Reinforcement Learning for a Hunter and Prey Robot

Suraj Murali
Abstract

Reinforcement Learning for a Hunter and Prey Robot

Suraj Murali

The surge in the use of adaptive Artificial Intelligent (AI) systems have been made possible by leveraging the increasing processing and storage power that modern computers are able to provide. These systems are designed to make quality decisions that assist in making predictions in a wide variety of application fields. When such a system is fueled by data, the foundation for a Machine Learning (ML) approach can be modelled. Reinforcement Learning (RL) is an active model of ML going beyond the traditional supervised or unsupervised ML methods. RL studies algorithms to take actions so that the resulting reward is expected to be optimal. This thesis investigates the use of methods of RL in a context where the reward is highly time-varying: a setup is studied where two agents compete for a common resource. Specifically, we study a robotic setting inspired by the "cat-and-mouse" (hunter-prey) game. We refer to the hunter robot as to Tom, and to the competing prey robot as Jerry. In order to study this setup, two practical setups are considered. The first one is based on a LEGO platform, enabling us to run a number of actual experiments. The second one is based on a known RL simulator environment, enabling us to run many virtual experiments. We use these environments to explore the setting: indicate the non-stationary solution it generates, evaluate a number of de-facto standard approaches to RL, and identify key future avenues to be addressed.
Acknowledgements

This master thesis would not have been possible without the supervision from Rasmus Muhrbeck and Kristiaan Pelckmans. Kristiaan's vision and guidance has been a constant source of inspiration for me. It has been a lot of fun working with him and I will always be grateful for the support he has provided. I have been incredibly lucky to have him by my side.

The biggest influence to pursue this academic goal roots from my family members and I thank you all. My mom Lalitha, dad Murali and sis Sapna have been there constantly backing my pursuit. I have been fortunate to have them support me in chasing my dream and the thesis would have been far from possible without their love, motivation and comfort.

A big thanks to all my friends (you know who you are!), both back at home and in Uppsala, who have believed in me and always been there to fuel the journey with a lot of fun adventures, laughter and happiness. A special thanks to my childhood friend Srinandan who has been there from discussing the Doppler effect in high school to investigating Deep learning models more recently. Finally, I thank every faculty member of Uppsala’s Information and Technology Department whose courses and wise words have taught me valuable lessons in laying the foundations for making this thesis work possible.
Table of Contents

Acknowledgements I
List of Tables IV
List of Figures V
Summary of Notations VII

1 Introduction 1
1.1 The act of learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Reinforcement learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.3 Games . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.4 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.5 Application . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Background 4
2.1 Agents-Environment Interface . . . . . . . . . . . . . . . . . . . . . . . . . 4
2.2 Markov Decision Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2.1 Problem Formulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 Dynamic Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3.1 Policies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3.2 Value function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3.3 Optimal Policies and Optimal Value functions: . . . . . . . . . . . . 7
2.3.4 Infinite horizon problems . . . . . . . . . . . . . . . . . . . . . . . . . 7

3 Literature Review 9
3.1 Thompson Sampling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
3.2 Exploration-Exploitation in Tabular Reinforcement learning (RL) . . . . . . 10
3.2.1 Greedy exploration strategy . . . . . . . . . . . . . . . . . . . . . . . . . 10
3.2.2 Posterior sampling strategy . . . . . . . . . . . . . . . . . . . . . . . . . 11
3.3 Research questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

4 Methodology 12
4.1 Modeling the simulated world . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
4.1.1 States (S) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
4.1.2 Actions (A) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
4.1.3 Transition dynamics (P) . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

© Uppsala University
Suraj Murali
## Table of Contents

**4.1.4** T & J Agents .................................................. 17  
**4.1.5** Reward shaping and termination condition \((R, T)\) ........... 17  
**4.2** Modeling the Real World .......................................... 19  
**4.2.1** State to floor mapping ........................................... 20  
**4.2.2** Transition dynamics .............................................. 20  
**4.2.3** Avoiding obstacles and reward handshake ....................... 23  
**4.3** RL algorithms .................................................... 25  
**4.3.1** Problem formulation for T & J .................................. 26  

**5** Results ..................................................................... 30  
**5.1** Experiments in Real World ........................................... 30  
**5.1.1** Stage 1: Setting up the Tom robot ................................ 30  
**5.1.2** Stage 2: Setting up the Tom and Jerry robots .................. 34  
**5.2** Experiments in Simulated World .................................... 35  
**5.2.1** Setting up the simulator ............................................ 36  
**5.2.2** Experimental Results ................................................ 36  

**6** Discussion & Analysis .................................................. 38  
**6.1** Experiments in Real World ........................................... 38  
**6.2** Experiments in Simulator ............................................ 38  
**6.2.1** Analysis ............................................................ 39  

**7** Conclusion ................................................................... 44  
**7.1** Research Answers and Future Work ................................ 44  
**7.2** Summary .............................................................. 45  

**Literature** ...................................................................... 47  

**Appendices** ................................................................ 50  
**A** Results .................................................................... 50  
**A.1** Experiments in Simulated World .................................... 50  
**A.1.1** Setting up the simulator ............................................ 50
List of Tables

Table 4.1: Initial transition dynamics $P_{ss'}^a$, built for the $6 \times 6$ grid world in which the $(state, action)$ tuple determines from what state which action where $s,s' \in S$ and $a \in A$. $A = north, south, east, west$, represented by $a_↑, a_↓, a←, a→$.

Table 4.2: The transition probabilities of moving to the new states: 8, 13, 15, 20 (Fig.4.2) when choosing actions: north, south, east, west respectively from state $= 14$.

Table 4.3: The transition probabilities of moving to the new states: 19, 26 (Fig.4.2) when choosing actions: north, east respectively while remaining in the same state $= 25$, otherwise.

Table 5.1: Initializing the qValue table for the Tom robot. The action space is represented by $\{a_↑, a_↓, a←, a→\}$ to indicate the direction in which the robot moves based on Figure 5.1.

Table 5.2: Initializing the qValue table for the Tom robot.

Table 5.3: Initial qMax table at episode $= 0$, i.e before learning the q-values across the different states across timesteps over the finite learning episodes.

Table 5.4: qMax table for the obstacle-based and obstacle free states at the final episode, i.e. episode $= 10$. The values on the obstacle states has not been shown since they are influenced by the negative reward.

Table 5.5: Tom win ratio and time taken when simulated over 10,000 episodes.

Table 5.6: Jerry win ratio and time taken when simulated over 10,000 episodes.
List of Figures

Figure 2.1: The agent-environment interaction in a Markov decision process. At each time step $t$, the agent receives some representation of the environment’s state, $S_t \in S$, and then decides to select an action, $A_t \in A$. As a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in R$, and finds itself in a new state, $S_{t+1}$.  

Figure 4.1: 6 x 6 gridworld with states accessed by (row, col).  
Figure 4.2: 6 x 6 gridworld with state numbers.  
Figure 4.3: The action mapping on the grid world, $A = \{north, south, east, west\}$ of size $|A| = 4$.  
Figure 4.4: The tom agent represented as $T$ is initialized at $(row = 1, col = 4)$ or $state = 10$.  
Figure 4.5: The jerry agent represented as $J$ is initialized at $(row = 4, col = 1)$ or $state = 25$.  
Figure 4.6: The $T$ & $J$ agents in the grid world.  
Figure 4.7: The tom agent represented as $T$ surrounded by its sound field, depicted by the dark yellow region.  
Figure 4.8: The tom agent represented as $T$ surrounded by the sound field, depicted by the light yellow region.  
Figure 4.9: A possible terminating scenario when Tom catches Jerry in the 6 x 6 grid world.  
Figure 4.10: Top down view of the basic frame.  
Figure 4.11: Side down view of the basic frame.  
Figure 4.12: Top down view with the Gyro Sensor attached.  
Figure 4.13: Side view on attaching the ultrasonic sensors.  
Figure 4.14: Final build on attaching the sound sensor.  

Figure 5.1: Tom robot placed at state = 8 in a 5 x 5 grid world with actions = \{north, south, west, east\}. $Fd$ represents the forward distance and $Bd$ represents the backward distance as probed by the ultrasonic sensors.  
Figure 5.2: Tom and Jerry robots placed at state = 12 and state = 36 in a 7 x 7 grid world with actions = \{north, south, west, east\}.  

Figure 6.1: Random Walking Jerry(left) and Tom(right) over 10 x 10 grid with the action space \{north, south, east, west\}.
List of Figures

Figure 6.2: Greedy Jerry(left) and Greedy Tom(right) over 10 × 10 grid with the action space \{north, south, east, west\}. The exploration-exploitation parameter $\epsilon = 0.1$. ........................................ 40

Figure 6.3: PSRL Jerry and PSRL Tom over 10 × 10 grid with the action space \{north, south, east, west\}. ........................................ 41

Figure 6.4: PSRL Jerry and PSRL Tom with the action space \{north, south, east, west, stay\}. ........................................ 42

Figure 6.5: PSRL Jerry and Greedy Tom with the action space \{north, south, east, west\}. ........................................ 42
### Summary of Notations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial intelligence</td>
</tr>
<tr>
<td>ML</td>
<td>Machine learning</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement learning</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov decision process</td>
</tr>
<tr>
<td>DP</td>
<td>Dynamic programming</td>
</tr>
<tr>
<td>OFU</td>
<td>Optimism in the face of uncertainty</td>
</tr>
<tr>
<td>PSRL</td>
<td>Posterior sampling for reinforcement learning</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-integral-derivative</td>
</tr>
</tbody>
</table>
1 Introduction

The surge in Artificial intelligence (AI) systems has been made possible by the increasing processing and storage power that modern computers are able to provide. With the emergence of the "Big Data" revolution, these systems - fueled by the abundance of data - are able to make quality decisions that impact a wide range of fields. In this thesis we aim to focus on the field of robotics, where the potential impact of artificial intelligence is huge. From agriculture to industries to homes, robotic solutions can be extremely beneficial in the way forward. From autonomous robots that can plan optimal strategies, such as to water crops on a field, to autonomous systems that can facilitate in aiding humans in industries or transportation - the central objectives and individual design of these robots are innumerable. Despite the varying forms the robots can take, the task of effectively learning and acting on their decision originates from the datasets that guide the training of such robots. In this thesis we seek to apply Machine learning (ML) techniques that can learn effectively from the available datasets in order to acquire insight into how such learning systems are built towards solving robotic problems.

1.1 The act of learning

The broad field of ML can be subdivided into three major approaches that provide a framework to think about the problems design [1]. The first two approaches, supervised and unsupervised learning focuses on learning from a fixed dataset. But in the case of robotics, the decision-maker, i.e. the robot should learn to adapt in different and changing environments, making it hard to find fixed datasets that precisely matches every possible scenario the robot might encounter. Given the enormous generalization capability required in challenging environments, the final approach in ML, namely RL provides a framework to design systems where the decisions made can directly affect the data acquired.

1.2 Reinforcement learning

Reinforcement learning is learning what to do– how to map situations to actions– so as to maximize a numerical reward signal [2]. Rooting from behavioral psychology this learning mechanism is an autonomous, self-teaching system that learns by the process of trial-and-error. On learning what actions provide maximum rewards, the RL system performs the task over and over again until it perfects its strategy towards what it aims to achieve.
Cat-and-mouse game: This thesis aims to make use of the RL framework to study the behaviour between two robots that take on the role of a hunter and prey in a "cat-and-mouse" game. In this setting, the hunter (TOM) tries to catch the prey (JERRY), who in turn tries to outsmart TOM. That is, the hunter tries to anticipate the position of its prey, while the prey tries to anticipate the hunter’s position. When both are equally capable, this enables the possibility of both to build up smarts beyond the mere task. The setting of these two duelling RL-driven robots makes this project reminiscent to methods of adversarial networks. It also makes this model highly time-varying, defying current theoretical understanding of such methods.

1.3 Games

Games and AI have a long history with each other [3]. A game can be considered a simulation that distills the interesting aspects of decision-making and forward-thinking capabilities. More formally, game theory is the study of mathematical models of strategic interaction between rational decision-makers [4]. On one hand, we could consider game models that considers just two rational decision-makers and study the depths of how these two decision-makers complement or contradict each other. On the other hand, we could have game models that considers multiple populations of decision-makers to study their overall evolution of strategies over time. On providing hypothetical scenarios within the confines of a game, game theory has a wide range of applications from psychology to evolutionary biology to economics.

A Zero-sum game: This thesis addresses the case study of a zero-sum game [5], i.e. one entity’s gain results in the loss for the other entity, where the two robots (TOM and JERRY) play the roles of a hunter and prey respectively. This simple setting, allows both Tom and Jerry to constantly devise evolving strategies to predict each others behavior and opportunities to outsmart each other. The RL framework fits perfectly into how such a game can be designed, analyzed and tested - providing an interesting lens to observe some form of intelligent behavior. The autonomous control problem can be established in both simulated worlds and the real world. Thus by focusing on the high-level path planning problem, we aim to reveal some of the strategies that guide the robots. This application of RL is slightly different from "standard RL" as the dynamics of such a setting never end up in equilibrium, i.e. there is no fixed strategy that will always work.

1.4 Organization

We begin our investigation by first introducing in Chapter 2 the key terminologies that is used in RL. This chapter also describes the general problem formulation as finite Markov decision process (MDP) and then discusses ideas such as, Dynamic programming (DP), that help solve for a finite MDP. In Chapter 3 we will discuss existing literature and
Introduction

reflect on how we aim to incorporate the literature into our case study. We will also address some of the research questions the thesis puts forward. The Chapter 4 will scale the cat-and-mouse game progressively, from simulations to the real world and then provide insight into the reinforcement algorithms considered. We investigate tabular solution methods in a *tabular rasa* or “blank slate” setting, one in which no or little prior knowledge is provided for the decision-makers. Chapter 5 follows on with the experiments carried out. Chapter 6 will then analyze and discuss over the results obtained. Finally, conclusion and future works are presented in Chapter 7.

1.5 Application

This thesis is intended to deploy autonomous control onto a demonstration robot for conferences and fairs, that has been built from scratch [6]. The demonstration robot is meant to create lasting impressions at the company booth at conferences or fairs. However, during the construction of the demonstration robot, the thesis makes use of two custom-built support LEGO MINDSTORMS EV3 robots. On establishing autonomous control over these robots, we aim to bridge the theory to practice and test the performance of RL algorithms when used in the context of robotics.
2 Background

We begin our investigation by first introducing the core ideas that revolve around the design and solution to reinforcement learning problems. This thesis explores the simplest form of solving problems through the representation of data in the form of state-action tables. Commonly known as tabular reinforcement learning, this solution method often can find exact solutions if perfect models of the problem are available. We also consider reinforcement learning in the tabular rasa or "blank slate" setting, where little or no prior knowledge is available to the learner. While tabular solutions methods are not a practical approach to solving large scale problems, they allow for clean analysis of a simple problem. Using the simple "cat-and-mouse" case study carried out, this method provides insight into the core challenges of reinforcement learning when used in the context of robotics. These challenges are addressed in the later chapters of the thesis. In this chapter we will first familiarize ourself with the key terminologies used in the framework, learn the general problem formulation as a finite MDP and then discuss ideas including DP that help solve the finite MDP.

2.1 Agents-Environment Interface

The learner and decision maker is called the agent. Everything outside the agent, in other words, everything the agent interacts with is considered part of the environment. This environment is composed of states. The agent resides in one such state and selects actions thereby interacting with the environment. The environment in-turn responds to the action selected by presenting new situations to the agent. The environment also has some states with rewards in them. Thus, an agent will seek to maximize this reward over time through its choice of actions. Through a sequence of discrete time steps $t = \{0,1,2,\ldots\}$, the agent and environment interactions vary and Fig 2.1 illustrates the agent-environment life cycle. The agent and environment together thereby give rise to a sequence or trajectory that begins as:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots$$
2.2 Markov Decision Process

With this understanding of the agent-environment life cycle in Sec 2.1, the next step is to understand an MDP. An MDP is designed to be a straightforward framing of the problem of learning from interactions towards achieving a goal. MDPs are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made [7]. RL algorithms commonly operate in an episodic nature. An episode constitutes a sequence of discrete time steps and can terminate based on a problem specific condition. In this section we represent an MDP by using a tuple \( M = \langle S, A, P, R, T \rangle \), where \( M \) is an MDP.

2.2.1 Problem Formulation

We consider the problem of learning to optimize an MDP \( M = \langle S, A, P, R, T \rangle \) in repeated finite episodes of interaction.

- \( S = \{1, \ldots, S\} \) is the state space of size \( S \),
- \( A = \{1, \ldots, A\} \) is the action space of size \( A \),
- \( T \) is the length of the episode through a sequence of time steps \( t = \{1, \ldots, T\} \),
- \( P \) is a state transition probability matrix. \( P \) gives the probability of ending in state \( s_{t+1} \in S \), after picking \( a_t \in A \) in \( s_t \in S \) and follows the Markov property implying that future states of the process depend only upon the present state, not on the sequence of events that preceded it. Thus, \( P \) varies through the time steps \( t = \{1, \ldots, T\} \). This work only needs the simplified form where \( P_t(a, s) : A \times S \rightarrow S \) gives the subsequent state when taking action \( a \) in state \( s \) at time \( t \).
• \( \mathcal{R} \) is a reward function on transition from \( s \in \mathcal{S} \) to \( s' \in \mathcal{S} \) given \( a \in \mathcal{A} \). When agent \( an \) in state \( s_t \in \mathcal{S} \) picks action \( a_t \in \mathcal{A} \) and moves to the rewarding state \( s_{t+1} \in \mathcal{S} \), it receives an immediate reward \( r \) in \([0,1]\) from the environment.

When considering the transition dynamics and reward function it is important to note that if these elements are known they can be embedded as rules that govern the environment, hence can be regarded environment-centric on initialization. But when an agent is operating in this environment, it has to learn the transitions and rewards that the environment offers and hence the agent builds a belief of how it interprets these elements from an agent-centric perspective. This will be highlighted when modelling the cat-and-mouse problem in Chapter 4.

2.3 Dynamic Programming

"An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" - Bellman [8]

Beyond the agent and the environment in Sec 2.1 and 2.2, the remaining sub-elements of reinforcement learning system to explore are: a policy, a value function and a model of the environment. This section will discuss the first two sub-elements which are more agent-centric. With the basic understanding of how to design MDPs (Sec 2.2) for a reinforcement learning problem the next step would be to find methods that help solve the learning problem towards a solution. When considering a finite MDP that assumes a perfect model of the environment, the collection of algorithms in DP can be used to compute optimal policies. A reinforcement learning algorithm allows the agent to learn through trial and error experiences in order to develop a policy.

2.3.1 Policies

A policy defines the learning agent’s way of behavior at a given time. A (deterministic) policy \( \pi: \mathcal{S} \rightarrow \mathcal{A} \) is a function mapping each state \( s \in \mathcal{S} \).

2.3.2 Value function

The reward signal indicates a good decision in an immediate sense, while a value function specifies what is good in the long run. All of reinforcement learning algorithms involve estimating value functions.

**Action-value function:** For an MDP \( M \) and policy \( \pi \), we define the Q-value function \( Q_{\pi,t}^{M,T}(s,a) \), where state \( s \in \mathcal{S} \), taking action \( a \in \mathcal{A} \), on following the policy \( \pi \) for a finite horizon \( T \). Equivalently, we have the function \( \pi_t: \mathcal{S} \rightarrow \mathcal{S} \) which maps the previous state at time \( t \) through one iteration of the policy into a new state.
\[ Q_{\pi, t}^{M,T}(s, a) := \sum_{s=t+1}^{T} R_s(\tilde{\pi}_{s-1}(\ldots \tilde{\pi}_{t+1}(P_t(a, s)))) \] (1)

where \( R_t \) denotes the reward observed at time \( t \).

**State-value function:** This value function is the total reward collected by iterating the policy starting from state \( s \) at iteration \( t \) and defined as,

\[ V_{t}^{M,T}(s, \pi_t) := \sum_{s=t+1}^{T} R_s(\tilde{\pi}_{s-1}(\ldots \tilde{\pi}_{t+1}(\pi_t(s)))) \] (2)

Here, it is important to note that the action \( a \in A \) is decided based on the policy \( \pi_t \) followed by the agent at time \( t \). The most common way to decide this action is through a greedy action selection, i.e. choosing the action that has proven to lead to maximum reward over time.

### 2.3.3 Optimal Policies and Optimal Value functions:

The central idea is to find an optimal policy that maximizes the total reward over all policies. The optimal value function specifies the best possible performance in an MDP. The optimal policy denoted as \( \pi_t^* \) gives a value \( V_{t}^{M,M}(s, \pi_t^*) \), hence

\[ V_{t}^{M}(s, \pi_t^*) = \max_{\pi_t} \left( V_{t}^{M,T}(s, \pi_t) \right) \] (3)

Given an optimal policy \( ^* \), the corresponding Q-function is \( Q_{\pi_t^*, t}^{M}(s, a) \). The policy \( \pi_t^* \) is optimal for an MDP \( M \), for all \( s \in S, a \in A \) and \( t = \{1, \ldots T\} \) and defined as,

\[ Q_{\pi_t^*, t}^{M}(s, a) = \max_{\pi_t} \left( Q_{\pi_t^*, t}^{M,T}(s, a) \right) \] (4)

### 2.3.4 Infinite horizon problems

Once we understand the construction of an MDP, it is important to address the cases of finite and infinite horizon problems. When the sum of expected rewards through time is within a fixed horizon for the MDP \( M \), i.e. \( T < \infty \), they are considered a finite horizon problem. The setting where \( T = \infty \), states the infinite horizon problem. In fact, the finite horizon problem is a special case of an infinite horizon problem [9]. Since we can control the episodic nature by setting a finite number of episodes to run the "cat-and-mouse" game it assumes a case of a finite horizon problem. While there is active research in bridging the infinite and finite horizon problem, this thesis reduces the infinite horizon problem to a more manageable finite horizon problem with undiscounted reward to comprehend the analytical results obtained in the later chapters of the thesis. Generally,
the infinite reward case is bound by a discount factor, \( \gamma \) which in our deterministic task is very close to the value 1 and hence can be ignored [10].
3 Literature Review

In this chapter, we will review the various approaches that focus on the efficiency of exploration, usually expressed as to how quickly an RL algorithm can approach an optimal decision-making policy. Exploration is the task of examining new regions of state-action spaces in the quest to gather previously unknown information. On the other hand, exploitation handles the dual option of trying to make the best of the past experiences. One of the central problems in RL literature is a dilemma commonly known as the exploration-exploitation dilemma. The exploration-exploitation dilemma has been most studied in a stylized problem known as the multi-armed bandit. A multi-armed bandit setting considers only a single state that presents multiple actions to choose with varying reward feedback for the different actions selected.

Example 1: The original form of this multi-armed or k-armed bandit problem, derives its name from the analogy to a slot machine, or "one-armed bandit", except we have k levers to choose from rather than one. Each action selection thus resembles a gambler's play on one of the slot machine’s lever, and the rewards are the payoffs from hitting the jackpot. Thus, by repeatedly attempting different actions one is likely to maximize their winnings by concentrating on actions on the best lever.

Example 2: Another analogy as studied by Thompson, considers the problem of a doctor choosing between experimental treatments for a series of ill patients. Each action in this case, is the selection of treatments and the reward is the survival or well-being of the patients the drug is administered to. One possibility to select an actions at each step can be based on their posterior probability of being the best action. This method, of sampling from the posterior is called posterior sampling or Thompson sampling.

Regret bounds: The most important feature that distinguishes RL from other types of learning is that it uses training information that evaluates the actions taken rather than instructs that give the correct actions. The most common way to assess the performance of a learning algorithm given its environment is in terms of regret. The regret quantifies the sub-optimality of the policies chosen by the learning algorithm π versus the optimal policy up to time T. Therefore, maximizing the cumulative sum of rewards (maximizing the objective) is equivalent to minimizing the regret.
3.1 Thompson Sampling

Thompson Sampling is a technique where the implemented policy is a sample of the posterior distribution spanned by past evidence. This is one of the oldest heuristics for balancing exploration and exploitation by using probability matching. The agent maintains a posterior distribution for its belief over the optimal action. Initially, this agent has no or very little knowledge regarding the prior distribution or initial belief over what the optimal actions are. A purely exploitative strategy might never learn the optimal policy given its greedy nature. Probability matching instead randomly chooses its action randomly, according to the probability that it is optimal. This randomized sampling incentives exploration of poorly understood states and actions through the variance of the posterior. Thus, as this algorithm gathers more data about the MDP it is trying to solve, through randomly sampling and updating its observation onto the posterior belief, the posterior samples begin to concentrate around the true values behind the MDP thereby balancing exploration and exploitation. [9]

3.2 Exploration-Exploitation in Tabular RL

The design and analysis of tabular algorithms gives valuable insights into simple RL problems, but are not practical in problems where the state-action spaces are enormous (due to the curse of dimensionality). In this thesis, we make use of tabular solutions since they provide amenable analysis of simple tabular problems, such as the "cat-and-mouse" game studied in this thesis. Richard E. Bellman [12] was the first to show how DP could be used to compute the optimal balance between exploration and exploitation within a Bayesian formulation of the problem. When deciding between how to select policies, the two variants this thesis considers are:

3.2.1 Greedy exploration strategy

The literature offers a rich collection of such algorithms (e.g. Bertsekas et al. [13]; Sutton and Barto [7]; Szepesvari [14]; Powell [15] references therein). Some of the algorithms of this genre have achieved impressive outcomes, notably in games such as backgammon (Tesauro [16]), Atari arcade games (Mnih et al. [17]), mastering Go (Silver et al. [18]; [19]). The trademark of a greedy policy is the principle of Optimism in the face of uncertainty (OFU), which has driven the majority of progress in the field of efficient reinforcement learning. The upper confidence for reinforcement learning (UCRL) establishes this existing state of the art for provably efficient RL (Jaksch [20]). This optimistic principle is simple:

"Select the policy which would obtain the best possible rewards in the best plausible environment".

But, Osband (2017) [21] argues that, naive exploration schemes are used in, for example, Silver et al. (2016) [18] which is possibly the reason that these applications use enormous
quantities of data, i.e. trained with neural networks over hundreds of billions to trillions of simulated games. In the Osband (2016) computational results demonstrate that Posterior sampling for reinforcement learning (PSRL) dramatically outperforms algorithms driven by optimism.

3.2.2 Posterior sampling strategy

Ian Osband’s Ph.D. dissertation titled "Deep exploration via randomized value function" presents an alternative approach inspired by Thompson sampling heuristic for multi-armed bandits which suggests:

"To randomly select a policy according to the probability that it is optimal".

The Ph.D dissertation employs the Thompson sampling approach and extends the idea from the multi-arm bandit setting (which deals with only one state) to the finite MDP that define RL problems.

3.3 Research questions

In the previous chapters and sections, we have investigated the background, literature and principles that forms the foundation for approaching an RL problem. Before elaborating on the methodology and implementation details regarding how the cat-and-mouse game is setup, it is useful to summarize some challenging research questions that this thesis aims to target:

- Given the decisions made by the RL system directly affect the data acquired, how does one go about building such a system which actively engages with its datasets when compared to the fixed dataset approaches used in supervised and unsupervised learning?

- How does one interpret the solutions for this RL system?

- Can the decision-maker detect other time-varying entities in the model? If so, is it possible to highlight such evolving strategies?

- Is it possible to transfer knowledge between a simulation of the task and the real world task? If so, can an architecture be developed to unify the gathered simulated experience and the real experience?

- How can the decision-maker learn environments it has not previously trained over to adapt and work reasonably well?

- How would the exploration and exploitation schemes, i.e the RL method driven by a greedy policy compare to an RL method based on Thompson sampling?
4 Methodology

This chapter has been organized into three important sections. The first section will introduce the detailed setup of the cat-and-mouse MDP in a simulated world. In the second section we will detail the model of the robots. The final section introduces the RL algorithms that have been applied to run the experiments for the cat-and-mouse game.

4.1 Modeling the simulated world

A simulated world is a distillation of the real world problem. In order to design a simulated world the thesis makes use of pycolab [23], a highly-customizable gridworld game engine that is used to test reinforcement learning algorithms. Let us begin by progressively designing and describing the elements that form the RL system as described in the problem formulation section (2.2), where the MDP $M = \{S,A,P,R,T\}$.

Gridworld environment

The environment maintains the: states, actions, transition dynamics and reward functions. The decision maker or agent can then interact with these elements to find the strategy or policy that maximizes its cumulative reward. Thus, in the next subsections we can go over these environment-centric elements layer by layer:

4.1.1 States ($S$)

First, let us organize the states into a gridworld. A gridworld is a rectangular grid of size $m \times n$, where $m$ designates the number of rows and $n$ designates the number of columns of the grid. The Figures 4.1, 4.2 show one such grid space of size $6 \times 6$ that represents the state space $S$ of a simple finite MDP where $|S| = 36$. The cells of the grid corresponds to the states of the environment. The cells shaded in brown represent the borders or walls of the environment, to ensure the agent does not exit the simulator. The two redundant state mappings (Figures 4.1, 4.2), can be used interchangeably based on how we choose to access the states. Access to the states is enabled by a tuple $(\text{row}, \text{col})$ or by numbering the states over the $m \times n$ grid. The state converter function (I) reaffirms this relation:

$$state\_number = (m \times row) + col$$ (I)
4.1.2 Actions ($\mathcal{A}$)

Once the states are organized and accessible (Fig. 4.1, 4.2), the next step is to setup the actions the agent makes use of to traverse the grid cells. At each cell, the agent is provided four possible moves: north, south, east and west, that represent the direction the agent chooses to traverse from one grid cell to the next. This introduces the action space $\mathcal{A} = \{\text{north, south, east, west}\}$ for the agent, where the size of the action space $\mathcal{A}$, is $|\mathcal{A}| = 4$. The Figure 4.3 shows the deterministic transition dynamics encoded in the $6 \times 6$ grid environment to facilitate movement for the agent between the different states.
4.1.3 Transition dynamics ($P$)

The transition dynamics refers to the set of rules encoded within the grid world that can transition an agent based on its current state and choice of actions towards a new state in the grid world. Before setting up the transition probability matrix for the grid world environment, it is important to recognize two key variants: a deterministic environment and a stochastic environment. In a deterministic environment, the agent chooses an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, in-turn the environment transitions the agent to a new state $s' \in \mathcal{S}$ with absolute certainty. An example of such an environment can be observed in the game of Chess, where the deterministic nature of the board suggests the different transitions allowed for the different chess pieces. On the other hand a stochastic setting can be understood with the help of an environment influenced by wind. In such a setting, it might not always be the case that choosing an action will lead the agent to a guaranteed next state and will rely on other environmental factors, say the direction the wind is blowing. As a simplification, we will consider the "cat-and-mouse" environment with a sense of determinism at its core. The stochastic and time-varying elements are introduced when the two agents have been established over the grid world (Section 4.1.4).

**Degree of Freedom:**

The state-action (Fig. 4.3) space can be classified into three categories based on the degree(s) of freedom the agent in a state can exercise:

- The **obstacle states** represent the static states where an agent has no actions it can exercise and can be considered states void of actions. Hence, such states have a degree of freedom of value 0. In the $6 \times 6$ grid example, we have 20 such states in the form of walls that border the grid world (in Figure 4.3).

- The **obstacle-based states** represent the states where some of the actions chosen by the agent could transition the agent towards an obstacle. Hence, the degree of freedom can vary between 1, 2 and 3 based on where the state is positioned. For instance, corner states have a degree of freedom of 2, since two of the four possible actions lead to obstacle states. Side states have a degree of freedom 3, since only one of the four actions lead to an obstacle state. In the $6 \times 6$ grid example (Figure 4.3), we have 12 obstacle-based states, specifically 4 corner states and 8 side states that form the inner lining after the border states.

- The **obstacle-free states** represent the states where all of the agent’s actions are guaranteed to transition the agent to a "safe" state that is free of obstacles. Hence, the degree of freedom is maximum, i.e. 4. In the $6 \times 6$ grid example (Figure 4.3), we have 4 such states centrally located.
Building Transition Tables:

The environment provides a functionality by listening to the agent’s action and transfers the agent to a new state. This is recorded on the transitions table or transition probability matrix. A model-based RL system is one in which the transition probabilities and reward function are known. A model-free RL system is one where the transition probabilities are unknown. On learning to build the transitions from scratch, we can in principle extend this functionality onto environments where the transition probabilities are unknown. With the help of the degree of freedom idea, we can construct the transition dynamics of the system with determinism at its core. Table 4.1 shows the probability of transition from a state \( s \in S \) to \( s' \in S \) using action \( a \in A \) over the 6 \( \times \) 6 grid example. On initialization, we assume no transitions are possible and hence the probability values are set to 0.0.

**Obstacle Transition Example:** Let us observe an example of an obstacle state where the degree of freedom is 0. Since agent’s do not have any actions to exercise in these states, the probability value of transitioning from such a state remains unchanged from initialization.

<table>
<thead>
<tr>
<th>(state, action)</th>
<th>state = 0</th>
<th>state = 1</th>
<th>state = 2</th>
<th>...</th>
<th>state = 35</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0, a_u))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((0, a_d))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((0, a_r))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((0, a_l))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>((35, a_u))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((35, a_d))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((35, a_r))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
<tr>
<td>((35, a_l))</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.1: Initial transition dynamics \( P_{ss'}^a \) built for the 6 \( \times \) 6 grid world in which the \((state, action)\) tuple determines from what state which action where \( s, s' \in S \) and \( a \in A \). \( A = north, south, east, west \), represented by \( a_u, a_d, a_r, a_l \).

**Obstacle-free Transition Example:** Let us now analyze an example of an obstacle-free state transition. If an agent in such a state was to choose from the actions: north, south, east, west, the environment should then transition the agent to a new state based on the agent’s choice of action. Thus, this functionality the environment provides by listening to the agent’s action and transferring it to a new state is recorded on the transitions table or transition probability matrix. In the case of state = 14 (based on Fig.4.2), we can observe
in the Table 4.2 how the transition dynamics suggests the new states based on the actions selected. Similarly, such updates of the transition probabilities can be replicated across each of the obstacle-free states over the environment’s transition dynamics. (Table 4.1)

<table>
<thead>
<tr>
<th>(state, action)</th>
<th>state = 8</th>
<th>state = 13</th>
<th>state = 15</th>
<th>state = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>(14, north)</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(14, south)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>(14, east)</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(14, west)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.2: The transition probabilities of moving to the new states: 8, 13, 15, 20 (Fig. 4.2) when choosing actions: north, south, east, west respectively from state = 14

**Obstacle-based Transition Example:** Let us consider, a state where the agent encounters an obstacle. For example, in a corner state such as state = 25 (based on Fig. 4.2), we have obstacles (in the form of walls) when choosing the actions: south, west. Therefore, in Table 4.3, we observe how the agent remains in the same state with absolute certainty when choosing these actions. In the 6 × 6 grid considered, we have 4 such corner states and 8 side states. Similarly, such updates of the transition probabilities can be replicated based on each of the obstacle-based states, i.e. states whose action/actions lead to an obstacle.

<table>
<thead>
<tr>
<th>(State, Action)</th>
<th>State = 19</th>
<th>State = 25</th>
<th>State = 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>(25, north)</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(25, south)</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>(25, east)</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>(25, west)</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4.3: The transition probabilities of moving to the new states: 19, 26 (Fig. 4.2) when choosing actions: north, east respectively while remaining in the same state = 25, otherwise.

**Limitations**

The reason for making this distinction clear between the different state-action pairs is to facilitate further ideas into optimization techniques, safe exploration and dynamic generation of worlds. Specifically, when the robot seeks to explore unfamiliar environments that it has not been pre-trained over, it needs to build a layout of the world it is exploring and needs to do so safely. These definitions will hold for both agents in the simulator and the robots operating in the real world. A model-based system is one in which the transition probabilities and reward function are known. A model-free system is one where the transition probabilities are unknown. In this thesis, we learn to update

© Uppsala University

Suraj Murali
the transition dynamics with a sense of determinism at its core (Tables 4.1, 4.2, 4.3). We can expand this idea not only to initialize the layout of a known environment (such as the $6 \times 6$ grid), but also to craft the finer transitions from an evolving simulated and real experience based system.

### 4.1.4 T & J Agents

On establishing the state, actions and transition mapping in the previous subsections, we can now introduce the two agents that traverse the grid world. Fig. 4.4 introduces the TOM agent (represented by $T$) at position $(row = 1, col = 4)$ or state number 10, while the Fig. 4.5 introduces the JERRY agent (represented by $J$) at position $(row = 4, col = 1)$ or state number 25. It is important to note that these agents are not aware of each others presence. According to each agent, the other agent is a stochastic element that is embedded into the environment. The position of one agent is the source of reward for the other and should be predicted in order to act in the environment. The illustration in Figure 4.6 combines the agents over the same grid world, but keeping in mind that they are not aware of each others positions throughout the learning episode until they overlap with each other.

![Diagram of the grid world with TOM and JERRY agents](image)

**Figure 4.4:** The tom agent represented as $T$ is initialized at $(row = 1, col = 4)$ or state $= 10$

**Figure 4.5:** The jerry agent represented as $J$ is initialized at $(row = 4, col = 1)$ or state $= 25$

### 4.1.5 Reward shaping and termination condition ($R, T$)

Now that the agents have been established in the environment, the next task will be to shape their rewards. In this zero-sum game, the two agents TOM and JERRY take on the roles of hunter and prey. The basic reward function for the two agents exists in $[-1, 1]$. An episode can terminate in two ways, one where TOM runs out of time-steps to catch the prey and the other is if TOM catches JERRY. Alternatively, one where JERRY successfully evades TOM or the other where JERRY gets caught by TOM.
More precisely, from TOM’s perspective, encountering JERRY, i.e. overlapping on
the position JERRY will terminate the episode with TOM receiving a reward of +1 and
JERRY receiving a reward of −1. At the same time, if TOM does not successfully find
JERRY within the the maximum number of time steps allocated for an episode TOM will
be punished with a reward −1 while JERRY receives a reward of +1 for successfully
evading TOM. Thus, this setting allows both TOM and JERRY to predict each others
position and thereby learn strategies to outsmart one another through self-play.

Practical considerations:

Since, such a reward setup assumes that the two agents overlap over the same position
to terminate a learning episode, the case of the real robots presents safety problems such as possibly crashing, or falsely identifying the other robot to be an obstacle when
probing the environment. Hence, the agents in the simulated world should be further
improved to depict the real world robots which makes use of sound sensors (introduced in
the next section 4.2). This robotic setup facilitates a handshake signal between the robots.
In order to illustrate this idea, we can build a sensory field in the form of a neighborhood
around the two agents to depict the sound field that surrounds the real robots. The Figures
4.7, 4.8 depicts the area of influences that surrounds the two agents. Hence, when the
agents overlap with each other (shown in Figure 4.9), i.e. when the area of influence of
one agent overlap with the position of the other agent, the reward signal and termination
as presented in Section 4.1.5 is triggered.
This concludes the modeling of the "cat-and-mouse" problem in the simulated world and the setup of the different elements that form the MDP $M = \{S, A, P, R, T\}$. The final element $T$ represents the length of the episode through a sequence of time steps $t = \{1, \ldots, T\}$. Hence, this limits the amount of time steps for the finite horizon problem of Tom to find Jerry and likewise, Jerry to evade Tom. In Section 4.3, we will extend from this structuring to suit existing state-of-the-art MDP formulations by building on the foundations covered so far.

### 4.2 Modeling the Real World

On establishing the simulated agents that run the simulations, the next task is the design of these simulated agents as robots in the real world. In this thesis, we make use of LEGO MINDSTORMS EV3 robots [26] that provides the necessary building blocks to design the hunter and prey robots. The hardware and software setup are both described in a progressive fashion in the next subsection, titled the robotic environment.
The Robot Environment:

As the decision maker the agents in the simulated world take the form of robots in the real world. We aim to replicate the MDP structure that was used to design the simulator \(4.1\) to build the robotic setup in the real world. Given the egocentric nature of the agents, the different motors and sensors used are considered part of the environment, more specifically - the robot environment.

4.2.1 State to floor mapping

The first task is to establish the grid world perspective presented in Section \(4.1.1\). This design choice is restricted to a 2D rectangular grid and can be imagined as a top down perspective that is directly above the floor plan the robots operate over. In order to make this environment mapping, we use the simulator’s 2D discretization of the continuous state space, which can be considered a handcrafted smart discretization of the state-action space \(27\). We can then stretch this (Figure 4.3) rectangular grid fabric over the floor plan where the robots operate. Thus, by digressing from the earlier \(6 \times 6\) example to a more real \(m \times n\) grid, we can manually calibrate the state-action space over the floor plan that the robots operate in the real world.

4.2.2 Transition dynamics

With the understanding of how the state space is mapped on to the floor, the next idea is to map the action mapping in the context of the robot. The transition dynamics in the real world suggest how the action space is mapped onto the floor the robots operate over to transition the robot seamlessly from one floor state to another.

The movement protocol:

In order to ensure that the robots are capable of moving based on the actions, i.e. \(A = \{north, south, east, west\}\) we correspond this action space of the agent to the robotic protocols \(A = \{move\_forward(), move\_backward(), move\_right(), move\_left()\}\). We make use of two EV3 LARGE MOTORS (Figure 4.10) and an EV3 STEEL TECHNIC BALL PIVOT(Figure 4.11) for each robot to facilitate movement.
**Methodology**

Figure 4.10: Top down view of the basic frame.

**Forward and Backward:** Given the *speed* and *duration* values, the motors are capable of spinning the wheels *forward* with a positive power value and *backward* with a negative power value. Thus, with a well balanced and symmetric setup of the robot’s frame (Figure 4.10), the `move_forward()` and `move_backward()` movements are a simple protocol that turns the two motors or wheels with a positive and negative power value respectively, to duplicate the *north* and *south* actions in the simulator.

**Right and Left:** The `move_right()` and `move_left()` commands require a more elaborate structure. In the simulations, the *east* and *west* actions simply transfer the Tom and Jerry agents, from one grid cell to the next in the *east* and *west* directions. But when placing that in perspective for the custom built robot, we would need to maintain a navigational orientation that the robot can align itself to. Hence, we can now bring this idea of a navigational compass to life in the form of an EV3 Gyro Sensor (Figure 4.12) that keeps measure of the inertial angles. This sensor measures angles (in degrees) such that when spun in the clockwise direction its value spirals outwards from the initial 0° value, towards 1°, 2°, … Likewise, anti-clockwise rotations are measured with −1°, −2°, … values. In order to standardize the angle measured by the sensor, we can use the modulus operator (%) that flattens the angle measured over 360°. Thus, we can now set fixed navigational values for the direction *[North Angle, East Angle, South Angle, West Angle]* such that they represent the angles {0°, 90°, 180°, 270°} respectively.
In the simulated model discussed in Section 4.1.1, the T and J characters always maintain this orientation of facing north. Thus, we transfer this idea into the robotic setup that guarantees the robots TOM and JERRY will always face north after executing any action. The movement command move_right() will breakdown into a series of the protocols: turn_right(), move_forward() and then turn_left() to ensure the robot can smoothly transfer from one grid cell to the next. Similarly, move_left() will breakdown into: turn_left(), move_forward() and then turn_right().

**Turn command:** A turn is accomplished by the act of spinning one of the wheels and pivoting over the other. Thus, turn_right() is performed by rotating the left wheel forward until the robot achieves a right angled (90°) turn. The EV3 Gyro Sensor (Figure 4.12) tracks the inertial angles as the robot turns a singular wheel to accomplish the turn. When an angle value of 90° is provided to the turn() function, this enables to set an offset value from the current angle measure to induce an error for the robot to minimize. Providing negative angles in the turn() command would rotate the appropriate wheel in the opposite direction. So, in order to maintain that the robot faces north - the move_right() action would break down into: turn_right(90°), move_forward(), turn_left(−90°). In addition, we need to ensure this turn is precise overtime because minor angular errors could result in the robot losing track of its true orientation. Also, it is possible the robots operate across floors of varying friction with the wheels. In order to solve such internal and external factors that builds angular errors, the robotic environment makes use of a Proportional-integral-derivative (PID) controller [28]. Specifically, only a simple P-controller is used to modulate the power value over the suitable motor, proportional to the angular error value it aims to minimize.
Similarly, the `turn_left(90°)`, `move_forward()`, `turn_right(−90°)` that represents the `move_left()` command is also equipped with the PID control logic. The PID control is also enabled over the `move_backward()` command to correct angular errors when moving forward and backward axis. It is also interesting to note that if the action space is broken into finer commands, for instance $A = \{north, northeast, east, south-east, south, south-west, west, north-west\}$ the offset angle will now be 45°. Such improvisations that are considered to "smartly" discretize over the continuous space would allow the robot more finer transitions but at the cost of increasing the overall complexity of the RL problem over the underlying states and transition dynamics. But we limit ourselves to the simple $A = \{north, south, east, west\}$ setting, given that we are interested in the simplest case that replicates the actions defined as per the simulator.

### 4.2.3 Avoiding obstacles and reward handshake

On establishing the basic movement mapping, the next task is to handle the cases where the robot should learn to avoid crashing into obstacles. It is also important that the two robots are able to recognize each others presence when they are close to one another.

**Detecting obstacles:** Each robot is equipped with two EV3 ULTRASONIC SENSORS (Figure 4.13) that measure the distance (in meters) of an obstacle in the direction the sensor is placed. Hence, for the TOM and JERRY robots - these two sensors are placed facing the north and south directions and uses the `update_ultrasonic_north()` and `update_ultrasonic_south()` respectively to probe the distance values in the directions mentioned. Thus, by obtaining the distance measure after each action the robot performs, we are able to sample discrete distance measures of the continuous space. In the case where obstacles are present in the `east` and `west` directions, the robot makes
use of the `turn_left()` and `turn_right()` commands to face the ultrasonic, specifically the `update_ultrasonic_north()`, in the direction the robot is interested in exploring. By using a threshold on these distance values, the robot can be made aware of obstacles that are close at run-time and receives an immediate negative reward of -1, to punish actions from states that are close to obstacles. At the same time the robot associates a negative reward for an action from a state, it also learns to evolve the transition matrix of the environment it currently is operating in. While this punishment is updated based on the basis of the ultrasonic distance thresholding, this does not qualify to terminate the episode since the objective is to hunt/evade the other robot. This building of the negative reward is made dynamic because the robots can be made to operate in varying environment layouts with possibly multiple static obstacles it must learn to avoid colliding into. In such an obstacle-based state (Section 4.1.3), the transition dynamics can also be built dynamically to help the robot associate the state-actions leading to such obstacle states and hence retain the robot in the same state. This architecture, in theory should allow the robots to operate over any $m \times n$ grid, where $m$ is the number of rows and $n$ is the number of columns, to build an evolving model of the world as the robots operate within them.

![Side view on attaching the ultrasonic sensors.](image)

Figure 4.13: Side view on attaching the ultrasonic sensors.
Handshake: Once the robots can safely manoeuvre through the state-action map, the next task is to shape the reward function, \( R \). In order that the two robots are able to communicate with each other, we make use of the LEGO NXT SOUND SENSOR (Figure 4.14). This sensor is capable of measuring sound pressure in decibels\( (\text{dB}) \). The sensitivity of the sensor can be adjusted and this can be intuitively thought of as a mechanism to threshold the area of influence around the robots current state position. Thus, by providing the robot with this \textit{listen()} capability, the sensor is able to pick up the motor noise from the other robot. In a scenario where the robots are close to each others area of influence (Figure 4.9), the first robot that listens to the other robots movement signals a short alarm signaling its presence and termination of its learning episode. As mentioned earlier in Section 4.1.5, if the TOM robot detects the JERRY robot, it receives a positive reward of +1 for catching the prey, while JERRY receives a negative reward of -1 for having been captured by the hunter. If the robot runs out of time i.e. reaches the maximum limit of time steps for the episode, the TOM robot receives a negative reward of -1 for not capturing the prey within the steps allocated and the JERRY robot receives a positive reward of +1 for successfully evading the hunter.

4.3 RL algorithms

In the earlier two sections, we were able to model the “cat-and-mouse” game with the agents that operate over simulations and the robots that operate in the real world. Now
that we have all the elements that form the \textit{RL} system, in this section we will detail the algorithm that stitches the system together so that we can run the necessary experiments.

Q-learning

Q-learning \cite{29} is an online reinforcement learning algorithm that only uses the last experience to update its policy. With the use of the value functions discussed in Chapter 2.3.2 we are able to update the state-value function and action-value function (\textit{Q-value}) based on the policy chosen by the agent. The policy is a mapping from states to actions in order to estimate how "good" it is for the learning agent to perform a given action at any given time. With the building and updating of this \textit{Q-table}, the agent is able to at any given time estimate the best strategy to follow to either exploit the knowledge gained or to choose to explore the environment. The \textit{Q-tables} are model-free since they are an internal representation of the environment that the agent makes use of for solving the problem.

4.3.1 Problem formulation for T & J

We make use of existing literature (notably Osband \cite{22}, Russo \cite{30}) towards extending the problem formulation (discussed earlier in 2.2) in order to accommodate tabular solution methods for studying the hunter and prey robots. Let us consider the two tabula rasa learners (Tom and Jerry) that has no prior knowledge of the environment it operates within. On considering a finite horizon \textit{MDP} $M = \{S, A, P, R, H, \rho\}$, where $S$ is the state space, $A$ is the action space, $H$ is the horizon, and $\rho$ is the initial distribution. An agent begins in a state $s_0$ and over each timestep $h = \{1, 2, \ldots H\}$ the agent selects action $a_h \in A$, receives a reward $r_h \sim R^M_{s_h, a_h}$, and transitions to a new state $s_{h+1} \sim P^M_{s_h, a_h}$. Here, $R^M_{s_h, a_h}$ and $P^M_{s_h, a_h}$ are probability distributions that represent the prior beliefs that the agent build on. A policy $\mu$ is a function mapping each state $s \in S$ and timestep $h = \{1, 2, \ldots H\}$ to an action $a \in A$. The value function $V^M_{\mu, h}(s) = E\left[ \sum_{j=h}^{H} r_j( s_j, \mu(s_j)) \right \mid s_h = s]$ encodes the expected reward accumulated under $\mu$ over the remaining episodes when starting from state $s$ and timestep $h$. We define the \textit{MDP} and all other random variables considered with respect to a probability space $(\Omega, F, P)$.

The TOM and JERRY agents are considered as an episodic \textit{RL} problem, the agents learn about $R^M$ and $P^M$ over episodes of interaction with the \textit{MDP}. It is also important to note (as depicted in Figures 4.4, 4.5), that $R^M$ and $P^M$ exist for each of the agents independently as they interact with the environment from their unique starting locations. As discussed in the agent-environment interface (Section 2.1), we consider the trajectory $H = (s_1, a_1, r_1, \ldots, s_{t-1}, a_{t-1}, r_{t-1})$ to denote the history of observations made prior to time $t$. To highlight this time evolution within the episodes we will index $s_t = s_{kh}$ for the state at timestep $t$ which is $h$ steps into the $k^{th}$ episode or $t = (k-1)H + h$ and similarly for $H_{kh}$. Since the two agents rely on each other to shape their reward function
through interactions with the environment, we can express the problem of maximizing the cumulative reward \( \sum_{k=1}^{K} \sum_{h=1}^{H} r(s_{kh}, a_{kh}) \) in a finite horizon MDP over periods \( k = \{1, 2, \ldots, K\} \), each involving the selection of a policy \( \mu_k \) for use over an episode of interaction between the agents and the MDP. We consider the underlying environment-centric elements of the MDP as defined in Section 4.1, such that the transition dynamics (Section 4.1.3) and reward functions (4.1.5) are both known. In the next subsection we will consider two variants of how policies are selected.

I) Greedy Reinforcement Learning

The most common approach in RL is to treat estimation and control problem separately. This algorithm hence approximates the true underlying MDP \( M^* \), which is unknown, by its best point estimate \( \hat{M}_k \) and follows the policy that is optimal for the estimate. This purely exploitative policy is called greedy, since it does not account for the potential future value of exploratory actions [22]. This algorithm works reasonably well and has been the classical approach for the early success in reinforcement learning [9]. While the greedy strategy is effective in solving a single known MDP, they might have a number of drawbacks given the purely exploitative greedy agent might never learn the optimal policy and can converge to a local optimum solution, whereas the goal is to find the global optimum strategy.

Algorithm 1: Greedy reinforcement learning [22]

1: **Input**: Point estimator for MDPs \( \phi \)
2: **for** episode \( k = 1, 2, \ldots \) **do**
3:   Estimate MDP \( \hat{M}_k = \phi(\mathcal{H}_{k1}) \)
4:   compute \( \mu_k \in \text{argmax}_{\mu'} (\mathbb{V}_{\hat{M}_k}^{\mu,1}) \)
5:   **for** time \( h = 1, 2, \ldots, H \) **do**
6:     take action \( a_{kh} = \mu_k(s_{kh}, h) \)
7:     observe \( r_{kh} \) and \( s_{kh+1} \)
8:     update \( \mathcal{H}_{kh} = \mathcal{H}_{kh} \cup (a_{kh}, r_{kh}, s_{kh+1}) \)
9: **end**
10: **end**

The \( \epsilon \)-greedy policy selection: One of the most common ways to deal with a greedy exploitative agent is to introduce the concept of dithering. Since the exploitative agent do not account for exploratory actions, dithering strategies purposefully inject some random noise during the choice of actions so that the agent is not purely greedy. To put this into perspective the thesis adopts the most common dithering strategy, i.e. \( \epsilon \)-greedy policy selection. Thus, Algorithm 2 differs from Algorithm 1 in the way actions are selected. Thus by varying the \( \epsilon \) value, it is possible to contrast a purely exploitative strategy to a purely exploratory strategy. On one hand, when the \( \epsilon \) value tends to 1, i.e
$\epsilon = 1$, it can be understood as a case of a random walk, in which the selection of actions is independent of the Q-values. On the other hand as $\epsilon$ approaches the value of 0, the greedier it is in its choice of action, making it more likely to choose actions that exploit reward based on the state and timestep it is on.

Algorithm 2: RL with $\epsilon$-greedy exploration \[22\]

1: **Input:** Point estimator for MDPs $\phi$
2: **for** episode $k = 1, 2, ..$ **do**
3:   Estimate MDP $\hat{M}_k = \phi(H_{k1})$
4:   compute $\mu_k \in \arg\max_{\mu'}(V_{\mu,1}^{\hat{M}_k})$
5: **for** time $h = 1, 2, .. H$ **do**
6:   sample $w_{kh} \sim \text{Unif}([0,1])$
7:   if $w_{kh} \leq \epsilon$ **then**
8:     take action $a_{kh} \sim \text{Unif}(A)$
8: **else**
9:     take action $a_{kh} = \mu_k(s_{kh}, h)$
10: **end**
11: observe $r_{kh}$ and $s_{kh+1}$
12: update $H_{kh} = H_{kh} \cup (a_{kh}, r_{kh}, s_{kh+1})$
13: **end**
14: **end**

II) Posterior Sampling Reinforcement Learning (PSRL):

This approach was introduced by Ian Osband \[9\] that guarantees to be more statistically efficient, in the way the agent explores its environment, and computationally efficient as it requires exactly the same computation as solving a single known MDP \[22\]. In this technique, PSRL takes a single sample from the posterior and follows the policy that is optimal for the duration of the episode. In Algorithm 3 we observe that at every episode, Thompson sampling takes a single sample from its posterior for the environment and then at every timestep, follows the policy that is optimal for that sample. Thompson sampling hence directs its exploration efficiently towards policies that are either informative or with a high posterior mean.
Algorithm 3: Posterior sampling for RL\textsuperscript{[22]}

1: **Input**: Point estimator for MDPs $\phi$
2: **for** episode $k = 1, 2, ..$ **do**
3: \hspace{1em} sample MDP $M_k = \phi(\cdot | H_{k1})$
4: \hspace{1em} compute $\mu_k \in \arg\max_{\mu'} (V_{\mu,h}^{M_k})$
5: \hspace{1em} **for** time $h = 1, 2, .. H$ **do**
6: \hspace{2em} take action $a_{kh} = \mu_k(s_{kh}, h)$
7: \hspace{2em} observe $r_{kh}$ and $s_{kh+1}$
8: \hspace{2em} update $H_{kh} = H_{kh} \cup (a_{kh}, r_{kh}, s_{kh+1})$
9: \hspace{1em} **end**
10: **end**
5 Results

In this chapter we will experiment with the algorithms discussed in the earlier chapter towards obtaining results that test the TOM and JERRY decision-makers. In the first section we will discuss the experiments conducted over the TOM & JERRY robots and later discuss the experiments run on the simulator.

5.1 Experiments in Real World

The experiments with robots have been broken down into two stages. The first stage conducts a small experiment with only one robot (the Tom robot) over a small environment to ensure that the robot behaves as intended. As most of the results are obtained in the form of tabular solutions, this section explains how one can make sense of these tables. In the second experiment, we introduce both the robots into an environment and study how an episode can be understood.

5.1.1 Stage 1: Setting up the Tom robot

In this experiment, the finite MDP is defined as:

- State space: \(|S| = 25\), on considering a 5 × 5 grid world.

Figure 5.1: Tom robot placed at state = 8 in a 5 × 5 grid world with actions = \{north, south, west, east\}. \(F_d\) represents the forward distance and \(B_d\) represents the backward distance as probed by the ultrasonic sensors.

© Uppsala University

Suraj Murali
• Action space: $|A| = 4$, where the actions are \{ north, south, west, east \} which breakdown into the robotic protocols - \{move_forward, move_backward, move_left, move_right\}.

• Length of episodes: $h = \{ 0, 1, \ldots H \}$, where $H = 20$.

• Number of episodes: $episodes = \{ 0, 1, \ldots maxEp \}$, where $maxEp = 10$.

• The epsilon greedy learning agent with $\epsilon = 0.1$ is used.

• The ultrasonic sensors are used to detect obstacles (Section 4.2.3). A distance threshold is set over the north and south ultrasonic sensors, i.e. forward distance $Fd = 0.45$ meters and backward distance $Bd = 0.27$ meters. Thus, on probing the environment at every timestep, the robot is able to detect obstacles when the distances are below these thresholds. In order to avoid choosing actions that might steer the robot to collide, a dynamic negative reward is built to ensure the robot refrains from choosing such actions from such obstacle-based states (Section 4.1.3).

• The rewarding state is encoded into the environment at $state = 16$, as shown in the illustration Figure 5.1.

In order to ensure that the robot stays within the confines of the grid world with or without the presence of physical obstacles a safety feature is introduced to surround the state-space by impassable states surrounding the grid world.

**How to interpret the qValue and qMax tables?** When the robot or agent begins its learning episodes, it learns the $q$-values over discrete states, actions and timesteps. These $q$-values tables are stored over the $qValue$ table and the $qMax$ tables. On first building the intuition with the stationary reward the robot is seeking to discover, we can study the trajectory that the agent learns to follow to maximize the reward function. On understanding this simple task, it will serve to build the intuition towards the later experiments conducted in the latter sections using both the robots and the simulator.

**Stage 1: qValue table**

The qValue table can be studied as an evolution of the $q$Values across the actions space, over the different states through time. The initialization of this table is presented in Table 5.1 based on Figure 5.1.
The states we are interested in studying revolves around the obstacle-free and obstacle-based states (Section 4.1.3). Hence, on running the greedy learner based on the setup discussed above (5.1.1) the Table 5.2 illustrates what has been learned.
Table 5.2: Initializing the qValue table for the Tom robot.

<table>
<thead>
<tr>
<th>State, t</th>
<th>$a_{↑}$</th>
<th>$a_{↓}$</th>
<th>$a_{←}$</th>
<th>$a_{→}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(6, 1)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(6, 2)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(6, 3)</td>
<td>-0.95</td>
<td>0.99</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>(6, 4)</td>
<td>-0.004</td>
<td>1.99</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>(6, 5)</td>
<td>0.99</td>
<td>2.99</td>
<td>1.05</td>
<td>1.05</td>
</tr>
<tr>
<td>State 7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(7, 1)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(7, 2)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(7, 3)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.004</td>
<td>0.0</td>
</tr>
<tr>
<td>(7, 4)</td>
<td>-0.95</td>
<td>0.99</td>
<td>0.99</td>
<td>0.00</td>
</tr>
<tr>
<td>(7, 5)</td>
<td>-0.002</td>
<td>1.99</td>
<td>1.99</td>
<td>0.04</td>
</tr>
<tr>
<td>State 8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(8, 1)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(8, 2)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(8, 3)</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(8, 4)</td>
<td>-0.99</td>
<td>0.004</td>
<td>0.004</td>
<td>-0.99</td>
</tr>
<tr>
<td>(8, 5)</td>
<td>-0.95</td>
<td>0.99</td>
<td>0.99</td>
<td>-0.95</td>
</tr>
<tr>
<td>State 11</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(11, 1)</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(11, 2)</td>
<td>-0.99</td>
<td>0.99</td>
<td>-0.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(11, 3)</td>
<td>0.04</td>
<td>1.99</td>
<td>-0.004</td>
<td>0.0</td>
</tr>
<tr>
<td>(11, 4)</td>
<td>0.99</td>
<td>2.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>(11, 5)</td>
<td>1.99</td>
<td>3.99</td>
<td>1.98</td>
<td>1.99</td>
</tr>
<tr>
<td>State 12</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(12, 1)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(12, 2)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(12, 3)</td>
<td>0.0</td>
<td>0.99</td>
<td>0.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(12, 4)</td>
<td>0.04</td>
<td>1.99</td>
<td>1.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(12, 5)</td>
<td>0.99</td>
<td>2.99</td>
<td>2.99</td>
<td>0.99</td>
</tr>
<tr>
<td>State 13</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(13, 1)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(13, 2)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(13, 3)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(13, 4)</td>
<td>0.0</td>
<td>0.99</td>
<td>0.99</td>
<td>-0.99</td>
</tr>
<tr>
<td>(13, 5)</td>
<td>0.04</td>
<td>1.99</td>
<td>1.99</td>
<td>-0.0004</td>
</tr>
<tr>
<td>State 16</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(16, 1)</td>
<td>0.0</td>
<td>0.99</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(16, 2)</td>
<td>0.0</td>
<td>1.99</td>
<td>0.04</td>
<td>0.0</td>
</tr>
<tr>
<td>(16, 3)</td>
<td>0.99</td>
<td>2.99</td>
<td>0.18</td>
<td>0.99</td>
</tr>
<tr>
<td>(16, 4)</td>
<td>1.99</td>
<td>3.99</td>
<td>0.52</td>
<td>1.99</td>
</tr>
<tr>
<td>(16, 5)</td>
<td>2.98</td>
<td>4.99</td>
<td>1.05</td>
<td>2.99</td>
</tr>
<tr>
<td>State 17</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(17, 1)</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(17, 2)</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(17, 3)</td>
<td>0.0</td>
<td>-0.001</td>
<td>1.99</td>
<td>0.0</td>
</tr>
<tr>
<td>(17, 4)</td>
<td>0.99</td>
<td>0.99</td>
<td>2.99</td>
<td>0.98</td>
</tr>
<tr>
<td>(17, 5)</td>
<td>1.99</td>
<td>1.99</td>
<td>3.99</td>
<td>1.99</td>
</tr>
<tr>
<td>State 18</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(18, 1)</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(18, 2)</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.0</td>
<td>-0.99</td>
</tr>
<tr>
<td>(18, 3)</td>
<td>0.0</td>
<td>-0.99</td>
<td>0.99</td>
<td>-0.99</td>
</tr>
<tr>
<td>(18, 4)</td>
<td>0.0</td>
<td>-0.0007</td>
<td>1.99</td>
<td>-0.009</td>
</tr>
<tr>
<td>(18, 5)</td>
<td>0.99</td>
<td>0.99</td>
<td>2.99</td>
<td>0.99</td>
</tr>
<tr>
<td>State 19</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(19, 1)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(19, 2)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(19, 3)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(19, 4)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(19, 5)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>State 20</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(20, 1)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(20, 2)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(20, 3)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(20, 4)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>(20, 5)</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Note: The table above shows the initialization of the qValue table for the Tom robot.
Stage 1: $q_{Max}$ table results

<table>
<thead>
<tr>
<th>Timestep</th>
<th>state = 0</th>
<th>state = 1</th>
<th>state = 16</th>
<th>state = 17</th>
<th>state = 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 5.3: Initial $q_{Max}$ table at $episode = 0$, i.e before learning the $q$-values across the different states across timesteps over the finite learning episodes.

<table>
<thead>
<tr>
<th>Timestep</th>
<th>$s_6$</th>
<th>$s_7$</th>
<th>$s_8$</th>
<th>$s_{11}$</th>
<th>$s_{12}$</th>
<th>$s_{13}$</th>
<th>$s_{16}$</th>
<th>$s_{17}$</th>
<th>$s_{18}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.99</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.99</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.99</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.99</td>
<td>0.0</td>
<td>0.0</td>
<td>2.99</td>
<td>1.99</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>1.99</td>
<td>0.99</td>
<td>0.04</td>
<td>2.9</td>
<td>1.9</td>
<td>0.99</td>
<td>3.99</td>
<td>2.99</td>
<td>1.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: $q_{Max}$ table for the obstacle-based and obstacle free states at the final episode, i.e. $episode = 10$. The values on the obstacle states has not been shown since they are influenced by the negative reward.

5.1.2 Stage 2: Setting up the Tom and Jerry robots

On interpreting the working of the TOM robot in the previous subsection, we can now introduce a similar setup for the JERRY robot and enable the handshake (4.2.3) as the source of reward for the two robots. For this experiment, we will consider the finite MDP as follows:

- State space: $|S| = 49$, on considering a $7 \times 7$ grid world. The Tom robot is initialized on $state = 12$ and Jerry robot is initialized on $state = 36$ as shown in Figure 5.2.
- Action space: $|A| = 4$,
• Length of episodes: \( H = 20 \),

• Number of episodes: \( maxEp = 10 \),

• Epsilon greedy agent with \( \epsilon = 0.1 \),

• Use the sound sensor as a source of reward.

On assuming an environment which is free of obstacles and noise, we can setup the two robots on the floor of the environment. In order to not manually reset the experiment each time an episode concludes an automatic restart feature is used in order that the robots move back to their initial positions before starting a fresh episode of learning.

![Figure 5.2: Tom and Jerry robots placed at state = 12 and state = 36 in a 7 x 7 grid world with actions = {north, south, west, east}.](image)

In order to facilitate the understanding of how an episode unfolds, this video [Youtube Link] provides an episode of how the Tom and Jerry robots work on the MDP based on the setup.

### 5.2 Experiments in Simulated World

In the earlier section we have observed how we setup the robots so that they can learn and interact with each other. This process of allowing the robots to learn in the real world is a time consuming activity which requires constant supervision to ensure each episode works as intended. Since it may take many thousand episodes before one can obtain results that show some characteristic behavior of the hunter or the prey, it is expected during the first thousand episodes during RL that the robots behave randomly, given they are learning to balance their exploration and exploitation boundaries. Hence in order to speed up the learning process, this thesis makes use of the simulations to
present some interesting behaviours observed with the TOM and JERRY on using the RL algorithms discussed in the earlier chapter (Section 4.3).

5.2.1 Setting up the simulator

In order to run experiments, we need to ensure we have the different parameters fixed for the finite MDP. In the experiment carried out we have:

- **State space:** $|S| = 100$, on considering a $10 \times 10$ grid board. The initial positions of Tom and Jerry agents are at opposite ends of the game board (as illustrated in Figure 5.2).

- **Action Space:** $|A| = 5$, apart from the actions discussed in Sec 4.1.2 we extend an option for the agent to *stay* in the same state. This extension is based on an observation made using the visualization provided in the Discussion and Analysis chapter (6.2.1). Thus, the action space is now $A = \{\text{north, south, east, west, stay}\}$.

- **Length of episodes:** $h = \{1, 2, \ldots, H\}$, where $H = 56$. Details regarding setting this horizon is elaborated in the Appendix A.1.1.

- **Number of episodes:** $\text{episodes} = \{1, 2, \ldots, \text{maxEp}\}$, where $\text{maxEp} = 10,000$.

- **The different variants of learning agents considered are:** Epsilon Greedy agents with $\epsilon = \{1, 0.5, 0.1\}$ and PSRL agents.

In the simulated world, it is possible to synchronize the movements of the Tom and Jerry agents by first receiving the agents choice of action before simultaneously updating them onto the environment so as to nullify any advantage in being the first to obtain an updated observation of the environment. This is however harder to presume with the real robots since different actions take varying times to execute. For instance, the action of moving *forward* or *backward* takes $\sim 1$ second time to execute whereas moving *right* or *left* takes $\sim 4$ seconds time to execute.

5.2.2 Experimental Results

On running experiments based on the above setup, we introduce a new evaluation metric, called *win ratio*. The win ratio for Tom ($T_{\text{Ratio}}$), calculates the number of episodes Tom successfully captures Jerry over the total number of episodes played. On running various Tom and Jerry agents against each other, the Table 5.5 presents the different win ratios and time taken for each of these experiments. Similarly, Table 5.6 presents the win ratio of Jerry ($J_{\text{Ratio}}$), calculated as the number of episodes Jerry evades Tom over the total number of episodes played.
### Table 5.5: Tom win ratio and time taken when simulated over 10,000 episodes.

<table>
<thead>
<tr>
<th></th>
<th>T_Ratio</th>
<th>Time(s)</th>
<th>T_Ratio</th>
<th>Time(s)</th>
<th>T_Ratio</th>
<th>Time(s)</th>
<th>T_Ratio</th>
<th>Time(s)</th>
<th>T_Ratio</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>ε = 1.0</td>
<td>0.5</td>
<td>5330.37</td>
<td>0.52</td>
<td>4528.3</td>
<td>0.51</td>
<td>4474.64</td>
<td>0.55</td>
<td>3990.79</td>
<td></td>
</tr>
<tr>
<td>Tom</td>
<td>ε = 0.5</td>
<td>0.95</td>
<td>3500.58</td>
<td>0.68</td>
<td>4354.19</td>
<td>0.73</td>
<td>4149.32</td>
<td>0.69</td>
<td>5123.32</td>
<td></td>
</tr>
<tr>
<td>Tom</td>
<td>ε = 0.1</td>
<td>0.97</td>
<td>4485.11</td>
<td>0.63</td>
<td>6017.77</td>
<td>0.61</td>
<td>4402.35</td>
<td>0.55</td>
<td>4129.96</td>
<td></td>
</tr>
<tr>
<td>Tom</td>
<td>PSRL</td>
<td>0.85</td>
<td>3307.7</td>
<td>0.67</td>
<td>4719.29</td>
<td>0.63</td>
<td>4437.42</td>
<td>0.32</td>
<td>4509.17</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5.6: Jerry win ratio and time taken when simulated over 10,000 episodes.

<table>
<thead>
<tr>
<th></th>
<th>J_Ratio</th>
<th>Time(s)</th>
<th>J_Ratio</th>
<th>Time(s)</th>
<th>J_Ratio</th>
<th>Time(s)</th>
<th>J_Ratio</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>ε = 1.0</td>
<td>0.5</td>
<td>5303.7</td>
<td>0.48</td>
<td>4528.3</td>
<td>0.49</td>
<td>4474.64</td>
<td>0.45</td>
</tr>
<tr>
<td>Tom</td>
<td>ε = 0.5</td>
<td>0.05</td>
<td>3500.58</td>
<td>0.32</td>
<td>4354.19</td>
<td>0.27</td>
<td>4149.32</td>
<td>0.31</td>
</tr>
<tr>
<td>Tom</td>
<td>ε = 0.1</td>
<td>0.33</td>
<td>4485.11</td>
<td>0.37</td>
<td>6017.77</td>
<td>0.39</td>
<td>4402.35</td>
<td>0.45</td>
</tr>
<tr>
<td>Tom</td>
<td>PSRL</td>
<td>0.15</td>
<td>3307.7</td>
<td>0.33</td>
<td>4719.29</td>
<td>0.37</td>
<td>4437.42</td>
<td>0.68</td>
</tr>
</tbody>
</table>
6 Discussion & Analysis

In this chapter, we will study the results produced in the previous chapter. In order to provide clean analysis of the tabular solutions we use visualizations to assist describe some of the evolving decision patterns observed from the hunter and prey entities.

6.1 Experiments in Real World

In Section 5.1 of the results chapter, the first stage of robotic experiment presents how the qValue and qMax tables (Tables 5.2, 5.4) show the trajectory converge to a shortest path as Tom discovers the stationary reward signal. On analyzing Table 5.2 we are able to study the trajectory chosen by the greedy policy:

State 8 $\rightarrow$ State 13 $\rightarrow$ State 18 $\rightarrow$ State 17 $\rightarrow$ State 16 $\rightarrow$ State 21

Thus, the robot after just 10 episodes is able to converge to an optimal path after avoiding the walls and learning to find the source of the reward signal on the 5 $\times$ 5 grid world. This small experiments establishes the proper working of the robotic hardware as commanded by the RL algorithm with a greedy policy.

6.2 Experiments in Simulator

In this section we will discuss some of the results obtained while running the experiments over the simulator. We will run the algorithms, and use heat plots to help compare and contrast the experiments carried out.

Heat plot: The heat plots are mapped over a 10 $\times$ 10 grid where each cell represents each state on the grid world. As shown in Figure 6.1 we initialize Jerry at grid position (Row 8, Col 11) and Tom at Grid position (Row 1, Col 8). While this follows the previous convention followed when designing the MDP (Section 4.1), the plotting uses the grid origin (Row 0, Col 0) from the bottom right instead of the grid origin from the top left (Figure 4.1). The heat plots only preserve the final winning position where Tom and Jerry receive their rewards. Hence, a value on Jerry heat plot is updated only if Jerry escapes within the finite horizon of the episode. Similarly, the value for Tom is updated when he catches Jerry from a particular position. The color map illustrates the z-value across the $(x, y)$ plot, where $x$ represents the row number and $y$ represents the column number on the grid world.

© Uppsala University

Suraj Murali
6.2.1 Analysis

On plotting the final winning position for either the Tom or Jerry agent through the episodes, we use a high-level abstraction to plot the value of certain positions from either agent’s perspective. With the help of these evolving heat map visualization we seek to observe some form of a pattern that can highlight the characteristic of the algorithms guiding the agent’s policy. Let us study them case by case to cleanly analyze the algorithms in action:

Case 1: Random walking agents

Figure 6.1: Random Walking Jerry(left) and Tom(right) over $10 \times 10$ grid with the action space \{north, south, east, west\}.

- The series of experiments with random walking agents aims to establish a baseline for the finite horizon value over the $10 \times 10$ grid considered. This number for a $10 \times 10$ grid is analytically found to be 56 based on the intersection times (Appendix A.1.1).

- When the two agents perform a random walk i.e. pure exploration with no exploitation of knowledge, we can observe(Figure 6.1) how the values scatter across the grid space. Evidently, the values do not enter the impassable obstacle states.

- In particular, the values for Tom remain more centralized suggesting that the intersection for Tom with Jerry is more likely around the center of the plot given the agents are constrained within a 2D grid space. Hence, for a random walking Jerry he is more likely to find his escape strategy along the corners of the plot.

- From the Tables 5.5 and 5.6 in the Experimental Results section (5.2.2), we are able to observe that the ratio of wins i.e. the wins for Tom across the total episodes and the wins for Jerry across the total episodes is maintained at 0.5 across 10,000 episodes, suggesting an even spread of wins for Tom and Jerry when $H = 56$.  

© Uppsala University Suraj Murali
Case 2: Greedy agents

Figure 6.2: Greedy Jerry(left) and Greedy Tom(right) over $10 \times 10$ grid with the action space \{north, south, east, west\}. The exploration-exploitation parameter $\epsilon = 0.1$.

- The experiments conducted over the greedy agents shows the expected evolution of strategies as the greedy agents exploit their knowledge about the other agent. The claim of an evolving strategy is backed by the extended experiment presented in the Appendix (Figure A.5).

- In order to analyze the strategy, let us discuss how the heat plot in Figure 6.2 above can be analyzed. We observe the first highly valued position for Jerry to be at \((\text{Row 7, Col 6})\) and can be considered a hiding spot of the Jerry agent. We observe how this hiding position for Jerry is revealed by Tom as the value over position \((\text{Row 6, Col 6})\) increases.

- On extending this idea across the plot presented in Appendix (Figure A.5), we see the new position for Jerry moves to position \((\text{Row 3, Col 2})\) until episode 4000 and is again revealed by Tom at position \((\text{Row 3, Col 3})\). Around episode 6000, Jerry shifts its hiding position to \((\text{Row 8, Col 2})\), again revealed by Tom from position \((\text{Row 8, Col 3})\) and so on.

- Figure 6.2 also helps use observe how very little Jerry and Tom explore their environment when compared to Figure 6.1 given they are exploiting their policy greedily.

- From the Tables 5.5 and 5.6 in the Experimental Results section (5.2.2), we are able to observe that the ratio of wins favours Tom with a value of 0.61 when played across 10,000 episodes for the case of the greedy agents.

Viewing the policy: In this clip we can observe: Episode 2000 - Jerry closely sticking to the walls of the simulator. Around Episode 3000 - Tom finds Jerry’s strategy and terminates the episode quickly. Around episode 5000: Tom works out a new strategy to find Jerry’s circling strategy.
Case 3: PSRL agents

Figure 6.3: PSRL Jerry and PSRL Tom over $10 \times 10$ grid with the action space \{north, south, east, west\}.

- The experiments conducted over the PSRL agents show a much more contrasting strategy when compared to the results observed previously with the random walking and $\epsilon$-greedy approaches.

- Firstly, we can observe a systematic balance between exploration and exploitation as illustrated by Jerry’s heat plot (Figure 6.3) when compared to Figure 6.2.

- From Jerry’s perspective, the winning strategy is to exploit staying put on its initial position. Tom’s heat map on the other hand, shows a strategy along the diagonal of the grid to maximize its chances of catching the prey.

- We can also observe some holes in the Jerry’s heat plot, i.e. (Row 7, Col 3), (Row 6, Col 3), (Row 4, Col 2), (Row 2, Col 2). . . The reason for this is linked to the length of the episodes being set to either an even number (i.e., the finite horizon value $H = 56$) or an odd number, thereby making certain states unreachable. In order to resolve this, we now take into account a new action stay that allows both the agents to choose to stay in particular position besides the actions that provide mobility \{north, south, east, west\}. In the plot below (Figure 6.4) we can already notice some improvement in how Jerry adapts to this change.

- The random sampling of the MDP incentives exploration, hence the agents use posterior sampling to build on their knowledge and have been observed to systematically explore and exploit its strategy.

- From Table 5.6 we are able observe the PSRL Jerry produces the highest win ratio of 0.68 for a Jerry agent when contested against the PSRL Tom strategy.
Case 4: **PSRL** Jerry vs Greedy Tom

This experiment provides some interesting results that conclude the analysis of the different combinations of algorithms tested.

- In Figure 6.5, we have the two winning strategies, i.e. the Epsilon greedy algorithm for the Tom agent and the PSRL algorithm for the Jerry agent play against each other.

- The PSRL Jerry agent behaves differently when put against the Greedy Tom agent and no longer exploits the strategy involving staying at its initial position as observed in Case 3 (Figures 6.3, 6.4).

- In the extended version of the plot (Figure A.7), we observe how the greedy Tom agent finds a patrolling circular strategy that it chooses to exploit after predicting the starting position of Jerry. The Jerry agent however anticipates this strategy of the hunter by opting to explore hiding spots outside this region and finds other venues for hiding.
• From the Tables 5.5 and 5.6 we study that the greedy strategy provides a win ratio of 0.55 towards Tom favour. But it is interesting to point that the PSRL Jerry is one of the algorithms for Jerry that works best with a win ratio of 0.45 over 10,000 learning episodes.

• In the illustration above in Figure 6.5 we can witness the strengths of both the algorithms when placed in their correct roles, i.e with relation to the hunter and the prey. A greedy hunter learns relatively quickly with an equally competent PSRL prey.

**Viewing the policy:** [Link] In this clip we present a sequence of games around episode 3000 to highlight Tom’s circling strategy. [Link] In this clip we observe around episode 6000 Jerry finding a hiding spot at Tom’s initial position.
7 Conclusion

In this chapter we will first present some of the answers to the research questions posed earlier and include directions the work can be extended for further development, before we conclude the thesis.

7.1 Research Answers and Future Work

In this section we discuss possible answers to the research questions (Section 3.3) that were posed during the start of the thesis before presenting some interesting venues to extend the project.

- Given the decisions made by the RL system directly affect the data acquired, how does one go about building such a system which actively engages with its datasets when compared to the fixed dataset approaches used in supervised and unsupervised learning?

  The key idea towards making quality decisions from the data lies with the design of the RL system. In this thesis, we have designed the cat-and-mouse problem from scratch and elaborated in-depth regarding the modelling techniques that influenced the decision-makers of the simulated and real world. On skillfully modelling the MDP to reflect on our problem statement, we were able to refine each of the elements of the MDP that impact the data acquisition process thereby gradually also improving the correctness of our initial problem statement. For example, when analyzing the data produced we observed a hole-like pattern (Figure 6.3) produced by Jerry. On interpreting the data we found the need to improve the action space ($A$) to include the stay action - that added an interesting spatial-temporal upgrade to the model.

- How does one interpret the solutions for this RL system?

  Given the highly time-varying problem this thesis aims to tackle, understanding the solution comes in two folds. In the first, we were able to highlight the value plot with the help of the heatmaps to understand where the Tom and Jerry decision-makers prioritized their strategy. Secondly, it is possible to observe the decision-making rule chosen at every timestep i.e. their policy on studying the policies followed at certain episodes using the simulator. On doing so, we were able to discover interesting strategies such as, the circular patrolling (Section 6.2.1) strategy for Tom, Jerry hiding at the hunter’s starting position (Section 6.2.1), to name a few. These strategies were motivated by the agents through self-play and with no prior knowledge, within reasonable clock times.
Can the decision-maker detect other time-varying entities in the model? If so, is it possible to highlight such evolving strategies?

In Figures (6.5, A.7) we are able to observe how both the greedy Tom and PSRL Jerry agents are able to detect each other's presence and plan strategies based on their algorithmic design. In the greedy agent setup (Figure A.5), we observe the classic scenario where there is a constant need to evolve strategy, i.e. the system does not reach a stable equilibrium as the hunter keeps anticipating the position of the prey.

Is it possible to transfer knowledge between a simulation of the task and the real world task? If so, can an architecture be developed to unify the gathered simulated experience and the real experience?

While this idea was inspired by the Dyna-Q architecture to integrate model-based and model-free RL, this was unfortunately not possible to fit due to lack of time. However, with the implementation details regarding the current robotic environment and the simulated environment - this project allows to experiment such possibilities to improve performance.

How can the decision-maker learn environments it has not previously trained over to adapt and work reasonably well?

The answer follows up from the previous question. The thesis only studied a simple setting where the obstacles were the walls that confined the agent/robot. However, the real world robot has been built to dynamically shape its reward and update its transition dynamics from a model-based setup towards a model-free setup where the transition dynamics and reward sources are unknown. Since we have also simplified the transitions to a simple deterministic setting by the discretization of the continuous space. Both the adaptation to a stochastic environment and extending to continuous state spaces are possible extensions.

How would the exploration and exploitation schemes, i.e. the RL method driven by a greedy policy compare to an RL method based on Thompson sampling in the cat-and-mouse game?

The different cases studied in the Analysis chapter has helped gain insight into how to compare and contrast the algorithms through a combination of "Tom and Jerry" tests with adversarial objective functions. The most interesting case that highlights the nature of the greedy strategy and Thompson sampling strategy can be observed in Figures (6.5, A.7). The illustrations bring to light the best of the greedy hunter and the smart prey when made to evolve off each other's behaviour through self-play.

7.2 Summary

On detailing the design and solution methods for the tabular problem, the project has successfully identified characteristic behaviour that can be associated to the hunter and prey decision-makers. The end-to-end system that has been built has multiple avenues for further research. Learning the building blocks for designing problems opens a
fresh perspective into how to engage actively with data in order to find scalable solutions for a wide range of fields. Thus, we have been able to find and generate non-stationary solution with some of the existing approaches and provided intuitive illustration to methods of RL to study highly-time varying models.
Literature


Appendices

A Results

A.1 Experiments in Simulated World

A.1.1 Setting up the simulator

Finding the horizon

We will be considering experiments in a $10 \times 10$ grid-board, ($|S| = 100$) where the agents have been initialized in opposite ends of the game board as shown in Figure 4.6. In order that we have a fair game, i.e. both Tom and Jerry have an equal chance of winning, we need to determine the finite horizon (i.e. the episode length) that both the Tom and Jerry agents use to play in the simulator.

Figure A.1: The intersection times observed when running the Tom and Jerry agents using random walk.
We can analytically find the episode length by running the simulations with the agents performing a purely exploratory strategy in the form of a random walk on the gridworld. As described in Section 4.3.1, the policy selection can be made to depict a random walk by setting the $\epsilon$ parameter for the greedy reinforcement learning to 1.0, i.e. $\epsilon = 1.0$. This implies that the agents will explore 100% of the time.

In Figure A.1, we run the simulations for the experiments over a 1000 episodes with an upper-bound on the step length set to 300 to find the exact timestep in the episode where Tom intersects with Jerry based on the termination condition mentioned in 4.1.5. The histogram in Figure A.2 provides the frequency of the episodes where a game terminates.

![Figure A.2: The histogram of the intersection times using random walk.](image)

On taking a mean of the different episode lengths over the episodes (as illustrated in Figure A.3), we are able to find the pivotal number of episodes that split the games evenly so as to allow both agents to figure out a strategy within the stipulated episode length value. This value for two random walking agents in a $10 \times 10$ board is at an episode length of 56. Once we are able to determine this value, we will now use this episode length over the $\epsilon$ greedy strategy, with the $\epsilon = 1.0$, episode_length = 56, over episodes = 10000 to observe how the strategies evolve.
Figure A.3: The mean intersection time found to be 56 in the 10 x 10 grid.

Tom and Jerry evolving strategy using random walk

In order to facilitate a visualization for the strategies executed, we make use of an evolving heatmap representation of the games. This heatmap takes into account the final winning position for the agent (Tom or Jerry) for a particular episode. In Figure A.4 we have 8 plots organized over the 10 x 10 board that the agents operate over. Each pair of plots (first Jerry and then Tom, from top to bottom) provide a snapshot of their respective Q-surface. The colour-scale across each plot gives a sense of how highly the agents value the states and also gives a sense of a scoreboard between the two agents. These heatmaps are generated over every 2500 episodes of length within the 10000 episodes the experiment is run over.
Figure A.4: Evolving Random Walk for Jerry and Tom over $10 \times 10$ grid with the action space \{north, south, east, west\}. 
Figure A.5: Evolving $\epsilon$ greedy algorithm for Jerry and Tom over a $10 \times 10$ grid with the action space \{north, south, east, west\}, where $\epsilon = 0.1$
Figure A.6: Evolving PSRL algorithm for Jerry and Tom over $10 \times 10$ grid with the action space \{north, south, east, west\}.
Figure A.7: Evolving PSRL algorithm for Jerry and Tom over $10 \times 10$ grid with the action space \{north, south, east, west\}.