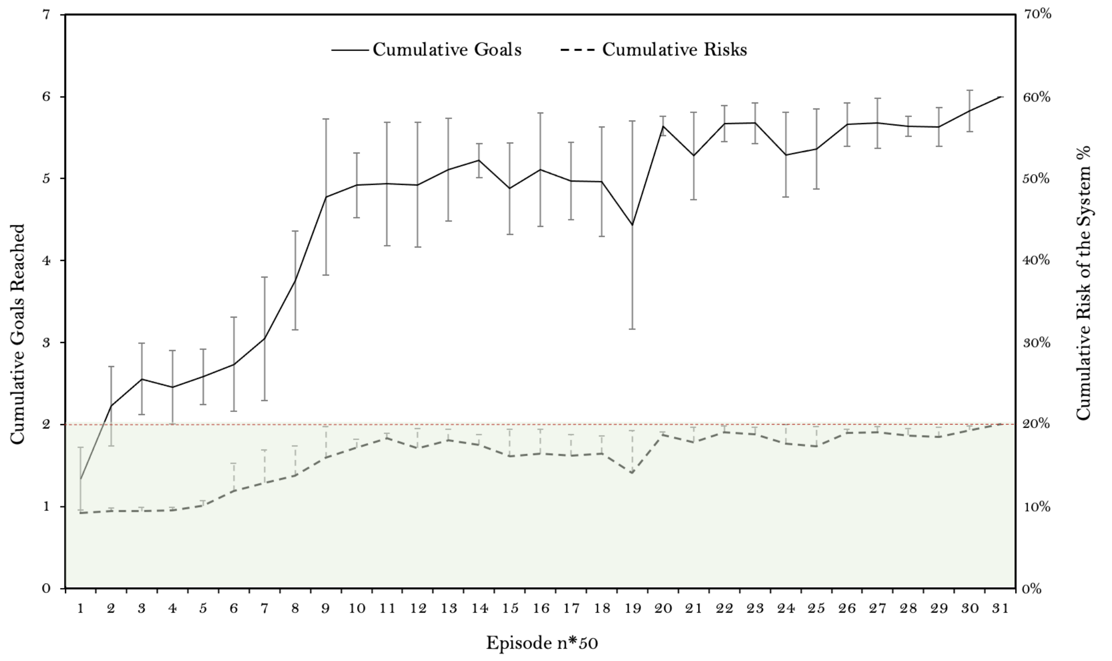


Figure 5.6: Performance of MARL utilising only AMARL

and C are also left unpopulated due to no prior knowledge being gained from them because of the previously selected safe abstract policy restricting the agents' access to them. However, due to similarities in the domains, the Q-values for rooms B, E, and F can be directly copied. Due to the nature of the domain, no further exploration is required within these rooms as the optimal policy was found. The circumstances of these rooms and the related Q-values mean that the learning space has been reduced from 264 states with 1056 possible transitions to 131 states with 524 available actions. With the complexity of the domain being reduced by more than half, this should reduce the required learning time.

To compare the potential utility of AMARL-PPR, we used AMARL without policy reuse to learn the safe policy for the altered domain, and the results are shown in Figure 5.6. Throughout the five learning runs, it took, on average, 450 episodes for the system to find all six available goals. However, it took a further 550 episodes for this to become reliable, meaning a static policy is likely to enable all six goals to be reached when one is randomly selected from an episode from this point on without the stochastic learning process. This same trend in behaviour can also be viewed in the

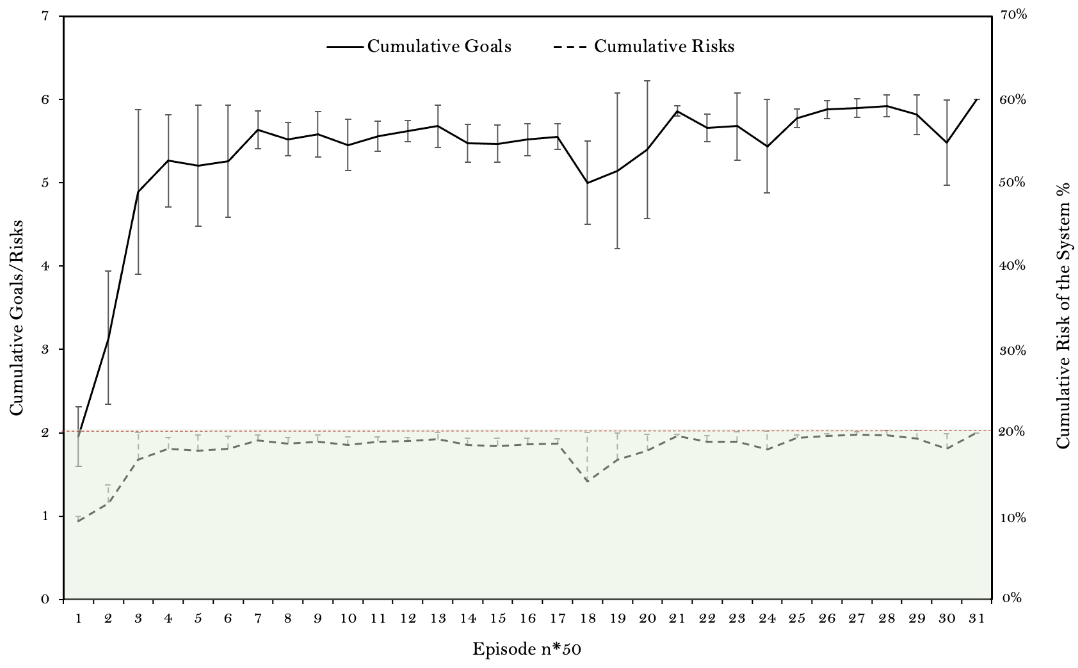


Figure 5.7: Performance of MARL utilising AMARL-PPR

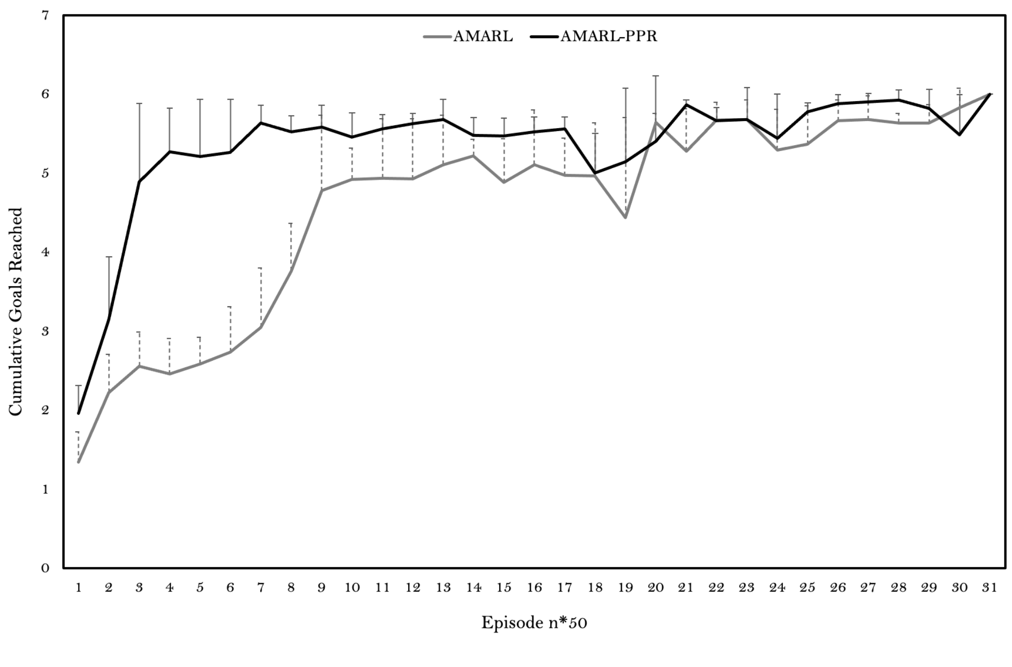


Figure 5.8: The Cumulative Goals Collected by a MARL System Utilising AMARL and then AMARL-PPR

risk plot, with the probability of damage reaching 20% simultaneously. This is because the probability of an agent succumbing to damage has a constraint of 20%, and due to the nature of the domain, once a risky transition has been allowed, the number of goals that may be reached increases. Conversely, if a risky transition is not taken, then the number of goals reached decreases. This explains the similar trends between the two lines of plotted data. At the end of the learning run, where stochasticity is removed, it can be seen that all six goals are found consistently.

The results associated with using AMARL-PPR are then shown in Figure 5.7. For ease of comparison, we show the cumulative goals achieved by episode for both AMARL and AMARL-PPR in Figure 5.8. The primary aim of AMARL-PPR is to reduce the time taken to produce a policy that meets the specified functional and safety requirements. The AMARL system took 450 episodes to find the total number of goals available to it and reliably found these six goals in episode 1000. However, the AMARL-PPR system learned to find the total number of goals in 150 learning episodes, and from episode 350, taking into account the exploration rate, reliably collected all six goals, with only slight variations around episode 900. In order to ensure that the learned policy of the AMARL-PPR system could reach the requirements, the agents' policies within the system were tested without exploration in episodes 350 and 400. As a result, they were proven to meet functional and safety requirements, reaching all six goals repeatedly.

The results obtained for AMARL-PPR show that the approach can reduce the required learning time compared to the reapplication of the AMARL process without partial policy reuse. When an abstracted state can be removed from the quantitative analysis process or the learning process, it can remove a large number of state transition pairings from the MARL domain, which significantly simplifies the problem.

5.4 Summary

AMARL with Partial Policy Reuse (PPR) is an extension to AMARL that allows for the automated reuse of partial policies obtained from previous learning runs when there is AMDP no longer adequately represents the state of the real-world context. This error in representation could be present in the AMDP's initial construction or due to changes to the domain, which are identified during the learning process. When such changes are identified, it is possible that the assurances delivered by AMARL regarding functional and safety requirements are undermined. Traditional approaches, including AMARL, require the learning process to be restarted from scratch when such errors are discovered. AMARL-PPR, by contrast, allows for the use of an automated AMDP generation algorithm and an automated policy reuse algorithm to semi-automate the process of correcting errors within the AMDP and selecting which Q-values to be reused based on the previous and new safe abstract policies used to constrain the MARL system. This semi-automated process reduces the time the learning process takes by removing previously optimised states from the learning process.

AMARL-PPR is demonstrated using a domain comprising of a grid world containing seven areas, within which multiple grid spaces were abstracted into single states for learning. Within the domain, two homogeneous agents undertook a flag collection task while complying with strict safety constraints. AMARL-PPR simplifies the MARL learning problem by removing previously optimised states from the learning process and by populating states with non-optimised values, which assists the learning process within these abstracted states. From this evaluation, it is observed that AMARL-PPR can greatly increase the learning speed for problem domains that have been partially explored previously.

AMARL-PPR has recognised limitations, which are as follows, the similarities between the old and new safe abstract policies determine the utility of AMARL-PPR. If there are no similarities between these safe abstract policies, then no parts of the previous policy can be reused. Also, in its current state, the algorithm suggested in this chapter is only applicable when tabular data structures are used to store learned information. Lastly, AMARL-PPR has only been evaluated using MARL techniques that utilise tabular data structures for storing learned information, such as Q-tables. This evaluation could be expounded to include other kinds of partial policy reuse, such as Deep MARL techniques.

Chapter 6

AMARL with Adaptive Constraint

In chapter 4, an approach for safe MARL named Assured Multi-Agent Reinforcement Learning (AMARL) was introduced, which utilises quantitative analysis to help guide the constraint of MARL systems during and after learning while providing assurances that safety and functional requirements will be met. However, if the AMDP does not accurately capture all the safety and functional requirements of the low-level domain, then this costly AMARL process would have to be restarted from scratch. AMARL Partial Policy Reuse (AMARL-PPR) was created in chapter 5, allowing inconsistencies between an AMDP and a problem domain to be corrected by information obtained after learning has begun. This is accomplished while allowing the partial reuse of learned policies from MARL which took place under outdated assumptions, resulting in more efficient learning. However, despite AMARL-PPR being more efficient than AMARL, it is currently only available offline, meaning the process can only take place when the learning process is stopped.

Within this chapter, Adaptive Constraint (AMARL-AC) is introduced. AMARL-AC was developed to allow a MARL system to adapt to changes within the low-level domain at run-time while still under constraints that are guided by quantitative analysis. This adaption is facilitated by quantitatively guided constraints being changed automatically by the MARL system during runtime. AMARL-AC uses the previously introduced algorithms within an automated process to allow a MARL system to adapt to changes without further input from a domain expert.

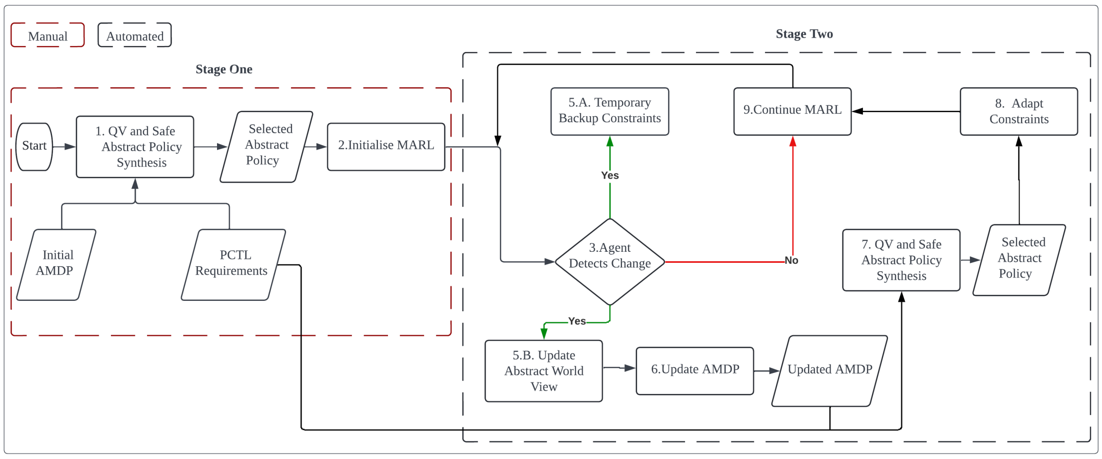


Figure 6.1: Adaptive Constraint Process.

6.1 Introduction

This Chapter introduces the AMARL-AC process, which aims to allow a MARL system to adapt to identified changes in an automated fashion. As can be seen in Figure 6.1, AMARL-AC consists of two stages, an initial manual stage which involves the domain expert providing information that is required for the second automated stage to function correctly. When this automatic stage is operational, the MARL system is able to identify inconsistencies between the current known AMDP and the low-level problem domain related to safety or functional requirements. This will then allow the AMDP to be updated accordingly, quantitative analysis to be utilised, and the AMARL system to update its own constraints based on the received safe abstract policy during run-time.

AMARL-AC was developed due to a significant issue within safe reinforcement learning and safe MARL, which was discussed in Chapter 5. This issue arises from inaccurate initial knowledge or changes which occur within the learning domain during or after the learning process and is likely to be encountered in practical safety and mission-critical settings. AMARL-PPR, introduced in Chapter 5, attempts to mitigate this problem. However, it is limited to reinforcement learning techniques that utilise tabular data structures to store information and is an entirely offline approach. Being offline means that the learning process must be completed, or the domain expert must halt the learning process when issues arise. Such issues may not be immediately apparent, resulting in wasted computational effort.

In order to build upon the AMARL-PPR, an approach is required which allows for varied learning techniques to be used and for the MARL system to react promptly to unexpected changes and outcomes within the environment. In this Chapter, AMARL-AC is introduced, AMARL-AC is an approach that allows the AMARL process to be autonomously triggered and utilised while learning can continue with minimal disruptions. Furthermore, by temporarily constraining learning to areas that have not changed, the MARL system can keep learning without the threat of breaking safety requirements and await the new quantitatively verified constraints to be synthesised.

AMARL-AC is broken into two stages, as shown in Figure 6.1, an initial manual stage followed by a second automated state. The manual stage, as with AMARL, requires preliminary information to be supplied. This preliminary information includes the expected abstracted information on states, transitions, and functionality and safety requirements (encoded as PCTL), the system's initial state, and the number of agents utilised by the system. This information is supplied directly within Algorithm 3 of Chapter 4 and is assumed to remain unchanged throughout the learning process.

The automated second stage consists of the MARL system checking that a change that affects the functionality and safety requirements has not taken place within the low-level domain. If a change has taken place, then stage two supplies the MARL system with a safe backup constraint while the AMDP is updated to reflect the change in the lower-level domain. Quantitative verification takes place, and then the MARL system receives a safe abstract policy that is used to guide the MARL system in constraining its own behaviour. Three Algorithms are utilised within this automated stage, two algorithms introduced in Chapter 4 and one introduced later in this Chapter.

This automated stage follows a *monitor, analyse, planning,* and *execute* structure. Firstly, The MARL system monitors the environment, checking for inconsistencies within the domain. Secondly, Algorithm 3 from Chapter 4 generate and analyse an updated AMDPS. Thirdly, Algorithm 7, is used in planning which selects which synthesised safe abstract policy to send to the MARL system. Lastly, the agents are supplied with the said safe abstract policy, which guides the MARL system in executing their new constraints.

The main focus of this approach is to alter information about states, transitions and rewards and risks in an automated fashion. Allowing the MARL system to update its learning constraints mitigates the limitations and issues presented by AMARL-PPR, resulting in a more flexible approach. However, AMARL-AC requires the MARL system to be able to recognise and identify goals, risks, transitions, and states.

6.2 Approach

Our two-stage approach is shown in Figure 6.1. The first manual stage begins with a basic implementation of AMARL, as seen in step 1 and step 2, which involves an AMDP of the low-level domain problem being created and subjected to quantitative analysis and the constraint of the MARL system guided by the synthesised safe abstract policies and the initialisation of MARL. Within this stage, it is fundamental that certain information regarding the MARL system and the domain problem is stored in an accessible place ready for use within the automated stage. This information includes the PCTL encoded requirements, MARL system size, and the initial states. The automated stage uses three individual algorithms, one of which is Algorithm 3 from Chapter 4, which allows the automated generation of an AMDP. It is suggested that this fundamental information is stored within this algorithm as variables. Due to Algorithm 3 being used within AMARL-AC, it is required that the domain expert create a data structure which will serve as an input into Algorithm 3 to facilitate the automated generation of an AMDP. This data structure is known as a Matrix WorldView multi-dimensional array which was introduced in Chapter 4 and illustrated in Figure 4.2.

This Matrix WorldView data structure is a multi-dimensional array containing abstracted information about the low-level domain, such as abstracted states, the transitions between these abstracted states, and the goals or risks to which these transitions will lead. The Matrix WorldView is integral to the AMARL-AC approach, which is underpinned by the ability of the agents to detect changes within their environment and identify how these changes are linked to the current AMDP. The existing AMARL-AC approach must be able to perceive risk and goals and rely on the agents within the system knowing their current location and where their location is concerning the abstracted transitions. This knowledge allows repeated queries to determine if a transition is new or has been removed. For example, in a grid-based world, an agent who is aware that it is within the state (3,2) will also know that state

(3,2) is within Room A. From these two pieces of information, the agent will be aware that it is within Room A and may query how far it is to a known transition out of the room. The agent will be able to update the Matrix WorldView and allow the synthesis of an updated AMDP.

When the manual stage is complete, the second automated stage can begin. The automated stage begins with step 3, which determines if an (Agent Detects Change). This step is part of a looping process which constantly checks if an agent has detected a change within the low-level domain while learning. Each agent monitors the environment to identify changes by comparing observations with its current understanding of the environment as encoded in the Matrix WorldView. If no change is detected, then this means the current safe abstract policies hold, and MARL can continue within step 9, which then loops back to step 3 to check if any changes have been detected. However, step 5-A (Temporary Backup Constraints) and step 5-B (Update Matrix WorldView) is triggered if a change has been detected.

Step 5-A (Temporary Backup Constraints) consists of the MARL system being allowed to continue learning within backup constraints before new constraints are applied. These backup constraints contain the MARL system to the areas of the domain which have not been directly affected by the changes that have been detected. This means that the MARL system can continue to learn while not being at risk of learning overly risky behaviour.

Step 5-B involves the MARL system updating the Matrix WorldView to contain the new information. This utilises the abilities that are a prerequisite of the AMARL-AC approach. Which, as stated, is the ability to perceive risk and goals and to know their current location and where their location is in regards to the abstracted states. This update will supply the MARL system with the ability to synthesise an updated AMDP using Algorithm 3, which is what is required in step 6.

Step 7 (QV and Safe Abstract Policy Synthesis) then takes as input the updated AMDP and the unchanged PCTL requirements, which were supplied in step 1. This involves quantitative verification being utilised on the updated AMDP, guided by the supplied PCTL requirements, and then the synthesis of safe abstract policies. If no safe abstract policies are synthesised, then the constraints will not be updated, and the MARL system will continue to learn under the Backup Constraints implemented in step 5-A. If safe abstract policies are generated, then the policies are converted into readable policies using Algorithm 5, introduced in Chapter 4. Then a new Algorithm, Algorithm 7 compares the generated policies in terms of functionality and safety and selects the most functional option that meets all of the defined safety requirements.

In step 8 (Adapt Constraints), the selected policy is given to the MARL system, and given the readable policy format, the MARL system can execute its own constraint adaption by reading the safe abstract policy and using this to guide which actions should not be taken within the low-level domain. Lastly, the MARL system is allowed to continue learning within the boundaries of its updated constraints.

The algorithms that allow stage two to take place are Python scripts that are callable from within the agents' code classes. The first algorithm used was Algorithm 3, which automatically generates an AMDP based on a Matrix WorldView. This algorithm was adapted slightly for constraint adaption in order to launch the PRISM model checking tool and supply it with the newly created AMDP and the PCTL requirements supplied in the manual stage. Next, algorithm 5 waits for safe abstract policies to be synthesised to translate the safe abstract policies into human-readable and agent-readable formats. Finally, algorithm 7 selects a policy and places it in a folder structure that the agents can access when the update algorithm commences. The agents read this policy and, based on the information contained within it, they constrain their behaviour based on the transitions within the safe abstract policy.

As well as the algorithms presented in Chapter 4, only one additional algorithm had to be created for this process to function correctly. This is Algorithm 7, and it is responsible for comparing and selecting a safe abstract policy after a set of safe policies have been synthesised. This algorithm inputs the matrix world description and the set of safe abstract policies.

The lists named Goal and Risk are declared on lines 2 and 3, and each index of these arrays will store the cumulative number of Goals and Risks that the policy allows. This collection of Goals and Risks is facilitated by the nested For loop structure from lines 5 to 13. Here every policy received as input, the transitions within those policies are individually inspected, and any corresponding goals and risks associated with an individual policy are added to an index within Goal and Risk lists.

Every policy is given an identifying index within these two lists using the integer counter declared on line 4. Finally, the For loop declared on line 14 cycles through the Goal and Risk lists and performs a check to see if the cumulative values for each

Algorithm 7 Policy Selection Algorithm

```
1: Function{PolicySelect}{MatrixDescription, policies[]}
2: Goal[]
3: Risk[]
4: PolicyCounter
5: for each policy in policies do
6:     for i in policy do
7:         S = i.split('_')[0]
8:         S' = i.split('_')[1]
9:         Reward[PolicyCounter] == Reward[PolicyCounter] + MatrixDescription[S, S',
   1]
10:        Risk[PolicyCounter] == Risk[PolicyCounter] + MatrixDescription[S, S', 2]
11:    end for
12:    PolicyCounter++
13: end for
14: for i in range(PolicyCounter)) do
15:    if Goal[PolicyCounter] == CRITERIA and Risk[PolicyCounter] == CRITERIA then
16:        ChosenPolicy = policies[policyCounter]
17:    end if
18: end for
19: EndFunction
```

index meet the criteria that the domain expert sets. Such as the policy with the greatest number of goals without exceeding safety requirements. The policy which meets this condition, specified on like 15, is then selected as the chosen policy and is sent to the multi-agent system.

6.3 Evaluation

In order to evaluate AMARL-AC's ability to adapt to issues with transitions, the domain utilised in the previous Chapter on AMARL-PPR was reused. This domain is the same as in Figure 5.4. However, to demonstrate AMARL-AC, the same adjustments were made that are seen in Figure 5.5 without supplying the system prior knowledge of any changes meaning it will be learning under outdated assumptions. However, as stated previously, AMARL-AC can be utilised to adapt a system to more than transitional changes and has been used to adapt to states being added to the domain. This capability is demonstrated in a preliminary evaluation using the deep learning domain introduced in Chapter 4 and shown in Figure 4.12.

6.3.1 Online Constraint Adaption

In this section, AMARL-AC transition adaption is evaluated regarding its ability to perform in the presence of changes to transitions and transition properties, such as the expected risk or goals collected by making the said transition. The MARL system is initially provided with domain knowledge and constraints for the unaltered domain shown in Figure 5.4. Unlike the experimental setup in Chapter 5, which saw MARL systems given 1550 learning episodes to find an efficient policy, the experimental setup for AMARL-AC permitted the use of 10,000 learning episodes, with 60 available action steps per learning episode. This allows the two-agent homogeneous system ample time to relearn behaviours as it encountered states that presented inaccuracies within its domain knowledge. This updated knowledge is then stored as an abstract matrix world description. It was found that fewer learning episodes increased the likelihood of the system chasing rewards that it had collected based on inaccurate information and not finding all six available goals. It was also necessary to supply the system with a much higher exploration rate to find and collect the information required for the system to perform well.

Within the domain, the agents had to discover that the key transition leading from $Room_D$ to $Room_E$ was blocked and that new transitions that led from $Room_A$ to $Room_C$ and then $Room_C$ to $Room_E$ had become available. This was primarily achieved with one agent, as the other agent was always constrained to collect a single flag found in $Room_B$. This result comes from quantitative analysis allowing one agent to move through an area of risk. Due to the layout of the domain, it served no purpose to allow a second agent through areas of risk apart from unnecessarily increasing the likelihood of damage to the system. The system removed and added these different transitions over time by updating the abstract matrix world description and allowing quantitative verification, policy synthesis, and policy selection to allow further exploration. This exploration was more easily allowed through constraints due to the system not finding any risk associated with the new transitions and given the preliminary knowledge that there was a goal present within the adjoining Room. The system

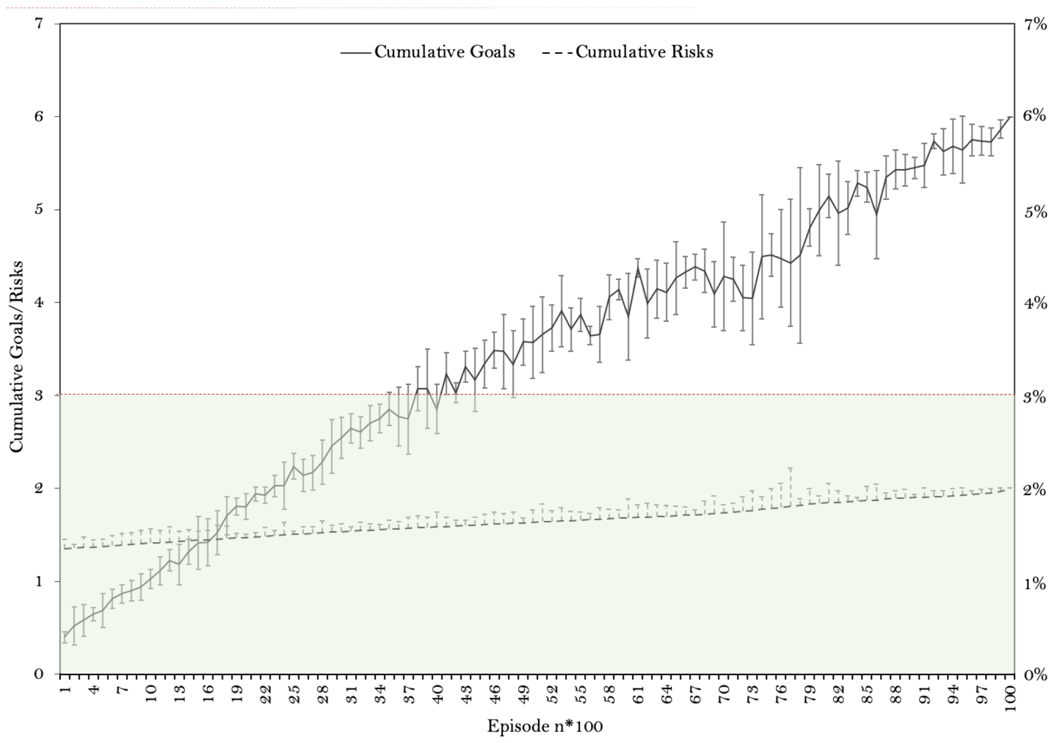


Figure 6.2: AMARL-AC results over 10000 learning episodes

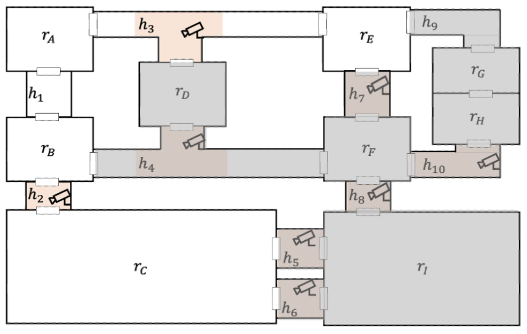
was able to update the AMDP through its abstract matrix world description, adding transition properties such as whether there was risk associated with the transition and if goals were able to be reached.

Given the explained setup above, the results from five learning runs were displayed in Figure 6.2. Given the length of the learning process, the results show a relatively smooth linear relationship between the number of learning episodes and the goals achieved. The slight fluctuations and plateaus that can be seen throughout the learning process can be attributed to the stochastic nature of reinforcement learning. However, it can be seen that these fluctuations become more turbulent at certain points in learning. Specifically, it can be noted that when the system is transitioning from commonly collecting three goals to four, between episodes 3400 to 6100, the system's behaviour was more varied than other episodes. This is due to the system relearning and altering past learned behaviour, which sometimes allowed the agent to collect four goals, and at other times caused the system to use all of its available actions before reaching goal 4. However, as learning continues, it becomes more difficult for the system to unlearn behaviours due to the decaying learning rate that pushes learning systems to exploit gained knowledge more as it learns.

Learning decay may cause a noticeable dip in progression between episodes 6700 to 8500 and then the continued learning towards the cumulative number of goals towards the end of the learning process. In this section of learning, the system would be venturing into $Room_E$ and $Room_F$, which it would have been less likely to visit up to this point due to learning and relearning behaviours. During this point in learning over the five learning runs, there is a relatively high variance between the goals collected over this learning period. This suggests the systems had difficulty finding behaviours that reliably allowed them to collect all six goals. Nevertheless, once this behaviour was found, all five learning runs produced policies that reached the cumulative amount of goals while keeping risk levels below the requirement and at a relatively low variance rate. Despite these positive outcomes, which show a MARL adapting its constraints to meet strict safety and functional requirements without further input from a domain expert, it also shows that a significantly longer learning time is required. Compared to AMARL-PPR, AMARL-AC requires over six times more learning time to produce the same result. This means that this AMARL-AC, with benefits in terms of system autonomy and flexibility, has trade-offs regarding the required time. Lastly, it must be reiterated that the assumption that the goals and risks are perceivable is a necessity.

6.3.2 Online State Adaption

This section evaluates AMARL-AC's ability to allow an AMARL system to adapt to significant structural changes to a domain, such as states that have failed to be incorporated within the AMDP model of said domain. By using the abstract matrix world description, a new state can be added by simply increasing the size of the matrix and populating its transition information to other states. A MARL system can complete this process without human intervention. In order to evaluate this feature of AMARL-AC, the deep learning domain introduced in Chapter 4 and displayed in Figure 4.12 was used. In this evaluation, the domain and system remain unchanged from how it was described in Chapter 4. However, the domain knowledge that is given to the MARL system and AMARL-AC is limited to enable unknown states to be found and for the system to adapt to them. As in the left section of Figure 6.3, the MARL system



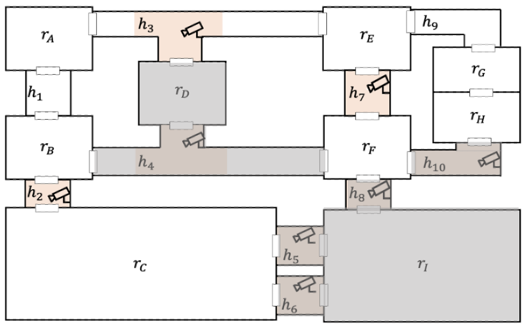


Figure 6.3: The initial domain knowledge supplied to the initial Process (Left). The final learned policy that the MARL system produced using AMARL-AC (Right). Grey areas are unknown/inaccessible to the MARL system.

was initially given information on four rooms, two hallways, and two surveillance cameras, meaning that five rooms, seven hallways, and six surveillance cameras are unknown the MARL system. The system also recognised when an agent was within a state and the name of that state. The system was also given the ability to recognise the presence of a surveillance camera and goals. This was achieved by giving each agent in the system sensors that could retrieve information from the environment. This information was received as a name tag for the item in this simplified domain. In order for AMARL-AC to function, several assumptions are required. The first of these assumptions involves the agents within the MARL system reliably knowing if they are in a state (Room) or a transition (hallway) and also which state and transition. The second assumption is that the agents within the system can reliably find and identify the risks and goals present within the domain. Without this assumption, there is no guarantee that formal verification would synthesise valid abstract policies.

The manual steps of AMARL-AC were completed, similarly to the previous section, and the agents within the system were given the functional abilities to recognise states (a state is added to the AMDP when a new goal is found), transitions, goals, and risks. The system was then allowed to learn for 6800 learning episodes. As can be seen in Figure 6.4, which contains the results of five learning runs, the system quickly, within 1000 learning episodes, learns to take advantage of its initial knowledge and collects three goals fairly consistently, these being in $Room_A$, $Room_B$, and $Room_E$. Eventually, the system also reliably collected the goal in $Room_C$, but this was more

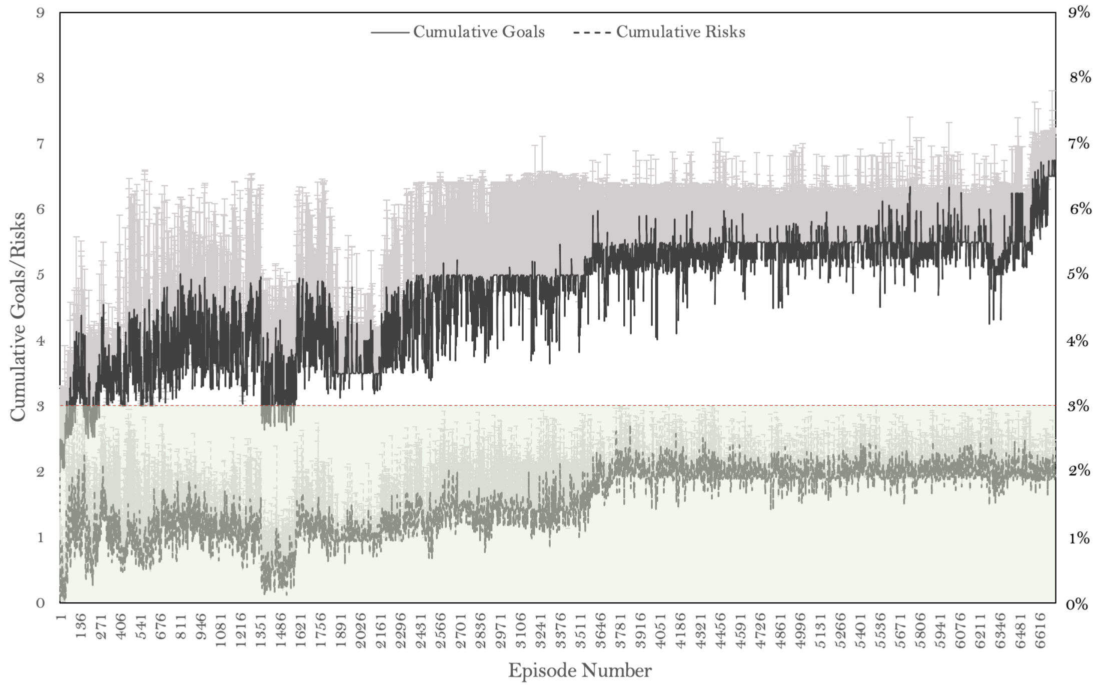


Figure 6.4: AMARL-AC Preliminary Results Over Five Learning Runs. Experimental Results From Allowing Agents to Adapt to Unknown States.

challenging as the goal was not visible until entering the Room, unlike the other known goals. However, between episodes 1000 to 2000, there is a sharp decline in the goals reached, followed by a short plateau, followed by another sharp increase in the goals reached. Between episodes 2000 to 3000, this same trend occurs again, followed by the consistent collection of the cumulative number of goals throughout the rest of the learning process. This trend of sharp declines, plateaus, and then sharp inclines corresponds to the system discovering, exploring, and eventually exploiting the new information the system has found. While erratic, this behaviour is still governed by assured constraints due to the system utilising AMARL-AC and reliably detecting the risks within the domain. With these constraints that adapt whenever an agent within the system detects a new source of risk, state, or goal, the system is gradually allowed to explore more of the domain space, depending on the safe abstract policy that the system is being utilised.

The exploration into unknown states is naturally more likely to be allowed to occur if the goal within these states is easily visible, as this is an incentive for the quantitative analysis to synthesise a policy that allows exploration into the said state. This can be seen clearly by the evolved behaviours displayed in the right section of Figure 6.3. The system first explores $Room_f$ as the goal with this Room is easily visible from $Room_E$. The next explored rooms were $Room_G$ and $Room_H$, with the discovery of these rooms taking much longer due to them being concealed from the system by the sharp bend in $hallways_9$. Finally, in $Room_D$, there is a visible goal that the agent passing through $hallways_3$ would have added to the AMDP. However, as the exploration into $Room_D$ would have entailed moving into view of a camera twice for one goal, it was not deemed a worthy pursuit through quantitative verification. While $Room_I$ was not seen to give the incentive to explore this area, the goal within $Room_I$ is difficult for the system to view. These experiments show the behaviours that AMARL-AC can produce in a system and how it facilitates exploration in unknown states. However, it also shows that AMARL-AC can be utilised in MARL systems to allow them to produce policies that meet strict safety and functional requirements.

Despite the promising results, the work presented here for state adaption is currently underdeveloped and only shown to work on a single domain. Further work must be pursued to show how the assumptions of this approach can be frequently met. These assumptions are currently the main limitations of this work.

6.4 Summary

This chapter introduces AMARL-AC, an extension to AMARL that allows inaccuracies within the AMDP to be automatically corrected by the MARL system during run time, allowing the system to constrain its behaviour and continue learning under safety assurances. This extension can enable systems to adapt to inaccuracies from the beginning of the learning process or inaccuracies formed during runtime due to changing environments. Furthermore, the extension allows inaccuracies in transitions to be corrected, whether a transition is present, and the reward and risk associated with the said transition. However, transitions are not the only factor that can be updated within a domain. Through preliminary evaluations, it has been shown that it is possible to update the states within the environment, more specifically, adapting to new states that were not initially incorporated into the AMDP.

The chapter has two sections, focusing on transition-based adaption and state-based adaption, respectively. The first of these sections use a grid-based flag collection domain with a two-agent homogeneous system which must collect flags while avoiding potential damage to the system. This domain is the same as what is utilised in Chapter5 and remains unchanged. In contrast, the second section uses a deep learning domain consisting of a three-agent homogeneous system that also must collect flags while avoiding detection by surveillance cameras. This domain is the same domain that is introduced in Chapter 4 and remains unchanged. It is shown in this chapter that by making use of algorithms and data structures introduced through Chapters 4 to 6 that AMARL-AC allows a MARL system to adapt to transitional changes within a domain as well as state-based changes. This occurs while uninterrupted learning produces behaviours that meet safety requirements during and after learning and functional requirements after learning. This chapter answers several questions raised in Chapter 5, such as if a solution can be devised that allows a domain expert to be removed from the process after learning has begun and if the said solution can be utilised to adapt to inaccuracies without the aid of a domain expert.

While AMARL-AC has answered the questions posed in Chapter 5, further questions have arisen. These questions involve concepts regarding shortening the learning time required to allow adaption to be used efficiently during runtime and whether the assumptions of its use can be reliably met or mitigated. These assumptions include the ability of the MARL system to identify which abstract state it is in, being in possession of a sensor or set of sensors that allow it to recognise new risks and goals, and finally, being able to recognise when it has located a new state.

Chapter 7

Conclusion

This thesis has discussed and addressed a significant limitation with the practical use of MARL systems within safety-critical and mission-critical scenarios. This issue is the lack of confidence in the multi-agent stochastic process regarding safety and, from this, the lack of assurances that safety and functional requirements will be met. Due to this lack of assurance, MARL is often avoided in safety-critical and mission-critical scenarios, where the agents within the MARL system should not sustain damage, damage other systems, or allow harm to come to humans.

Safe MARL, a relatively new research area closely tied to safe reinforcement learning, has produced several techniques and methods to mitigate this limitation. However, most of these techniques do not offer any safety assurances; they only provide a way of producing 'safer' MARL behaviour. Furthermore, despite some techniques offering a type of safety assurance to MARL, this safety assurance can easily compromise functionality and therefore lacks functional assurances. An overview of the techniques that have been developed was given in Chapter 3.

In order to produce a solution to this issue within the research area of safe MARL, this thesis introduced AMARL, a multi-stage plug-in-styled approach for constraining MARL to meet safety requirements. AMARL uses quantitative verification to provide formal assurances that these safety requirements will be met while also providing formal assurances that the learning process could meet functional requirements if left to learn for long enough. Additionally, AMARL-PPR was introduced, which allows the reduction of the learning time when AMARL has been applied to a domain space with inaccurate safe abstract policy constraints due to inaccuracies during AMDP construction or due to the learning domain changing over time. This allows previously learned information about the domain to be partially reused within the new learning process. Finally, AMARL-AC was introduced, which allows the MARL system to update the AMDP of the domain when the MARL system discovers changes. This AMDP is updated using an algorithm introduced in Chapter 4 and verified using requirements pre-specified in probabilistic computational tree logic. New constraints are then given to the individual agents within the MARL system using an automated process during run time. This run-time adaption differs from AMARL-PPR, which can only be used once the learning process has been finished or halted.

AMARL is designed to be generalisable for use across all types of domains and with different systems. However, AMARL-PPC has the limitation of being designed for MARL techniques that utilise Q-Tables, and AMARL-AC requires risks and goals to be perceivable.

The rest of this chapter details the contributions of the thesis, the limitations of those contributions, and possible directions for future work.

7.1 Contributions

This thesis offers five contributions to the field of safe MARL research, with the main contribution being the multi-stage plugin-styled AMARL approach described in Chapter 4. The AMARL approach differs from existing methods in safe MARL research by using quantitative verification, which supplies formal assurances of safety during and after learning. Quantitative analysis, the distinguishing feature of AMARL, is facilitated through the construction of an AMDP, which mitigates a significant limitation of quantitative analysis, i.e., its inefficiency when faced with large models. The use of an AMDP also allows AMARL to be utilised when preliminary knowledge is not complete, meaning only states and transitions related to functional and safety requirements need to be known. The AMDP is then analysed using quantitative verification to synthesise Pareto-optimal safe abstract policies. This quantitative verification is guided by the safety and functional requirements described in PCTL, meaning that the synthesised abstract policies are guaranteed to meet these requirements. Finally, given the set of safe abstract policies, a domain expert selects one that gives the most appropriate compromise between safety and functionality regarding the specific domain needs.

The selected safe abstract policy is then used to constrain the low-level states and transitions by removing these transitions from the MDPs of the agents within the MARL system. This reduction of the MDPs means that all low-level states and transitions not incorporated within the abstracted states present in the abstract policy are removed. As well as this, it can also mean that specific transitions can be limited in regard to agent use, allowing movement into unsafe areas in a highly controlled fashion. These constraints are set in place before learning and remain throughout and after the learning process. These constraints guarantee that safety requirements will be met throughout the learning process through the assurances of the synthesised safe abstract policy. Additionally, safety and functional requirements will be met consistently within the final learned policy.

Compared to other methods and techniques within safe MARL research, AMARL has one main significant advantage: the use of quantitative verification that gives formal assurances that both safety and functional requirements will be met. Other techniques and methods rely on safety being built into reward structures, states and transitions being constrained without assurances or other methods which do not have the added benefit of formal assurances to build confidence. Another advantage of AMARL is the Pareto-optimal guarantees, meaning that despite the compromises needed in safety or functionality, the policies received will provide optimal trade-offs between functionality and safety. Lastly, AMARL is a flexible approach, being conceptualised as a plugin-styled approach, allowing many tools, techniques, systems, and domain types to be used as needed for specific applications and scenarios.

The second contribution of this thesis, presented in Chapter 5, is the AMARL extension named AMARL-PPR. AMARL-PPR is an extension that is applicable when the learning process is halted or finished. This extension is applicable when it is identified that the safe abstract policies do not result in the expected safety and functionality assurances that the policy offers. This deviation from expectation will be caused by inconsistencies in the constructed AMDP, resulting in an analysis of said AMDP that fails to reflect the realities of the actual problem domain. In standard AMARL, the learning and the time it took would be wasted, with all learned information discarded and the process starting from the beginning. With AMARL-PPR, the learned information from the MARL system's first learning process can be partially reused to increase the speed of the follow-up learning process. AMARL-PPR consists of updating and correcting the AMDP, rerunning the quantitative analysis, and selecting a new, accurate, safe abstract policy. These abstract policies are then autonomously compared to the abstract policies from the previous learning process. Any abstract states and transitions shared within these policies result in the low-level state-action Q-values related to the abstracted states and transitions to be copied over and reused in the new learning run. This contribution has practicality when working with partial knowledge due to the possibility of new knowledge changing the preconceptions of the scenario or the scenario changing over time.

The third contribution of this thesis, introduced in Chapter 6, is another extension to AMARL, which, unlike AMARL-PPR, is intended to be utilised during run time and is named AMARL-AC. AMARL-AC equips the MARL system with the ability to trigger an automated process which updates outdated AMDPs and launches quantitative verification on this new AMDP. While this analysis is running, the system sets the AMARL system to backup constraints based on what is known to be correct with the previous AMDP, allowing partial learning to continue. When the analysis has been completed, a new safe abstract policy is selected and given to the MARL system, which then constrains itself based on the received abstract policy. This process is triggered when an agent within the MARL system explores areas of the domain and discovers that the transitions, transition probabilities, or even states do not align with that of the agent's matrix worldview. The agents' matrix worldview, introduced in Chapter 4, is a multi-dimensional array which describes the AMDP in a simplified fashion and which the agents can access and manipulate. The benefit of AMARL-AC is the removal of human intervention after some initial properties are declared, allowing the system to be more resilient in terms of changing environments and incomplete knowledge. However, in order for AMARL-AC to be used, the rewards and risks must be perceivable to allow the agents to alter the AMDP. Also, currently, AMARL-AC is only shown to function with navigation-based domains. Despite these limitations, AMARL-AC is the only approach that utilises constrained MDPS with assurance, which can offer automated adaption to incorrect prior knowledge.

The fourth contribution is a new algorithm that can automate the generation of navigation-based AMDPs within the PRISM high-level modelling language. This algorithm, as discussed in Chapter 4, takes as input a multi-dimensional array structure known as the matrix worldview, which contains information on abstracted states and transitions. This information records whether a transition between states is viable, the risk associated with taking said transition, and the reward associated with taking said transition. Humans or agents can easily update this structure to change the structure of an AMDP by removing or adding possible transitions between states, changing risk and reward values, and even adding states. This matrix worldview is taken as input to the algorithm, as well as the initial starting state of the MARL system and the number of agents within it. This information is then used to construct an AMDP, which can be used in quantitative analysis without any further input from a domain expert. This contribution dramatically improves the accessibility of AMARL, removing the need for advanced knowledge of the PRISM language to use it, and greatly decreases the setup time for AMARL. It also facilitates the automated functionalities of AMARL-PPR and AMARL-AC, which use this algorithm. AMARL-AC uses the matrix worldview within the agents' code as a fundamental concept.

The fifth contribution is a set of three case studies and an evaluation of AMARL and its extensions which made use of said case studies. The first of these case studies consist of three domains within a ROS patrolling simulator, introduced in Chapter 4. These three domains have been created or updated to include areas of risk that the agents must avoid while completing their patrolling mission to complete the mission with the most amount of battery conserved. The second case study, also introduced in Chapter 4, is a domain implemented within the Unity games engine and uses the MLAgents plugin to allow easy access to deep MARL algorithms. In addition, it provides a new domain for safe MARL-utilising neural networks. The last case study introduced within this thesis in Chapter 5 is a traditional grid-based world adopted from previous research in safe reinforcement learning but updated for AMARL-PPR and multiple agents. This case study was also created within the Unity games engine and could easily be replicated in many other languages and software. All these domains are navigation-based, allowing MARL systems to evaluate safe MARL approaches and methods.

These case studies were then utilised within the evaluation of AMARL, which spanned three different domains, using: several traditional MARL techniques and one deep MARL technique; homogeneous and heterogeneous systems of agents; and slightly varied system sizes. This broad evaluation was used to demonstrate the versatility of the plugin style of AMARL. Throughout all these setups, AMARL could constrain the systems never to breach safety constraints, and the final learned policy always met functional requirements. AMARL-PPR was also evaluated within the grid-world domain to show the potential benefits to the speed of the learning process when rerunning the learning process is required, and the results showed that, within a setting where the previous safe abstract policy and the new safe abstract policy share commonalities, the learning process can be sped up. AMARL-AC was also evaluated within the grid-world domain to demonstrate that the automated process can constrain itself with an updated, safe abstract policy. Within this domain, the MARL system added and removed transitions and rewards, allowing safe MARL to continue efficiently. Lastly, AMARL-AC was shown to be able to add entire abstracted states and transition to an AMDP and continue to constrain itself without breaching safety requirements. This important benefit was accomplished using deep MARL.

7.2 Limitations

The main limitation of AMARL and its extensions, common in safe reinforcement learning and safe MARL approaches, is the need for preliminary knowledge. Without preliminary knowledge that gives enough insight into safety and functionality, a full enough AMDP will not be able to be constructed, any abstract policies that are produced will not be valid, and assurances will not hold.

The quantitative analysis that produces these abstract policies also has limitations. The first limitation (which requires further exploration) is the explosion of model sizes and how this could affect the time required for the analysis to conclude. This explosion of model sizes is impactful when multiple agents are concerned. The model must contain the possible transitions of every agent within the system, causing the model size to increase exponentially. It is reasonable to assume that the analysis of a model of too great complexity would take an unreasonable amount of time to conclude. The second limitation of the quantitative analysis process is the potentially limited number of safe abstract policies supplied by analysing the model. After a lengthy analysis process, no abstract policy may be produced, and further refinement of the model may be required, which could be a time-consuming process. However, we note that the lack of abstract policies can also be an advantage of the process if the model is entirely accurate and cannot be refined, meaning that the analysis has concluded that the MARL system should not be deployed.

As alluded to when discussing the explosion of model sizes, a limitation of this approach is the number of agents that can represent within a model and, in turn, the number of agents that can be practically used. The maximum number of agents which were utilised during the course of this work was three agents. While it is theoretically possible to use a higher number of agents with this approach, a limitation of this work is that the higher limit of the number of agents that can be used was not identified. However, dependent on the time and resource requirements of the verification tools, verifying a model with more than three agents would be a demanding process.

Another limitation of using quantitative verification tools is linked to the same time and resource requirements that were mentioned previously. The limitation is the number of abstract states defined within the model. This work has not identified the maximum number of abstract states that can be incorporated into a model and can still be utilised with AMARL. However, the same issues that limit the number of agents apply to the number of abstracted states. Most notably, the time and resource intensiveness of verifying properties and synthesising policies in large models can be demanding. This means the size of a problem domain, even when abstracted, will be a limiting factor to AMARL.

The tools used within quantitative analysis make use of PCTL to define the objectives that will drive the analysis. These are objectives that can be tracked, such as how much battery was used and how much damage was dealt to the system. However, at the time of writing, the tools available struggle to track more than two objectives or mission constraints simultaneously. This lack of functionality means that the safe abstract policies that are synthesised are only guaranteed to meet two objectives. While it is possible to construct a model that represents this safe abstract policy and run the verification of other properties, it has the potential to be a lengthy process. Furthermore, there is no guarantee that any of the safe abstract policies that are synthesised will then be shown to meet other objectives or properties.

The algorithm that automates AMDP generation makes it possible to construct an AMDP without much knowledge of the PRISM language. However, it is still advisable that the user of AMARL understands the language. This suggestion is in case the algorithm cannot be used for any revisions that may be required. This means that for reliable use of the AMARL algorithm, there must be a degree of expertise in utilising the PRISM model checker and the PRISM language to use AMARL to its full extent. The same limitation is also present due to the need to encode the system requirements into PCTL. Even when utilising the automated systems, the PCTL requirements must be given as input to use AMARL and its extensions.

7.3 Future Work

At the current state of understanding of AMARL and its extensions, multiple areas could be focused upon for future work. These primarily include adaptions that can be used to increase the ease of use and efficiency of AMARL and also investigations to determine the limits of AMARL.

Partial Quantitative Analysis could be used to reduce the time required for quantitative analysis when similar domains are being analysed or when AMDPs have been constructed inaccurately. The concept behind this suggested area of future work is to utilise previous knowledge from safe abstract policies which are still partially relevant when considering the new condition of a domain. From this partial safe abstract policy, the AMDP can be reformed to retract all unnecessary states and possible transitions that have already been covered in the parts of the safe abstract policy that are still valid. For example, suppose that the previous safe abstract policy governs one agent within a system in a way that remains entirely valid with the new understanding of the domain by this agent. In that case, the functional objectives the agent is instructed to complete and the risks it encounters can be removed entirely from the AMDP. However, this technique will only be applicable if these objectives and risks are also factored into the PCTL requirements to allow the complete domain to be captured.

By utilising a method similar to a partial quantitative analysis, the time required to analyse models could be reduced by the degree that the previous safe abstract policy is still valid. However, such a method requires the utmost confidence that the portions of the safe abstract policy are still valid. Any inaccuracies in this process will result in potentially unsafe behaviours and functional objectives being neglected.

Plain-English Descriptions of PCTL requirements could increase the accessibility of AMARL significantly. This area of suggested future work would enable users to write the domain requirements using structured English grammar, removing the need to know how to define these requirements using PCTL. This suggested area of future work has foundations that are found in the pattern system of the most common PCTL commands named ProProST [53], which has been continued in further work attempting to increase the number of commands that are available in this pattern system [10]. Such pattern systems have been proposed and utilised in recent research, e.g., to develop a toolchain that allows such techniques to be utilised on mobile robots for use with LTL, CTL, and PCTL temporal logics [91, 92, 135]. If a system such as this could be incorporated into the AMARL approach, users would no longer have to know PCTL.

AMARL Solutions from a Simple User-Interface would remove the need for users to become familiar with the algorithm structure presented within this thesis or how they work together, significantly increasing the ease of use of AMARL. By containing all user interaction within a simple user interface, this would allow users to supply an agent worldview Matrix description of the AMDP, PCTL requirements either in PCTL or structured English grammar, the number of agents within the system, and the initial state of the system without altering any code. After these inputs have been supplied using the said user interface, an AMDP would be automatically generated, the quantitative analysis started, and safe abstract policies presented on-screen with further details of their safety and functional assurances. This process would remove the need for the user to interact with a quantitative analysis tool and be given a list of policies they can select from, as well as instructions on how to use these safe abstract policies. For specific domains, currently exclusive to navigation domains, this user interface would aid users with limited knowledge of the notations and tools that AMARL requires. However, with the current state of the algorithms presented within the thesis, any domain which is not a navigation domain would not be able to be used with this user interface. In order to allow this, further algorithms would have to be created to deal with additional problem domains.

Evaluation of AMARL within Larger System and Domain Sizes would significantly increase the validity of AMARL as a flexible approach. This thesis does not address the limitations of AMARL regarding the size of systems and domains for which it can be feasibly used, primarily working with AMDPs of a smaller size and a system size no greater than three agents. Given the use of AMDPs, the number of lower-level states can be significantly large, but if the AMDP becomes too large, issues can arise when the analysis is running. The leading causes for AMDPs to quickly become significantly large are the number of states within the model and how many agents the model is capturing. Depending on these properties, the time required to analyse a model can vary drastically, possibly becoming infeasible. Large-scale multi-agent domains that can be used to explore the scalability of AMARL are reviewed in [120].

Evaluation of AMARL in Non-Navigation Domains is required to confirm AMARL's flexibility. Within this thesis, the focus has been on multi-robot systems, which are often used within navigation-based domains. As such, the domains used to evaluate AMARL and its extensions have been navigation based. However, AMARL is presented and shows no apparent limitations as a domain-flexible approach. As such, it should be able to be utilised with systems other than multi-robot navigation-based systems. It would benefit AMARL greatly if this hypothesis were evaluated thoroughly within additional domain types, demonstrating and showcasing how it can be utilised, e.g., in the non-navigation-based domains discussed in [56].

Appendix A

PRISM Model for Radiation Domain

```
mdp

const N = 4;

module RobotSystem
r0 : [0..7] init 1;
r1 : [0..7] init 2;

visits0 : [0..N] init 0;
visits1 : [0..N] init 0;
visits2 : [0..N] init 0;
visits3 : [0..N] init 0;
visits4 : [0..N] init 0;
visits5 : [0..N] init 0;
visits6 : [0..N] init 0;
visits7 : [0..N] init 0;

done : bool init false;
complete : [0..1] init 0;

[] visits0 >= 3 & visits1 >= 3 & visits2 >= 3 & visits3 >= 3 & visits4 >= 3 &
    visits5 >= 3 & visits6 >= 3 & visits7 >= 3  & r0 != 3 & r0 != 4 & r1 != 3 &
     r1 != 4-> (done'= true)&(complete'=1);


[visits0_1_0] !done & r0=0 & visits1<N -> 1:(r0'=1)&(visits1'=visits1+1);
[visits0_1_1] !done & r1=0 & visits1<N -> 1:(r1'=1)&(visits1'=visits1+1);
```

```
[visits0_2_0] !done & r0=0 & visits2<N -> 1:(r0'=2)&(visits2'=visits2+1);
[visits0_2_1] !done & r1=0 & visits2<N -> 1:(r1'=2)&(visits2'=visits2+1);
[visits0_3_0] !done & r0=0 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits0_3_1] !done & r1=0 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits0_5_0] !done & r0=0 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits0_5_1] !done & r1=0 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);

[visits1_0_0] !done & r0=1 & visits0<N -> 1:(r0'=0)&(visits0'=visits0+1);
[visits1_0_1] !done & r1=1 & visits0<N -> 1:(r1'=0)&(visits0'=visits0+1);
[visits1_2_0] !done & r0=1 & visits2<N -> 1:(r0'=2)&(visits2'=visits2+1);
[visits1_2_1] !done & r1=1 & visits2<N -> 1:(r1'=2)&(visits2'=visits2+1);
[visits1_3_0] !done & r0=1 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits1_3_1] !done & r1=1 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits1_5_0] !done & r0=1 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits1_5_1] !done & r1=1 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);

[visits2_0_0] !done & r0=2 & visits0<N -> 1:(r0'=0)&(visits0'=visits0+1);
[visits2_0_1] !done & r1=2 & visits0<N -> 1:(r1'=0)&(visits0'=visits0+1);
[visits2_1_0] !done & r0=2 & visits1<N -> 1:(r0'=1)&(visits1'=visits1+1);
[visits2_1_1] !done & r1=2 & visits1<N -> 1:(r1'=1)&(visits1'=visits1+1);
[visits2_3_0] !done & r0=2 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits2_3_1] !done & r1=2 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits2_5_0] !done & r0=2 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits2_5_1] !done & r1=2 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);
[visits2_6_0] !done & r0=2 & visits6<N -> 1:(r0'=6)&(visits6'=visits6+1);
[visits2_6_1] !done & r1=2 & visits6<N -> 1:(r1'=6)&(visits6'=visits6+1);

[visits3_0_0] !done & r0=3 & visits0<N -> 1:(r0'=0)&(visits0'=visits0+1);
[visits3_0_1] !done & r1=3 & visits0<N -> 1:(r1'=0)&(visits0'=visits0+1);
[visits3_1_0] !done & r0=3 & visits1<N -> 1:(r0'=1)&(visits1'=visits1+1);
[visits3_1_1] !done & r1=3 & visits1<N -> 1:(r1'=1)&(visits1'=visits1+1);
[visits3_2_0] !done & r0=3 & visits2<N -> 1:(r0'=2)&(visits2'=visits2+1);
[visits3_2_1] !done & r1=3 & visits2<N -> 1:(r1'=2)&(visits2'=visits2+1);
[visits3_4_0] !done & r0=3 & visits4<N -> 1:(r0'=4)&(visits4'=visits4+1);
[visits3_4_1] !done & r1=3 & visits4<N -> 1:(r1'=4)&(visits4'=visits4+1);
[visits3_5_0] !done & r0=3 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits3_5_1] !done & r1=3 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);
[visits3_6_0] !done & r0=3 & visits6<N -> 1:(r0'=6)&(visits6'=visits6+1);
[visits3_6_1] !done & r1=3 & visits6<N -> 1:(r1'=6)&(visits6'=visits6+1);
```

```

[visits4_3_0] !done & r0=4 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits4_3_1] !done & r1=4 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits4_6_0] !done & r0=4 & visits6<N -> 1:(r0'=6)&(visits6'=visits6+1);
[visits4_6_1] !done & r1=4 & visits6<N -> 1:(r1'=6)&(visits6'=visits6+1);
[visits4_7_0] !done & r0=4 & visits7<N -> 1:(r0'=7)&(visits7'=visits7+1);
[visits4_7_1] !done & r1=4 & visits7<N -> 1:(r1'=7)&(visits7'=visits7+1);

[visits5_0_0] !done & r0=5 & visits0<N -> 1:(r0'=0)&(visits0'=visits0+1);
[visits5_0_1] !done & r1=5 & visits0<N -> 1:(r1'=0)&(visits0'=visits0+1);
[visits5_1_0] !done & r0=5 & visits1<N -> 1:(r0'=1)&(visits1'=visits1+1);
[visits5_1_1] !done & r1=5 & visits1<N -> 1:(r1'=1)&(visits1'=visits1+1);
[visits5_2_0] !done & r0=5 & visits2<N -> 1:(r0'=2)&(visits2'=visits2+1);
[visits5_2_1] !done & r1=5 & visits2<N -> 1:(r1'=2)&(visits2'=visits2+1);
[visits5_3_0] !done & r0=5 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits5_3_1] !done & r1=5 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits5_6_0] !done & r0=5 & visits6<N -> 1:(r0'=6)&(visits6'=visits6+1);
[visits5_6_1] !done & r1=5 & visits6<N -> 1:(r1'=6)&(visits6'=visits6+1);
[visits5_7_0] !done & r0=5 & visits7<N -> 1:(r0'=7)&(visits7'=visits7+1);
[visits5_7_1] !done & r1=5 & visits7<N -> 1:(r1'=7)&(visits7'=visits7+1);

[visits6_2_0] !done & r0=6 & visits2<N -> 1:(r0'=2)&(visits2'=visits2+1);
[visits6_2_1] !done & r1=6 & visits2<N -> 1:(r1'=2)&(visits2'=visits2+1);
[visits6_3_0] !done & r0=6 & visits3<N -> 1:(r0'=3)&(visits3'=visits3+1);
[visits6_3_1] !done & r1=6 & visits3<N -> 1:(r1'=3)&(visits3'=visits3+1);
[visits6_4_0] !done & r0=6 & visits4<N -> 1:(r0'=4)&(visits4'=visits4+1);
[visits6_4_1] !done & r1=6 & visits4<N -> 1:(r1'=4)&(visits4'=visits4+1);
[visits6_5_0] !done & r0=6 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits6_5_1] !done & r1=6 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);
[visits6_7_0] !done & r0=6 & visits7<N -> 1:(r0'=7)&(visits7'=visits7+1);
[visits6_7_1] !done & r1=6 & visits7<N -> 1:(r1'=7)&(visits7'=visits7+1);

[visits7_4_0] !done & r0=7 & visits4<N -> 1:(r0'=4)&(visits4'=visits4+1);
[visits7_4_1] !done & r1=7 & visits4<N -> 1:(r1'=4)&(visits4'=visits4+1);
[visits7_5_0] !done & r0=7 & visits5<N -> 1:(r0'=5)&(visits5'=visits5+1);
[visits7_5_1] !done & r1=7 & visits5<N -> 1:(r1'=5)&(visits5'=visits5+1);
[visits7_6_0] !done & r0=7 & visits6<N -> 1:(r0'=6)&(visits6'=visits6+1);
[visits7_6_1] !done & r1=7 & visits6<N -> 1:(r1'=6)&(visits6'=visits6+1);

```

```
endmodule

rewards "risk"
    [visits0_3_0] true : 0.1;
    [visits0_3_1] true : 0.1;

    [visits1_3_0] true : 0.1;
    [visits1_3_1] true : 0.1;

    [visits2_3_0] true : 0.1;
    [visits2_3_1] true : 0.1;

    [visits3_0_0] true : 0.1;
    [visits3_0_1] true : 0.1;
    [visits3_1_0] true : 0.1;
    [visits3_1_1] true : 0.1;
    [visits3_2_0] true : 0.1;
    [visits3_2_1] true : 0.1;
    [visits3_4_0] true : 0.2;
    [visits3_4_1] true : 0.2;
    [visits3_5_0] true : 0.1;
    [visits3_5_1] true : 0.1;
    [visits3_6_0] true : 0.1;
    [visits3_6_1] true : 0.1;

    [visits4_3_0] true : 0.2;
    [visits4_3_1] true : 0.2;
    [visits4_6_0] true : 0.1;
    [visits4_6_1] true : 0.1;
    [visits4_7_0] true : 0.1;
    [visits4_7_1] true : 0.1;

    [visits5_3_0] true : 0.1;
    [visits5_3_1] true : 0.1;

    [visits6_3_0] true : 0.1;
    [visits6_3_1] true : 0.1;
    [visits6_4_0] true : 0.1;
    [visits6_4_1] true : 0.1;

    [visits7_4_0] true : 0.1;
    [visits7_4_1] true : 0.1;
endrewards



rewards "battery"
    [visits0_1_0] true : 3;
    [visits0_1_1] true : 3;
    [visits0_2_0] true : 3;
    [visits0_2_1] true : 3;
    [visits0_3_0] true : 6;
    [visits0_3_1] true : 6;
    [visits0_5_0] true : 8;
    [visits0_5_1] true : 8;

    [visits1_0_0] true : 2;
    [visits1_0_1] true : 2;
    [visits1_2_0] true : 2;
    [visits1_2_1] true : 2;
    [visits1_3_0] true : 2;
    [visits1_3_1] true : 2;
    [visits1_5_0] true : 5;
    [visits1_5_1] true : 5;

    [visits2_0_0] true : 2;
    [visits2_0_1] true : 2;
    [visits2_1_0] true : 2;
    [visits2_1_1] true : 2;
    [visits2_3_0] true : 2;
    [visits2_3_1] true : 2;
    [visits2_5_0] true : 2;
    [visits2_5_1] true : 2;
    [visits2_6_0] true : 2;
    [visits2_6_1] true : 2;

    [visits3_0_0] true : 2;
    [visits3_0_1] true : 2;
    [visits3_1_0] true : 2;
    [visits3_1_1] true : 2;
    [visits3_2_0] true : 3;
    [visits3_2_1] true : 3;
    [visits3_4_0] true : 2;
    [visits3_4_1] true : 2;
    [visits3_5_0] true : 2;
    [visits3_5_1] true : 2;
    [visits3_6_0] true : 5;
    [visits3_6_1] true : 5;
    [visits4_3_0] true : 1;
    [visits4_3_1] true : 1;
    [visits4_6_0] true : 7;
    [visits4_6_1] true : 7;
    [visits4_7_0] true : 5;
    [visits4_7_1] true : 5;

    [visits5_0_0] true : 3;
    [visits5_0_1] true : 3;
    [visits5_1_0] true : 2;
    [visits5_1_1] true : 2;
    [visits5_2_0] true : 2;
    [visits5_2_1] true : 2;
    [visits5_3_0] true : 1;
    [visits5_3_1] true : 1;
    [visits5_6_0] true : 4;
    [visits5_6_1] true : 4;
    [visits5_7_0] true : 7;
    [visits5_7_1] true : 7;

    [visits6_2_0] true : 7;
    [visits6_2_1] true : 7;
    [visits6_3_0] true : 5;
    [visits6_3_1] true : 5;
    [visits6_4_0] true : 7;
    [visits6_4_1] true : 7;
    [visits6_5_0] true : 4;
    [visits6_5_1] true : 4;
    [visits6_7_0] true : 4;
    [visits6_7_1] true : 4;

    [visits7_4_0] true : 3;
    [visits7_4_1] true : 3;
    [visits7_5_0] true : 7;
    [visits7_5_1] true : 7;
    [visits7_6_0] true : 4;
    [visits7_6_1] true : 4;
endrewards

//Automated PRISM Code Generation
```

Listing A.1: PRISM Model Adaption for Radiation Domain Map A

Appendix B

PRISM Model for Infiltration Domain

```
mdp

//A = 0; B = 1; C = 2; D = 3; E = 4; F = 5; G = 6; H = 7; I = 8.
const int N = 3;
const int goal = 1;

///[Command Name_CurrentRoom_RoomToVisit_RobotName]
module RobotSystem

  r1 : [0..8] init 0;
  r2 : [0..8] init 0;
  r3 : [0..8] init 0;

  visits0 : [0..N] init 0; // visit counter foor room 0
  visits1 : [0..N] init 0; // visit counter foor room 1
  visits2 : [0..N] init 0; // visit counter foor room 3
  ...

  //Has the goal in room N been completed?
  goal0 : bool init false;
  goal1 : bool init false;
  goal2 : bool init false;
  ...

  //Is the mission completed?
  done : bool init false;

```

```
//  In room Zero and making a movmement choice
//without goal
[visit0_1_1]  !done & r1=0 & goal1 & visits1<N -> 1:(r1'=1)&(visits1'=visits1
  +1); // robot 1 visits room 1
[visit0_3_1]  !done & r1=0 & goal3 & visits3<N -> 1:(r1'=3)&(visits3'=visits3
  +1); // robot 1 visits room 3
[visit0_4_1]  !done & r1=0 & goal4 & visits4<N -> 1:(r1'=4)&(visits4'=visits4
  +1); // robot 1 visits room 4

//with goal
[visit0_1_1_G]  !done & r1=0 & !goal1 & visits1<N -> 1:(r1'=1)&(visits1'=
  visits1+1)&(goal1'=true); // robot 1 visits room 1
[visit0_3_1_G]  !done & r1=0 & !goal3 & visits3<N -> 1:(r1'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 1 visits room 3
[visit0_4_1_G]  !done & r1=0 & !goal4 & visits4<N -> 1:(r1'=4)&(visits4'=
  visits4+1)&(goal4'=true); // robot 1 visits room 4

//without goal
[visit0_1_2]  !done & r2=0 & goal1 & visits1<N -> 1:(r2'=1)&(visits1'=visits1
  +1); // robot 2 visits room 1
[visit0_3_2]  !done & r2=0 & goal3 & visits3<N -> 1:(r2'=3)&(visits3'=visits3
  +1); // robot 2 visits room 3
[visit0_4_2]  !done & r2=0 & goal4 & visits4<N -> 1:(r2'=4)&(visits4'=visits4
  +1); // robot 2 visits room 4

//with goal
[visit0_1_2_G]  !done & r2=0 & !goal1 & visits1<N -> 1:(r2'=1)&(visits1'=
  visits1+1)&(goal1'=true); // robot 2 visits room 1
[visit0_3_2_G]  !done & r2=0 & !goal3 & visits3<N -> 1:(r2'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 2 visits room 3
[visit0_4_2_G]  !done & r2=0 & !goal4 & visits4<N -> 1:(r2'=4)&(visits4'=
  visits4+1)&(goal4'=true); // robot 2 visits room 4
...

//  In room One and making a movmement choice
//without goal
[visit1_0_1]  !done & r1=1 & goal0 & visits0<N -> 1:(r1'=0)&(visits0'=visits0
  +1); // robot 1 visits room 1
```

```
[visit1_2_1]  !done & r1=1 & goal2 & visits2<N -> 1:(r1'=2)&(visits2'=visits2
  +1); // robot 1 visits room 3
[visit1_3_1]  !done & r1=1 & goal3 & visits3<N -> 1:(r1'=3)&(visits3'=visits3
  +1); // robot 1 visits room 4
[visit1_5_1]  !done & r1=1 & goal5 & visits5<N -> 1:(r1'=5)&(visits5'=visits5
  +1); // robot 1 visits room 4

//with goal
[visit1_0_1_G]  !done & r1=1 & !goal0 & visits0<N -> 1:(r1'=0)&(visits0'=
  visits0+1)&(goal0'=true); // robot 1 visits room 1
[visit1_2_1_G]  !done & r1=1 & !goal2 & visits2<N -> 1:(r1'=2)&(visits2'=
  visits2+1)&(goal2'=true); // robot 1 visits room 3
[visit1_3_1_G]  !done & r1=1 & !goal3 & visits3<N -> 1:(r1'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 1 visits room 4
[visit1_5_1_G]  !done & r1=1 & !goal5 & visits5<N -> 1:(r1'=5)&(visits5'=
  visits5+1)&(goal5'=true); // robot 1 visits room 5

//without goal
[visit1_0_2]  !done & r2=1 & goal0 & visits0<N -> 1:(r2'=0)&(visits0'=visits0
  +1); // robot 2 visits room 1
[visit1_2_2]  !done & r2=1 & goal2 & visits2<N -> 1:(r2'=2)&(visits2'=visits2
  +1); // robot 2 visits room 3
[visit1_3_2]  !done & r2=1 & goal3 & visits3<N -> 1:(r2'=3)&(visits3'=visits3
  +1); // robot 2 visits room 4
[visit1_5_2]  !done & r2=1 & goal5 & visits5<N -> 1:(r2'=5)&(visits5'=visits5
  +1); // robot 2 visits room 4

//with goal
[visit1_0_2_G]  !done & r2=1 & !goal0 & visits0<N -> 1:(r2'=0)&(visits0'=
  visits0+1)&(goal0'=true); // robot 2 visits room 1
[visit1_2_2_G]  !done & r2=1 & !goal2 & visits2<N -> 1:(r2'=2)&(visits2'=
  visits2+1)&(goal2'=true); // robot 2 visits room 3
[visit1_3_2_G]  !done & r2=1 & !goal3 & visits3<N -> 1:(r2'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 2 visits room 4
[visit1_5_2_G]  !done & r2=1 & !goal5 & visits5<N -> 1:(r2'=5)&(visits5'=
  visits5+1)&(goal5'=true); // robot 2 visits room 5
...

//  In room Two and making a movmement choice
```

```
//without goal
[visit2_1_1]  !done & r1=2 & goal1 & visits1<N -> 1:(r1'=1)&(visits1'=visits1
  +1); // robot 1 visits room 1
[visit2_8_1_N]  !done & r1=2 & goal8 & visits8<N -> 1:(r1'=8)&(visits8'=
  visits8+1); // robot 1 visits room 8
[visit2_8_1_S]  !done & r1=2 & goal8 & visits8<N -> 1:(r1'=8)&(visits8'=
  visits8+1); // robot 1 visits room 8

//with goal
[visit2_1_1_G]  !done & r1=2 & !goal1 & visits1<N -> 1:(r1'=1)&(visits1'=
  visits1+1)&(goal1'=true); // robot 1 visits room 1
[visit2_8_1_N_G]  !done & r1=2 & !goal8 & visits8<N -> 1:(r1'=8)&(visits8'=
  visits8+1)&(goal8'=true); // robot 1 visits room 8
[visit2_8_1_S_G]  !done & r1=2 & !goal8 & visits8<N -> 1:(r1'=8)&(visits8'=
  visits8+1)&(goal8'=true); // robot 1 visits room 8
...

//  In room Three and making a movmement choice
//without goal
[visit3_0_1]  !done & r1=3 & goal0 & visits0<N -> 1:(r1'=0)&(visits0'=visits0
  +1); // robot 1 visits room 0
[visit3_1_1]  !done & r1=3 & goal1 & visits1<N -> 1:(r1'=1)&(visits1'=visits1
  +1); // robot 1 visits room 1
[visit3_4_1]  !done & r1=3 & goal4 & visits4<N -> 1:(r1'=4)&(visits4'=visits4
  +1); // robot 1 visits room 4
[visit3_5_1]  !done & r1=3 & goal5 & visits5<N -> 1:(r1'=5)&(visits5'=visits5
  +1); // robot 1 visits room 5

//with goal
[visit3_0_1_G]  !done & r1=3 & !goal0 & visits0<N -> 1:(r1'=0)&(visits0'=
  visits0+1)&(goal0'=true); // robot 1 visits room 1
[visit3_1_1_G]  !done & r1=3 & !goal1 & visits1<N -> 1:(r1'=1)&(visits1'=
  visits1+1)&(goal1'=true); // robot 1 visits room 3
[visit3_4_1_G]  !done & r1=3 & !goal4 & visits4<N -> 1:(r1'=4)&(visits4'=
  visits4+1)&(goal4'=true); // robot 1 visits room 4
[visit3_5_1_G]  !done & r1=3 & !goal5 & visits5<N -> 1:(r1'=5)&(visits5'=
  visits5+1)&(goal5'=true); // robot 1 visits room 5
...

```

```
//  In room Four and making a movmement choice
//without goal
[visit4_0_1]  !done & r1=4 & goal0 & visits0<N -> 1:(r1'=0)&(visits0'=visits0
  +1); // robot 1 visits room 0
[visit4_3_1]  !done & r1=4 & goal3 & visits3<N -> 1:(r1'=3)&(visits3'=visits3
  +1); // robot 1 visits room 3
[visit4_5_1]  !done & r1=4 & goal5 & visits5<N -> 1:(r1'=5)&(visits5'=visits5
  +1); // robot 1 visits room 5
[visit4_6_1]  !done & r1=4 & goal6 & visits6<N -> 1:(r1'=6)&(visits6'=visits6
  +1); // robot 1 visits room 6

//with goal
[visit4_0_1_G]  !done & r1=4 & !goal0 & visits0<N -> 1:(r1'=0)&(visits0'=
  visits0+1)&(goal0'=true); // robot 1 visits room 0
[visit4_3_1_G]  !done & r1=4 & !goal3 & visits3<N -> 1:(r1'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 1 visits room 3
[visit4_5_1_G]  !done & r1=4 & !goal5 & visits5<N -> 1:(r1'=5)&(visits5'=
  visits5+1)&(goal5'=true); // robot 1 visits room 5
[visit4_6_1_G]  !done & r1=4 & !goal6 & visits6<N -> 1:(r1'=6)&(visits6'=
  visits6+1)&(goal6'=true); // robot 1 visits room 6
...

//  In room Five and making a movmement choice
//without goal
[visit5_1_1]  !done & r1=5 & goal1 & visits1<N -> 1:(r1'=1)&(visits1'=visits1
  +1); // robot 1 visits room 1
[visit5_3_1]  !done & r1=5 & goal3 & visits3<N -> 1:(r1'=3)&(visits3'=visits3
  +1); // robot 1 visits room 3
[visit5_4_1]  !done & r1=5 & goal4 & visits4<N -> 1:(r1'=4)&(visits4'=visits4
  +1); // robot 1 visits room 4
[visit5_7_1]  !done & r1=5 & goal7 & visits7<N -> 1:(r1'=7)&(visits7'=visits7
  +1); // robot 1 visits room 7
[visit5_8_1]  !done & r1=5 & goal8 & visits8<N -> 1:(r1'=8)&(visits8'=visits8
  +1); // robot 1 visits room 8

//with goal
[visit5_1_1_G]  !done & r1=5 & !goal1 & visits1<N -> 1:(r1'=1)&(visits1'=
  visits1+1)&(goal1'=true); // robot 1 visits room 0
```

```
[visit5_3_1_G]  !done & r1=5 & !goal3 & visits3<N -> 1:(r1'=3)&(visits3'=
  visits3+1)&(goal3'=true); // robot 1 visits room 3
[visit5_4_1_G]  !done & r1=5 & !goal4 & visits4<N -> 1:(r1'=4)&(visits4'=
  visits4+1)&(goal4'=true); // robot 1 visits room 4
[visit5_7_1_G]  !done & r1=5 & !goal7 & visits7<N -> 1:(r1'=7)&(visits7'=
  visits7+1)&(goal7'=true); // robot 1 visits room 7
[visit5_8_1_G]  !done & r1=5 & !goal8 & visits8<N -> 1:(r1'=8)&(visits8'=
  visits8+1)&(goal8'=true); // robot 1 visits room 8
...

//  In room Six and making a movmement choice
//without goal
[visit6_4_1]  !done & r1=6 & goal4 & visits4<N -> 1:(r1'=4)&(visits4'=visits4
  +1); // robot 1 visits room 4
[visit6_7_1]  !done & r1=6 & goal7 & visits7<N -> 1:(r1'=7)&(visits7'=visits7
  +1); // robot 1 visits room 7

//with goal
[visit6_4_1_G]  !done & r1=6 & !goal4 & visits4<N -> 1:(r1'=4)&(visits4'=
  visits4+1)&(goal4'=true); // robot 1 visits room 4
[visit6_7_1_G]  !done & r1=6 & !goal7 & visits7<N -> 1:(r1'=7)&(visits7'=
  visits7+1)&(goal7'=true); // robot 1 visits room 7
...

//  In room Seven and making a movmement choice
//without goal
[visit7_5_1]  !done & r1=7 & goal5 & visits5<N -> 1:(r1'=5)&(visits5'=visits5
  +1); // robot 1 visits room 5
[visit7_6_1]  !done & r1=7 & goal6 & visits6<N -> 1:(r1'=6)&(visits6'=visits6
  +1); // robot 1 visits room 6

//with goal
[visit7_5_1_G]  !done & r1=7 & !goal5 & visits5<N -> 1:(r1'=5)&(visits5'=
  visits5+1)&(goal5'=true); // robot 1 visits room 5
[visit7_6_1_G]  !done & r1=7 & !goal6 & visits6<N -> 1:(r1'=6)&(visits6'=
  visits6+1)&(goal6'=true); // robot 1 visits room 6
...

//  In room Eight and making a movmement choice
```

```
  //without goal
  [visit8_2_1_N]  !done & r1=8 & goal2 & visits2<N -> 1:(r1'=2)&(visits2'=
    visits2+1); // robot 1 visits room 2
  [visit8_2_1_S]  !done & r1=8 & goal2 & visits2<N -> 1:(r1'=2)&(visits2'=
    visits2+1); // robot 1 visits room 2
  [visit8_5_1]  !done & r1=8 & goal5 & visits5<N -> 1:(r1'=5)&(visits5'=visits5
    +1); // robot 1 visits room 5

  //with goal
  [visit8_2_1_N_G]  !done & r1=8 & !goal2 & visits2<N -> 1:(r1'=2)&(visits2'=
    visits2+1)&(goal2'=true); // robot 1 visits room 2
  [visit8_2_1_S_G]  !done & r1=8 & !goal2 & visits2<N -> 1:(r1'=2)&(visits2'=
    visits2+1)&(goal2'=true); // robot 1 visits room 2
  [visit8_5_1_G]  !done & r1=8 & !goal5 & visits5<N -> 1:(r1'=5)&(visits5'=
    visits5+1)&(goal5'=true); // robot 1 visits room 5
  ...

   // Done with all the visits
  [done] goal0 & goal1 & goal2 & goal3 & goal4 & goal5 & goal6 & goal7 & goal8
    -> (done'= true);
endmodule


rewards "risk"
  //Room A Risky Actions
  [visit0_3_1] true : 0.05;
  [visit0_4_1] true : 0.05;
  [visit0_3_1_G] true : 0.05;
  [visit0_4_1_G] true : 0.05;

  //Room B Risky Actions
  [visit1_2_1] true : 0.1;
  [visit1_3_1] true : 0.1;
  [visit1_5_1] true : 0.1;
  [visit1_2_1_G] true : 0.1;
  [visit1_3_1_G] true : 0.1;
  [visit1_5_1_G] true : 0.1;

  //Room C Risky Actions
```

```
[visit2_1_1] true : 0.1;
[visit2_8_1_N] true : 0.05;
[visit2_8_1_S] true : 0.1;
[visit2_1_1_G] true : 0.1;
[visit2_8_1_N_G] true : 0.05;
[visit2_8_1_S_G] true : 0.1;
...

//Room A Risky Actions
[visit0_3_2] true : 0.05;
[visit0_4_2] true : 0.05;
[visit0_3_2_G] true : 0.05;
[visit0_4_2_G] true : 0.05;

//Room B Risky Actions
[visit1_2_2] true : 0.1;
[visit1_3_2] true : 0.1;
[visit1_5_2] true : 0.1;
[visit1_2_2_G] true : 0.1;
[visit1_3_2_G] true : 0.1;
[visit1_5_2_G] true : 0.1;

//Room C Risky Actions
[visit2_1_2] true : 0.1;
[visit2_8_2_N] true : 0.05;
[visit2_8_2_S] true : 0.1;
[visit2_1_2_G] true : 0.1;
[visit2_8_2_N_G] true : 0.05;
[visit2_8_2_S_G] true : 0.1;
...

//Room A Risky Actions
[visit0_3_3] true : 0.05;
[visit0_4_3] true : 0.05;
[visit0_3_3_G] true : 0.05;
[visit0_4_3_G] true : 0.05;

//Room B Risky Actions
[visit1_2_3] true : 0.1;
```

```
  [visit1_3_3] true : 0.1;
  [visit1_5_3] true : 0.1;
  [visit1_2_3_G] true : 0.1;
  [visit1_3_3_G] true : 0.1;
  [visit1_5_3_G] true : 0.1;

  //Room C Risky Actions
  [visit2_1_3] true : 0.1;
  [visit2_8_3_N] true : 0.05;
  [visit2_8_3_S] true : 0.1;
  [visit2_1_3_G] true : 0.1;
  [visit2_8_3_N_G] true : 0.05;
  [visit2_8_3_S_G] true : 0.1;
  ...
endrewards

rewards "goal"
  [visit0_1_1_G] true : 1;
  [visit0_3_1_G] true : 1;
  [visit0_4_1_G] true : 1;

  [visit1_0_1_G] true : 1;
  [visit1_2_1_G] true : 1;
  [visit1_3_1_G] true : 1;
  [visit1_5_1_G] true : 1;

  [visit2_1_1_G]  true : 1;
  [visit2_8_1_N_G] true : 1;
  [visit2_8_1_S_G] true : 1;
  ...

  [visit0_1_2_G] true : 1;
  [visit0_3_2_G] true : 1;
  [visit0_4_2_G] true : 1;

  [visit1_0_2_G] true : 1;
  [visit1_2_2_G] true : 1;
  [visit1_3_2_G] true : 1;
  [visit1_5_2_G] true : 1;

  [visit2_1_2_G]  true : 1;
  [visit2_8_2_N_G] true : 1;
  [visit2_8_2_S_G] true : 1;
  ...

  [visit0_1_3_G] true : 1;
  [visit0_3_3_G] true : 1;
  [visit0_4_3_G] true : 1;

  [visit1_0_3_G] true : 1;
  [visit1_2_3_G] true : 1;
  [visit1_3_3_G] true : 1;
  [visit1_5_3_G] true : 1;

  [visit2_1_3_G]  true : 1;
  [visit2_8_3_N_G] true : 1;
  [visit2_8_3_S_G] true : 1;
  ...
endrewards
```

Listing B.1: PRISM Model for Deep AMARL Infiltration Domain. Automated Generation.

Appendix C

PRISM Model for Search and Rescue Domain

```
mdp


//MDP test
module RobotSystem
r0 : [0..6] init 0;
r1 : [0..6] init 0;

visits0 : [0..4] init 0;
visits1 : [0..4] init 0;
visits2 : [0..4] init 0;
visits3 : [0..4] init 0;
visits4 : [0..4] init 0;
visits5 : [0..4] init 0;
visits6 : [0..4] init 0;


 goal0 : bool init false;
 goal1 : bool init false;
 goal2 : bool init false;
 goal3 : bool init false;
 goal4 : bool init false;
 goal5 : bool init false;
 goal6 : bool init false;
done : bool init false;

```

```
[visits0_1_0] !done & !goal1 & r0=0 & visits1<4 -> 1:(r0'=1)&(visits1'=visits1+1)&(goal1'=true);
[visits0_1_0G] !done & goal1= true & r0=0 & visits1<4 -> 1:(r0'=1)&(visits1'=visits1+1);
[visits0_1_1] !done & !goal1 & r1=0 & visits1<4 -> 1:(r1'=1)&(visits1'=visits1+1)&(goal1'=true);
[visits0_1_1G] !done & goal1= true & r1=0 & visits1<4 -> 1:(r1'=1)&(visits1'=visits1+1);

[visits0_2_0] !done & !goal2 & r0=0 & visits2<4 -> 1:(r0'=2)&(visits2'=visits2+1)&(goal2'=true);
[visits0_2_0G] !done & goal2= true & r0=0 & visits2<4 -> 1:(r0'=2)&(visits2'=visits2+1);
[visits0_2_1] !done & !goal2 & r1=0 & visits2<4 -> 1:(r1'=2)&(visits2'=visits2+1)&(goal2'=true);
[visits0_2_1G] !done & goal2= true & r1=0 & visits2<4 -> 1:(r1'=2)&(visits2'=visits2+1);

[visits0_4_0] !done & !goal4 & r0=0 & visits4<4 -> 1:(r0'=4)&(visits4'=visits4+1)&(goal4'=true);
[visits0_4_0G] !done & goal4= true & r0=0 & visits4<4 -> 1:(r0'=4)&(visits4'=visits4+1);
[visits0_4_1] !done & !goal4 & r1=0 & visits4<4 -> 1:(r1'=4)&(visits4'=visits4+1)&(goal4'=true);
[visits0_4_1G] !done & goal4= true & r1=0 & visits4<4 -> 1:(r1'=4)&(visits4'=visits4+1);

[visits1_0_0] !done & !goal0 & r0=1 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1)&(goal0'=true);
[visits1_0_0G] !done & goal0= true & r0=1 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1);
[visits1_0_1] !done & !goal0 & r1=1 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1)&(goal0'=true);
[visits1_0_1G] !done & goal0= true & r1=1 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1);

[visits2_0_0] !done & !goal0 & r0=2 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1)&(goal0'=true);
```

```
[visits2_0_0G] !done & goal0= true & r0=2 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1);
[visits2_0_1] !done & !goal0 & r1=2 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1)&(goal0'=true);
[visits2_0_1G] !done & goal0= true & r1=2 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1);

[visits3_4_0] !done & !goal4 & r0=3 & visits4<4 -> 1:(r0'=4)&(visits4'=visits4+1)&(goal4'=true);
[visits3_4_0G] !done & goal4= true & r0=3 & visits4<4 -> 1:(r0'=4)&(visits4'=visits4+1);
[visits3_4_1] !done & !goal4 & r1=3 & visits4<4 -> 1:(r1'=4)&(visits4'=visits4+1)&(goal4'=true);
[visits3_4_1G] !done & goal4= true & r1=3 & visits4<4 -> 1:(r1'=4)&(visits4'=visits4+1);

[visits4_0_0] !done & !goal0 & r0=4 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1)&(goal0'=true);
[visits4_0_0G] !done & goal0= true & r0=4 & visits0<4 -> 1:(r0'=0)&(visits0'=visits0+1);
[visits4_0_1] !done & !goal0 & r1=4 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1)&(goal0'=true);
[visits4_0_1G] !done & goal0= true & r1=4 & visits0<4 -> 1:(r1'=0)&(visits0'=visits0+1);

[visits4_3_0] !done & !goal3 & r0=4 & visits3<4 -> 1:(r0'=3)&(visits3'=visits3+1)&(goal3'=true);
[visits4_3_0G] !done & goal3= true & r0=4 & visits3<4 -> 1:(r0'=3)&(visits3'=visits3+1);
[visits4_3_1] !done & !goal3 & r1=4 & visits3<4 -> 1:(r1'=3)&(visits3'=visits3+1)&(goal3'=true);
[visits4_3_1G] !done & goal3= true & r1=4 & visits3<4 -> 1:(r1'=3)&(visits3'=visits3+1);

[visits4_5_0] !done & !goal5 & r0=4 & visits5<4 -> 1:(r0'=5)&(visits5'=visits5+1)&(goal5'=true);
[visits4_5_0G] !done & goal5= true & r0=4 & visits5<4 -> 1:(r0'=5)&(visits5'=visits5+1);

```

```
[visits4_5_1] !done & !goal5 & r1=4 & visits5<4 -> 1:(r1'=5)&(visits5'=visits5
    +1)&(goal5'=true);
[visits4_5_1G] !done & goal5= true & r1=4 & visits5<4 -> 1:(r1'=5)&(visits5'=
    visits5+1);

[visits5_4_0] !done & !goal4 & r0=5 & visits4<4 -> 1:(r0'=4)&(visits4'=visits4
    +1)&(goal4'=true);
[visits5_4_0G] !done & goal4= true & r0=5 & visits4<4 -> 1:(r0'=4)&(visits4'=
    visits4+1);

[visits5_4_1] !done & !goal4 & r1=5 & visits4<4 -> 1:(r1'=4)&(visits4'=visits4
    +1)&(goal4'=true);
[visits5_4_1G] !done & goal4= true & r1=5 & visits4<4 -> 1:(r1'=4)&(visits4'=
    visits4+1);

[visits5_6_0] !done & !goal6 & r0=5 & visits6<4 -> 1:(r0'=6)&(visits6'=visits6
    +1)&(goal6'=true);
[visits5_6_0G] !done & goal6= true & r0=5 & visits6<4 -> 1:(r0'=6)&(visits6'=
    visits6+1);

[visits5_6_1] !done & !goal6 & r1=5 & visits6<4 -> 1:(r1'=6)&(visits6'=visits6
    +1)&(goal6'=true);
[visits5_6_1G] !done & goal6= true & r1=5 & visits6<4 -> 1:(r1'=6)&(visits6'=
    visits6+1);

[visits6_5_0] !done & !goal5 & r0=6 & visits5<4 -> 1:(r0'=5)&(visits5'=visits5
    +1)&(goal5'=true);
[visits6_5_0G] !done & goal5= true & r0=6 & visits5<4 -> 1:(r0'=5)&(visits5'=
    visits5+1);

[visits6_5_1] !done & !goal5 & r1=6 & visits5<4 -> 1:(r1'=5)&(visits5'=visits5
    +1)&(goal5'=true);
[visits6_5_1G] !done & goal5= true & r1=6 & visits5<4 -> 1:(r1'=5)&(visits5'=
    visits5+1);


[done] visits0 >= 4 & visits1 >= 4 & visits2 >= 4 & visits3 >= 4 & visits4 >= 4
     & visits5 >= 4 & visits6 >= 4-> (done'= true);
endmodule
```

```

rewards "risk"
[visits0_1_0] true : 0.1;
[visits0_1_0G] true : 0.1;
[visits0_1_1] true : 0.1;
[visits0_1_1G] true : 0.1;
[visits1_0_0] true : 0.1;
[visits1_0_0G] true : 0.1;
[visits1_0_1] true : 0.1;
[visits1_0_1G] true : 0.1;
[visits3_4_0] true : 0.1;
[visits3_4_0G] true : 0.1;
[visits3_4_1] true : 0.1;
[visits3_4_1G] true : 0.1;
[visits4_3_0] true : 0.1;
[visits4_3_0G] true : 0.1;
[visits4_3_1] true : 0.1;
[visits4_3_1G] true : 0.1;
[visits4_5_0] true : 0.1;
[visits4_5_0G] true : 0.1;
[visits4_5_1] true : 0.1;
[visits4_5_1G] true : 0.1;
[visits5_4_0] true : 0.1;
[visits5_4_0G] true : 0.1;
[visits5_4_1] true : 0.1;
[visits5_4_1G] true : 0.1;
[visits5_6_0] true : 0.1;
[visits5_6_0G] true : 0.1;
[visits5_6_1] true : 0.1;
[visits5_6_1G] true : 0.1;
[visits6_5_0] true : 0.1;
[visits6_5_0G] true : 0.1;
[visits6_5_1] true : 0.1;
[visits6_5_1G] true : 0.1;
endrewards

rewards "goal"
[visits0_1_0] true : 1;
[visits0_1_1] true : 1;
```

```
[visits0_2_0] true : 1;
[visits0_2_1] true : 1;
[visits4_3_0] true : 1;
[visits4_3_1] true : 1;
[visits4_5_0] true : 1;
[visits4_5_1] true : 1;
[visits5_6_0] true : 2;
[visits5_6_1] true : 2;
[visits6_5_0] true : 1;
[visits6_5_1] true : 1;
endrewards
//Automated PRISM Code Generation
```

Listing C.1: PRISM Model for Grid-Based Search and Rescue Domain. Automated Generation.

Bibliography

[1] Abbeel, P., Coates, A., Quigley, M. and Ng, A. [2006], 'An application of reinforcement learning to aerobatic helicopter flight', *Advances in neural information processing systems* **19**.

[2] Aceto, L., Ingólfsdóttir, A., Larsen, K. G. and Srba, J. [2007], *Reactive systems: modelling, specification and verification*, cambridge university press.

[3] Agha, G. and Palmskog, K. [2018], 'A survey of statistical model checking', *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **28**(1), 1–39.

[4] Akanksha, E., Sharma, N., Gulati, K. et al. [2021], Review on reinforcement learning, research evolution and scope of application, *in* '2021 5th International Conference on Computing Methodologies and Communication (ICCMC)', IEEE, pp. 1416–1423.

[5] Allen, J. [2017], *Reactive design patterns.*, Simon & Schuster.

[6] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S. and Topcu, U. [2018], Safe reinforcement learning via shielding, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 32.

[7] Altmann, J. and Gubrud, M. [2004], 'Anticipating military nanotechnology', *IEEE Technology and Society Magazine* **23**(4), 33–40.

[8] Arcuri, A. and Briand, L. [2011], A practical guide for using statistical tests to assess randomized algorithms in software engineering, *in* '2011 33rd International Conference on Software Engineering (ICSE)', IEEE, pp. 1–10.

[9] Arulkumaran, K., Deisenroth, M. P., Brundage, M. and Bharath, A. A. [2017], 'Deep reinforcement learning: A brief survey', *IEEE Signal Processing Magazine* **34**(6), 26–38.

[10] Autili, M., Grunske, L., Lumpe, M., Pelliccione, P. and Tang, A. [2015], 'Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar', *IEEE Transactions on Software Engineering* **41**(7), 620–638.

[11] Bacci, E. and Parker, D. [2022], Verified probabilistic policies for deep reinforcement learning, *in* 'NASA Formal Methods Symposium', Springer, pp. 193–212.

[12] Bakir, M. E., Gheorghe, M., Konur, S. and Stannett, M. [2016], Comparative analysis of statistical model checking tools, *in* 'International Conference on Membrane Computing', Springer, pp. 119–135.

[13] Bastani, O., Li, S. and Xu, A. [2021], Safe reinforcement learning via statistical model predictive shielding., *in* 'Robotics: Science and Systems'.

[14] Basu, A., Bhattacharyya, T. and Borkar, V. S. [2008], 'A learning algorithm for risk-sensitive cost', *Mathematics of operations research* **33**(4), 880–898.

[15] Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L. and Schnoebelen, P. [2013], *Systems and software verification: model-checking techniques and tools*, Springer Science & Business Media.

[16] Berger-Tal, O., Nathan, J., Meron, E. and Saltz, D. [2014], 'The exploration-exploitation dilemma: a multidisciplinary framework', *PloS one* **9**(4), e95693.

[17] Berkenkamp, F., Turchetta, M., Schoellig, A. and Krause, A. [2017], 'Safe model-based reinforcement learning with stability guarantees', *Advances in neural information processing systems* **30**.

[18] Bhalla, S., Ganapathi Subramanian, S. and Crowley, M. [2020], Deep multi agent reinforcement learning for autonomous driving, *in* 'Canadian Conference on Artificial Intelligence', Springer, pp. 67–78.

[19] Bishop, C. M. [1994], 'Neural networks and their applications', *Review of scientific instruments* **65**(6), 1803–1832.

[20] Bisi, L., Sabbioni, L., Vittori, E., Papini, M. and Restelli, M. [2019], 'Risk-averse trust region optimization for reward-volatility reduction', *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* pp. 4583–4589.

[21] Blakley, B., McDermott, E. and Geer, D. [2001], Information security is information risk management, *in* 'Proceedings of the 2001 workshop on New security paradigms', pp. 97–104.

[22] Bowling, M. and Veloso, M. [2000], An analysis of stochastic game theory for multiagent reinforcement learning, Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science.

[23] Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J. and Schoellig, A. P. [2022], 'Safe learning in robotics: From learning-based control to safe reinforcement learning', *Annual Review of Control, Robotics, and Autonomous Systems* **5**, 411–444.

[24] Buşoniu, L., Babuška, R. and Schutter, B. D. [2010], 'Multi-agent reinforcement learning: An overview', *Innovations in multi-agent systems and applications-1* pp. 183–221.

[25] Cai, M. and Vasile, C.-I. [2021], 'Safe-critical modular deep reinforcement learning with temporal logic through gaussian processes and control barrier functions', *arXiv preprint arXiv:2109.02791* .

[26] Cai, Z., Cao, H., Lu, W., Zhang, L. and Xiong, H. [2021], 'Safe multi-agent reinforcement learning through decentralized multiple control barrier functions', *arXiv preprint arXiv:2103.12553* .

[27] Calinescu, R., Kikuchi, S. and Johnson, K. [2012], Compositional reverification of probabilistic safety properties for large-scale complex it systems, *in* 'Monterey Workshop', Springer, pp. 303–329.

[28] Chatterjee, K., Majumdar, R. and Henzinger, T. A. [2006], Markov decision processes with multiple objectives, *in* 'Annual symposium on theoretical aspects of computer science', Springer, pp. 325–336.

[29] Cheng, R., Khojasteh, M. J., Ames, A. D. and Burdick, J. W. [2020], Safe multi-agent interaction through robust control barrier functions with learned uncertainties, *in* '2020 59th IEEE Conference on Decision and Control (CDC)', IEEE, pp. 777–783.

[30] Choi, J., Castaneda, F., Tomlin, C. J. and Sreenath, K. [2020], 'Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions', *Robotics: Science and Systems* .

[31] Chow, Y., Ghavamzadeh, M., Janson, L. and Pavone, M. [2017], 'Risk-constrained reinforcement learning with percentile risk criteria', *The Journal of Machine Learning Research* **18**(1), 6070–6120.

[32] Chow, Y., Nachum, O., Duenez-Guzman, E. and Ghavamzadeh, M. [2018], 'A lyapunov-based approach to safe reinforcement learning', *Advances in neural information processing systems* **31**.

[33] Clarke, E. M. and Emerson, E. A. [1981], Design and synthesis of synchronization skeletons using branching time temporal logic, *in* 'Workshop on logic of programs', Springer, pp. 52–71.

[34] Clarke, E. M., Klieber, W., Nováček, M. and Zuliani, P. [2011], Model checking and the state explosion problem, *in* 'LASER Summer School on Software Engineering', Springer, pp. 1–30.

[35] Clouse, J. A. and Utgoff, P. E. [1992], A teaching method for reinforcement learning, *in* 'Machine learning proceedings 1992', Elsevier, pp. 92–101.

[36] Dehnert, C., Junges, S., Katoen, J.-P. and Volk, M. [2017], A storm is coming: A modern probabilistic model checker, *in* 'International Conference on Computer Aided Verification', Springer, pp. 592–600.

[37] den Hengst, F., François-Lavet, V., Hoogendoorn, M. and van Harmelen, F. [2022], 'Planning for potential: efficient safe reinforcement learning', *Machine Learning* pp. 1–20.

[38] ElSayed-Aly, I., Bharadwaj, S., Amato, C., Ehlers, R., Topcu, U. and Feng, L. [2021], 'Safe multi-agent reinforcement learning via shielding', *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* .

[39] ElSayed-Aly, I. and Feng, L. [2022], 'Logic-based reward shaping for multi-agent reinforcement learning', *arXiv preprint arXiv:2206.08881* .

[40] Eriksson, H., Basu, D., Alibeigi, M. and Dimitrakakis, C. [2022], 'Risk-sensitive bayesian games for multi-agent reinforcement learning under policy uncertainty', *Cornell* .

[41] Fan, J. and Li, W. [2019], 'Safety-guided deep reinforcement learning via online gaussian process estimation', *SafeML ICLR 2019 Workshop* .

[42] Fei, Y., Yang, Z., Chen, Y., Wang, Z. and Xie, Q. [2020], 'Risk-sensitive reinforcement learning: Near-optimal risk-sample tradeoff in regret', *Advances in Neural Information Processing Systems* **33**, 22384–22395.

[43] Forejt, V., Kwiatkowska, M., Norman, G. and Parker, D. [2011], Automated verification techniques for probabilistic systems, *in* 'International school on formal methods for the design of computer, communication and software systems', Springer, pp. 53–113.

[44] Garcia, F. and Rachelson, E. [2013], 'Markov decision processes', *Markov Decision Processes in Artificial Intelligence* pp. 1–38.

[45] Garcia, J. and Fernández, F. [2012], 'Safe exploration of state and action spaces in reinforcement learning', *Journal of Artificial Intelligence Research* **45**, 515–564.

[46] Garcıa, J. and Fernández, F. [2015], 'A comprehensive survey on safe reinforcement learning', *Journal of Machine Learning Research* **16**(1), 1437–1480.

[47] Gaskett, C. [2003], 'Reinforcement learning under circumstances beyond its control'.

[48] Geibel, P. [2006], Reinforcement learning for mdps with constraints, *in* 'European Conference on Machine Learning', Springer, pp. 646–653.

[49] Geibel, P. and Wysotzki, F. [2005], 'Risk-sensitive reinforcement learning applied to control under constraints', *Journal of Artificial Intelligence Research* **24**, 81–108.

[50] Gerasimou, S., Calinescu, R. and Banks, A. [2014], Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration, *in* 'Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems', pp. 115–124.

[51] Gopalan, N., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., Winder, J., Wong, L. L. et al. [2017], Planning with abstract markov decision processes, *in* 'Twenty-Seventh International Conference on Automated Planning and Scheduling'.

[52] Gosavi, A. [2015], Control optimization with stochastic dynamic programming, *in* 'Simulation-Based Optimization', Springer, pp. 123–195.

[53] Grunske, L. [2008], Specification patterns for probabilistic quality properties, *in* '2008 ACM/IEEE 30th International Conference on Software Engineering', pp. 31–40.

[54] Gu, S., Kuba, J. G., Wen, M., Chen, R., Wang, Z., Tian, Z., Wang, J., Knoll, A. and Yang, Y. [2021], 'Multi-agent constrained policy optimisation', *arXiv preprint arXiv:2110.02793* .

[55] Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., Yang, Y. and Knoll, A. [2022], 'A review of safe reinforcement learning: Methods, theory and applications', *arXiv preprint arXiv:2205.10330* .

[56] Gupta, S., Singal, G. and Garg, D. [2021], 'Deep reinforcement learning techniques in diversified domains: a survey', *Archives of Computational Methods in Engineering* **28**(7), 4715–4754.

[57] Hahn, E. M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Křetínský, J., Parker, D., Quatmann, T., Ruijters, E. and Steinmetz, M. [2019], The 2019 comparison of tools for the analysis of quantitative formal models, *in* 'International Conference on Tools and Algorithms for the Construction and Analysis of Systems', Springer, pp. 69–92.

[58] Haney, B. S. [2020], 'Applied artificial intelligence in modern warfare and national security policy', *Hastings Sci. & Tech. LJ* **11**, 61.

[59] Hans, A., Schneegaß, D., Schäfer, A. M. and Udluft, S. [2008], Safe exploration for reinforcement learning., *in* 'ESANN', Citeseer, pp. 143–148.

[60] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I. et al. [2018], Deep q-learning from demonstrations, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 32.

[61] Hewing, L., Wabersich, K. P., Menner, M. and Zeilinger, M. N. [2020], 'Learning-based model predictive control: Toward safe learning in control', *Annual Review of Control, Robotics, and Autonomous Systems* **3**, 269–296.

[62] Huang, Y., Wu, S., Mu, Z., Long, X., Chu, S. and Zhao, G. [2020], A multi-agent reinforcement learning method for swarm robots in space collaborative exploration, *in* '2020 6th International Conference on Control, Automation and Robotics (ICCAR)', IEEE, pp. 139–144.

[63] Hussain, R. and Zeadally, S. [2018], 'Autonomous cars: Research results, issues, and future challenges', *IEEE Communications Surveys & Tutorials* **21**(2), 1275–1313.

[64] Iqbal, J., Tahir, A. M., ul Islam, R. et al. [2012], Robotics for nuclear power plants—challenges and future perspectives, *in* '2012 2nd international conference on applied robotics for the power industry (CARPI)', IEEE, pp. 151–156.

[65] Jansen, D. N., Katoen, J.-P., Oldenkamp, M., Stoelinga, M. and Zapreev, I. [2007], How fast and fat is your probabilistic model checker? an experimental performance comparison, *in* 'Haifa verification conference', Springer, pp. 69–85.

[66] Johnson, J. [2019], 'Artificial intelligence & future warfare: implications for international security', *Defense & Security Analysis* **35**(2), 147–169.

[67] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M. et al. [2018], 'Unity: A general platform for intelligent agents', *arXiv preprint arXiv:1809.02627*.

[68] Junges, S., Jansen, N., Dehnert, C., Topcu, U. and Katoen, J.-P. [2016], Safety-constrained reinforcement learning for mdps, *in* 'International conference on tools and algorithms for the construction and analysis of systems', Springer, pp. 130–146.

[69] Kadota, Y., Kurano, M. and Yasuda, M. [2006], 'Discounted markov decision processes with utility constraints', *Computers & Mathematics with Applications* **51**(2), 279–284.

[70] Kourie, D. G. and Watson, B. W. [2012], *The Correctness-by-Construction Approach to Programming*, Vol. 264, Springer.

[71] Kuba, J. G., Chen, R., Wen, M., Wen, Y., Sun, F., Wang, J. and Yang, Y. [2021], 'Trust region policy optimisation in multi-agent reinforcement learning', *ICLR 2022 Conference*.

[72] Kwiatkowska, M. [2007], Quantitative verification: models techniques and tools, *in* 'Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering', pp. 449–458.

[73] Kwiatkowska, M., Norman, G. and Parker, D. [2011], Prism 4.0: Verification of probabilistic real-time systems, *in* 'International conference on computer aided verification', Springer, pp. 585–591.

[74] Kyrarini, M., Lygerakis, F., Rajavenkatanarayanan, A., Sevastopoulos, C., Nambiappan, H. R., Chaitanya, K. K., Babu, A. R., Mathew, J. and Makedon, F. [2021], 'A survey of robots in healthcare', *Technologies* **9**(1), 8.

[75] Lanham, M. [2018], *Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games*, Packt Publishing Ltd.

[76] Law, E. L. [2005], 'Risk-directed exploration in reinforcement learning'.

[77] Li, L., Walsh, T. J. and Littman, M. L. [2006], 'Towards a unified theory of state abstraction for mdps.', *ISAIM* **4**(5), 9.

[78] Li, W., Wang, X., Jin, B., Sheng, J. and Zha, H. [2021], 'Dealing with non-stationarity in multi-agent reinforcement learning via trust region decomposition', *CLR 2022 Conference* .

[79] Littman, M. L. [1994], Markov games as a framework for multi-agent reinforcement learning, *in* 'Machine learning proceedings 1994', Elsevier, pp. 157–163.

[80] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y. and Alsaadi, F. E. [2017], 'A survey of deep neural network architectures and their applications', *Neurocomputing* **234**, 11–26.

[81] Liu, Y., Halev, A. and Liu, X. [2021], Policy learning with constraints in model-free reinforcement learning: A survey., *in* 'IJCAI', pp. 4508–4515.

[82] Manna, Z. and Pnueli, A. [2012], *Temporal verification of reactive systems: safety*, Springer Science & Business Media.

[83] Mannucci, T., van Kampen, E.-J., De Visser, C. and Chu, Q. [2017], 'Safe exploration algorithms for reinforcement learning controllers', *IEEE transactions on neural networks and learning systems* **29**(4), 1069–1081.

[84] Marthi, B. [2007], Automatic shaping and decomposition of reward functions, *in* 'Proceedings of the 24th International Conference on Machine learning', pp. 601–608.

[85] Marvi, Z. and Kiumarsi, B. [2021], 'Safe reinforcement learning: A control barrier function optimization approach', *International Journal of Robust and Nonlinear Control* **31**(6), 1923–1940.

[86] Mason, G. [2018], Safe Reinforcement Learning Using Formally Verified Abstract Policies, PhD thesis, University of York.

[87] Mason, G. R., Calinescu, R. C., Kudenko, D. and Banks, A. [2017], Assured reinforcement learning with formally verified abstract policies, *in* '9th International Conference on Agents and Artificial Intelligence (ICAART)', York.

[88] Matignon, L., Laurent, G. J. and Le Fort-Piat, N. [2012], 'Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems', *The Knowledge Engineering Review* **27**(1), 1–31.

[89] Mehta, M., Palade, V. and Chatterjee, I. [2023], *Explainable AI: Foundations, methodologies and applications*, Vol. 1 of *Intelligent Systems Reference Library*, 1 edn, SPRINGER INTERNATIONAL PU.

[90] Meng, W., Zheng, Q., Shi, Y. and Pan, G. [2021], 'An off-policy trust region policy optimization method with monotonic improvement guarantee for deep reinforcement learning', *IEEE Transactions on Neural Networks and Learning Systems* **33**(5), 2223–2235.

[91] Menghi, C., Tsigkanos, C., Askarpour, M., Pelliccione, P., Vazquez, G., Calinescu, R. and Garcia, S. [2022], 'Mission specification patterns for mobile robots: Providing support for quantitative properties', *IEEE Transactions on Software Engineering* .

[92] Menghi, C., Tsigkanos, C., Berger, T. and Pelliccione, P. [2019], Psalm: Specification of dependable robotic missions, *in* '2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)', IEEE, pp. 99–102.

[93] Mihatsch, O. and Neuneier, R. [2002], 'Risk-sensitive reinforcement learning', *Machine learning* **49**(2), 267–290.

[94] Moldovan, T. and Abbeel, P. [2012*a*], 'Risk aversion in markov decision processes via near optimal chernoff bounds', *Advances in neural information processing systems* **25**.

[95] Moldovan, T. M. and Abbeel, P. [2012*b*], 'Safe exploration in markov decision processes', *Proceedings of the 29th International Coference on International Conference on Machine Learning* .

[96] Mqirmi, P. E., Belardinelli, F. and León, B. G. [2021], 'An abstraction-based method to check multi-agent deep reinforcement-learning behaviors', *arXiv preprint arXiv:2102.01434* .

[97] Nagatani, K., Kiribayashi, S., Okada, Y., Otake, K., Yoshida, K., Tadokoro, S., Nishimura, T., Yoshida, T., Koyanagi, E., Fukushima, M. et al. [2013], 'Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots', *Journal of Field Robotics* **30**(1), 44–63.

[98] Ndousse, K. K., Eck, D., Levine, S. and Jaques, N. [2021], Emergent social learning via multi-agent reinforcement learning, *in* 'International Conference on Machine Learning', PMLR, pp. 7991–8004.

[99] Nguyen, T. T., Nguyen, N. D. and Nahavandi, S. [2020], 'Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications', *IEEE transactions on cybernetics* **50**(9), 3826–3839.

[100] Onyekpe, U., Lu, Y., Apostolopoulou, E., Palade, V., Eyo, E. U. and Kanarachos, S. [2023], Explainable machine learning for autonomous vehicle positioning using shap, *in* 'Explainable AI: Foundations, Methodologies and Applications', Springer, pp. 157–183.

[101] OroojlooyJadid, A. and Hajinezhad, D. [2019], 'A review of cooperative multi-agent deep reinforcement learning', *Applied Intelligence* .

[102] Panesar, S., Cagle, Y., Chander, D., Morey, J., Fernandez-Miranda, J. and Kliot, M. [2019], 'Artificial intelligence and the future of surgical robotics', *Annals of surgery* **270**(2), 223–226.

[103] Paternain, S., Calvo-Fullana, M., Chamon, L. F. and Ribeiro, A. [2019], Learning safe policies via primal-dual methods, *in* '2019 IEEE 58th Conference on Decision and Control (CDC)', IEEE, pp. 6491–6497.

[104] Pathak, D., Agrawal, P., Efros, A. A. and Darrell, T. [2017], Curiosity-driven exploration by self-supervised prediction, *in* 'International conference on machine learning', PMLR, pp. 2778–2787.

[105] Pecka, M., Šalanskỳ, V., Zimmermann, K. and Svoboda, T. [2016], Autonomous flipper control with safety constraints, *in* '2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 2889–2894.

[106] Pecka, M. and Svoboda, T. [2014], Safe exploration techniques for reinforcement learning–an overview, *in* 'International Workshop on Modelling and Simulation for Autonomous Systems', Springer, pp. 357–375.

[107] Pecka, M., Zimmermann, K. and Svoboda, T. [2015], Safe exploration for reinforcement learning in real unstructured environments, *in* 'Proc. of the Computer Vision Winter Workshop'.

[108] Perkins, T. J. and Barto, A. G. [2001], Lyapunov-constrained action sets for reinforcement learning, *in* 'ICML', Vol. 1, pp. 409–416.

[109] Platt, R. [2018], 'Markov decision process (MDP)', `https://www.ccs.neu.edu/home/rplatt/cs7180_fall2018/slides/mdps.pdf/`.
Online; accessed 10 February 2022.

[110] Portugal, D., Iocchi, L. and Farinelli, A. [2019], A ros-based framework for simulation and benchmarking of multi-robot patrolling algorithms, *in* 'Robot Operating System (ROS)', Springer, pp. 3–28.

[111] Puterman, M. L. [1990], 'Markov decision processes', *Handbooks in operations research and management science* **2**, 331–434.

[112] Qiu, W., Wang, X., Yu, R., He, X., Wang, R., An, B., Obraztsova, S. and Rabinovich, Z. [2020], 'Rmix: Risk-sensitive multi-agent reinforcement learning', *ICLR 2021 Conference* .

[113] Quintía Vidal, P., Iglesias Rodríguez, R., Rodríguez González, M. Á. and Vázquez Regueiro, C. [2013], 'Learning on real robots from experience and simple user feedback', *Journal Of Physical Agents, Vol. 7, NO. 1,* .

[114] Riley, J. [2020], 'Assured multi-agent reinforcement learning using quantitative verification', *In Doctoral Consortium on Agents and Artificial Intelligence (DCAART2021)* .

[115] Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A. [2021*a*], Reinforcement learning with quantitative verification for assured multi-agent policies, *in* '13th International Conference on Agents and Artificial Intelligence', York.

[116] Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A. [2021*b*], 'Utilising assured multi-agent reinforcement learning within safety-critical scenarios', *Procedia Computer Science* **192**, 1061–1070.

[117] Riley, J., Calinescu, R., Paterson, C., Kudenko, D. and Banks, A. [2022], Assured deep multi-agent reinforcement learning for safe robotic systems, *in* 'International Conference on Agents and Artificial Intelligence', Springer, pp. 158–180.

[118] Sampedro, C., Rodriguez-Ramos, A., Bavle, H., Carrio, A., de la Puente, P. and Campoy, P. [2019], 'A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques', *Journal of Intelligent & Robotic Systems* **95**(2), 601–627.

[119] Scharre, P. [2018], 'How swarming will change warfare', *Bulletin of the atomic scientists* **74**(6), 385–389.

[120] Schatten, M., Tomičić, I. and Đurić, B. O. [2017], A review on application domains of large-scale multiagent systems, *in* 'Central european conference on information and intelligent systems', Faculty of Organization and Informatics Varazdin, pp. 201–206.

[121] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. [2017], 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347* .

[122] Sharma, S., Sharma, S. and Athaiya, A. [2017], 'Activation functions in neural networks', *towards data science* **6**(12), 310–316.

[123] Shyamsundar, S., Mannucci, T. and van Kampen, E.-J. [2016], Reinforcement learning based algorithm with safety handling and risk perception, *in* '2016 IEEE Symposium Series on Computational Intelligence (SSCI)', IEEE, pp. 1–7.

[124] Singh, A., Halpern, Y., Thain, N., Christakopoulou, K., Chi, E., Chen, J. and Beutel, A. [2020], Building healthy recommendation sequences for everyone: A safe reinforcement learning approach, *in* 'FAccTRec Workshop'.

[125] Sniedovich, M. [1991], *Dynamic programming*, Vol. 297, CRC press.

[126] Sutton, R. S. and Barto, A. G. [2018], *Reinforcement learning: An introduction*, MIT press.

[127] Sutton, R. S., Precup, D. and Singh, S. [1999], 'Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning', *Artificial intelligence* **112**(1-2), 181–211.

[128] Tan, M. [1993], Multi-agent reinforcement learning: Independent vs. cooperative agents, *in* 'Proceedings of the tenth international conference on machine learning', pp. 330–337.

[129] Thananjeyan, B., Balakrishna, A., Nair, S., Luo, M., Srinivasan, K., Hwang, M., Gonzalez, J. E., Ibarz, J., Finn, C. and Goldberg, K. [2021], 'Recovery rl: Safe reinforcement learning with learned recovery zones', *IEEE Robotics and Automation Letters* **6**(3), 4915–4922.

[130] Thiebes, S., Lins, S. and Sunyaev, A. [2021], 'Trustworthy artificial intelligence', *Electronic Markets* **31**(2), 447–464.

[131] Torrado, R. R., Bontrager, P., Togelius, J., Liu, J. and Perez-Liebana, D. [2018], Deep reinforcement learning for general video game ai, *in* '2018 IEEE Conference on Computational Intelligence and Games (CIG)', IEEE, pp. 1–8.

[132] Varga, B., Kulcsár, B. and Chehreghani, M. H. [2021], 'Constrained policy gradient method for safe and fast reinforcement learning: a neural tangent kernel based approach', *International Conference on Machine Learning* .

[133] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P. et al. [2019], 'Grandmaster level in starcraft ii using multi-agent reinforcement learning', *Nature* **575**(7782), 350–354.

[134] Viseras, A., Wiedemann, T., Manss, C., Magel, L., Mueller, J., Shutin, D. and Merino, L. [2016], Decentralized multi-agent exploration with online-learning of gaussian processes, *in* '2016 IEEE International Conference on Robotics and Automation (ICRA)', IEEE, pp. 4222–4229.

[135] Vogel, T., Carwehl, M., Rodrigues, G. N. and Grunske, L. [2023], 'A property specification pattern catalog for real-time system verification with uppaal', *Information and Software Technology* **154**, 107100.

[136] Wang, H., Lei, Z., Zhang, X., Zhou, B. and Peng, J. [2016], 'Machine learning basics', *Deep learning* pp. 98–164.

[137] Wang, X. and Sandholm, T. [2002], 'Reinforcement learning to play an optimal nash equilibrium in team markov games', *Advances in neural information processing systems* **15**.

[138] Watkins, C. J. and Dayan, P. [1992], 'Q-learning', *Machine learning* **8**(3), 279–292.

[139] Wooldridge, M. [2009], *An introduction to multiagent systems*, John wiley & sons.

[140] Xu, J., Li, B., Lu, B., Liu, Y.-H., Dou, Q. and Heng, P.-A. [2021], Surrol: An open-source reinforcement learning centered and dvrk compatible platform for surgical robot learning, *in* '2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 1821–1828.

[141] Yang, Q., Simão, T. D., Tindemans, S. H. and Spaan, M. T. [2021], Wcsac: Worst-case soft actor critic for safety-constrained reinforcement learning., *in* 'AAAI', pp. 10639–10646.

[142] Yang, T.-Y., Rosca, J., Narasimhan, K. and Ramadge, P. J. [2021], Accelerating safe reinforcement learning with constraint-mismatched baseline policies, *in* 'International Conference on Machine Learning', PMLR, pp. 11795–11807.

[143] Younes, H. L., Kwiatkowska, M., Norman, G. and Parker, D. [2006], 'Numerical vs. statistical probabilistic model checking', *International Journal on Software Tools for Technology Transfer* **8**(3), 216–228.

[144] Zhang, K., Sun, T., Tao, Y., Genc, S., Mallya, S. and Basar, T. [2020], 'Robust multi-agent reinforcement learning with model uncertainty', *Advances in neural information processing systems* **33**, 10571–10583.

[145] Zhang, K., Yang, Z. and Başar, T. [2021], 'Multi-agent reinforcement learning: A selective overview of theories and algorithms', *Handbook of Reinforcement Learning and Control* pp. 321–384.

[146] Zhang, Q., Dong, H. and Pan, W. [2020], Lyapunov-based reinforcement learning for decentralized multi-agent control, *in* 'International Conference on Distributed Artificial Intelligence', Springer, pp. 55–68.

[147] Zhang, W., Bastani, O. and Kumar, V. [2019], 'Mamps: Safe multi-agent reinforcement learning via model predictive shielding', *International Conference on Autonomous Agents and Multiagent Systems* .

[148] Zhou, Z.-H. [2021], *Machine learning*, Springer Nature.