AN ABSTRACT OF THE THESIS OF

Brett A. Valenti for the degree of Master of Science in Mechanical Engineering presented on March 12, 2010.

Title: Condensing Observation of Locale and Agents: A State Representation

Abstract approved: _____

Kagan Tumer

Computing agents require state information to make coherent and useful decisions. A state representation is a numerical translation of the environment and conditions that are pertinent factors in an agent's decision making. Although many representations, when paired with clever learning algorithms, are able to coordinate and capture prey in specific domain types, an inherent problem is that they rarely scale well with domain changes. This is due to much of their information being of a form that is dependent upon factors such as the size of the world and number of agents. When the state information is instead a scaled and condensed view of surrounding agents, mapped to an action list from which the agent is able to choose, the state is then in a form that is independent of both world size and number of agents. Coupled with a simple, 1-step Q-Learning algorithm, this representation proves to be quite robust, outperforming a hand-coded policy and two other state-space representations. Using several 1v1

simulations, we will show that a predatory agent is capable of tracking and capturing a moving target in a simple gridworld domain. It is also able to transfer its experience to new domains that have larger gridworlds, more dynamic environments, and even sensor noise and failure.

Condensing Observation of Locale and Agents: A State Representation

by

Brett A. Valenti

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 12, 2010
Commencement June 2010

Master of Science thesis of Brett A. Valenti presented on March 12, 2010.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Brett A. Valenti, Author

ACKNOWLEDGEMENTS

Had it not been for Dr. Nancy Squires, Dr. Brian Bay, and Dr. Kagan Tumer, I would not have pursued a graduate degree. Thank you all for your support, and not only for believing in me and my work, but telling me that you do. I would also like to thank Dr. Henri Jansen and the OSU Physics faculty for teaching me what it means to be intellectually humble and a strong thinker. Without your guidance, I would not be the person I am today, truly, thank you. Also, the OSU Mechanical Engineering department gave me the opportunity and the means to further my education, and for that I am forever grateful. Dr. Matt Knudson, whenever I've needed help or advice, you've responded quickly and selflessly, I cannot begin to tell you how much you've helped me, thank you. And a very, very special thanks to Dr. Tumer for everything he's done for me, despite what I've put him through as his grad student. He truly is a saint, and a fantastic professor/researcher/soccer player/chef/friend.

And to Jill, my love, to whom the most thanks is due, and she knows why.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Chapter 1 – Introduction

1.1 Autonomous Agents and Multiagent Systems

With the growth in complex technologies and the information and sensor networks of which they subsist, it has become increasingly difficult for humans to perform tasks that demand a high amount of memory, processing, and time. Tasks such as large-scale searches, data compilation, and monitoring for long, continuous durations are all functions an autonomous, computing agent could perform just as well as a human, and potentially much faster. The purpose of integrating autonomous agents into systems, no matter how complex or simple, is to relieve humans of repetitive and time-consuming duties so that their expertise and attention may be focused elsewhere. Already, autonomous agents aid in several applications such as email filtration, information routing, and visual effects allowing us many of the conveniences and pleasures we enjoy today[8, 29, 44].

There exist several research domains in which collectives of these autonomous agents, or Multiagent Systems (MAS), work in accomplishing tasks of even higher complexity. In the field of ground traffic, intelligent lane assignment and control may prove to alleviate congestion during rush hour times, allowing drivers to reach their destinations with little time lost [14, 5, 15]. Also, traffic in the air may benefit from the incorporation of a multiagent presence by aiding air traffic controllers

in monitoring sectors of high congestion; potentially saving the airline industry billions of dollars each year, and saving passengers hundreds of thousands of hours that were once lost to delay[46, 48]. Agents may also be given the capacity to move and alter their environment, as in many disciplines of robot coordination. In the case of robotic exploration, a team of robots may intelligently gather data in regions where it would be impossible or unsafe to send humans. Such situations may include search and rescue in hazardous environments, sample collecting in areas of high radioactivity, or even the exploration of extraterrestrial planets and moons. Several instances have shown that teams of robots can collectively observe, track, and find stationary and moving targets in simulated environments [18, 30, 47, 39, 42].

The field of robot coordination is a multi-faceted domain spanning the full breadth of research from theoretical simulations to real-life robotic soccer teams[24, 16, 40]. Although all coordination problems deal with creating coherent, adaptive actions within a collective of autonomous robots, the overall problem objectives can be quite diverse. Some robotic teams are charged with relocating a large object, too large to be moved by any one of the team's members, but not large enough to resist a cooperative push. Other teams are faced with a large environment, sparsely populated by points of interest. In this instance, it is more prudent to adopt a 'divide and conquer' approach rather than 'stick together and push.' Thus, the definition of 'good' behavior depends upon the robots' objectives, and their learning of such behavior depends upon the way in which they are rewarded or penalized for the actions they choose.

Despite the differences of objectives and resultant behavior between subdisciplines, the underlying algorithms and concepts are quite similar and have been shown to be applicable in more than one set of circumstances[17, 45]. A clever algorithm may boast high performance for specific environmental and circumstantial parameters, but if it is not robust to changes in these parameters, slight or otherwise, then it will have poor chances in real world environments where noise, sensor failure, and unpredictable conditions are quite prevalent. It is therefore desirable to create high-performance algorithms that are robust in these ways, and are executable across several domains without much modification or tuning. For a more complete exploration into the research of robot coordination and how these algorithms are employed, consult [31, 41, 16, 30, 54].

Within the field of robot coordination, and the main focus of this paper, is the pursuit domain. Also known as Predator-Prey and Pursuit-Evasion, the pursuit discipline of games is a domain introduced by Benda et al. [7] in which the objective of one group of agents (pursuers or predators) is the capture of another group (evaders or prey). Due to its extensive use in areas such as defense strategies[10, 13], robotic soccer[50, 23, 40], and dynamic target tracking[52, 33, 20, 19], an appreciable amount of literature has been generated from its research. With various environments, action-spaces, state representations, and reward schemes, all share the same goal in maximizing agents' respective objectives: pursue or evade.

1.2 State-Space Representation within the Pursuit Domain

Accomplishing successful pursuit algorithms is no trivial task. One challenge with achieving intelligent decision making in any MAS is extracting the system information (environment, agents in proximity, etc.) necessary and translating that information into a set of numerical values $\{x_0, x_1, ..., x_n\}$ that computing agents can process. These values form what is know as an agent's state $s = \{x_0, x_1, ..., x_n\}$, and it represents the agent's perception of the world. The concept of a state-space S (of which, every state is a part $s \in S$) is then introduced and it is simply the collection of all possible ways the world may be oriented to the agent.

A state can also be thought of as the list of inputs an agent can sense (be it environmental factors, positions/actions of other agents, agent specific information such as battery life, sensor information, etc.), and it is this information that influences an agent's learning and decision-making capabilities. Action-space A (of which all possible actions are a part $a \in A$) contains the ways in which an agent can carry out the decisions it has made based on its state (moving to a new position, communicate to human or other agents, actuate an appendage, etc.) and it is through these actions that an agent can interact with the world and experience new states (s'). Using the information given by the agent's state, it may learn to associate certain states with positive outcomes (rewards r), for example: moving towards the prey when adjacent to it results in capture. Through a heuristic process, predatory agents develop the behavior leading to quick, consistent capture. In fact, current algorithms and reward schemes have proven to be quite good at

tracking dynamic prey in well-defined environments[26, 40, 55].

The problem, unfortunately, is trying to scale these algorithms to scenarios where there are larger, more complex environments and more agents present. Due to inherent "curses of dimensionality" [34] within most state-space representations, the act of scaling results in drastically increased state-space size which can render learning unfeasible. It would then be preferable to have an algorithm that can track a moving target (prey), in addition to being applicable to any environment with any number of predators or prey involved in the game. A great deal of effort has been, and is being, put into clever learning and scaling tricks to overcome this dimensionality problem, but perhaps a simpler solution exists.

Quantitatively defining environmental surroundings (locale), nearby agent presence, (agents) location (orientation and/or position), and capabilities (possible actions) are all a part of identifying that agent's state/action-space. It would be possible to avoid some dimensionality problems if a state-space representation contained the previously stated information while remaining independent of the position and number of agents present. This paper explores a representation that accomplishes this by condensing an agent's surroundings into a simple, localized form. Paired with a basic reinforcement Q-Learner, this Condensed Observation of Locale and Agents (COLA) will show how minimal state information can lead to robust, environment- and population-independent solutions to the predator-prey domain.

A closer look into the pursuit domain can be found in Chapter 2. Here, the definitions of states, actions, and rewards will be discussed as well as previous

approaches in applying them. Also, the 1-step Q-Learning algorithm which all agent types will utilize will be shown and explained.

The problem setup, dynamics, and simulation are addressed in Chapter 3. Several state-space representations will be presented along with how they are computed and what they each mean conceptually. Section 3.1 will develop the world and its dynamics which all agents must obey. The actual simulation of the world and all agents within in it will be shown in Section 3.3.

Chapter 4 will discuss how the parameters and actions are selected in the Q-Learning algorithm. Several action selection methods will be presented and compared by performance. The top performers will be chosen for future simulations.

In Chapter 5, we will compare performances of proposed state-space representations for static and dynamic prey. Certain examples will show why some state-space representations are inherently ill-suited to several pursuit domain problems.

Many of the dimensionality problems existing the pursuit domain will be brought to light in Chapter 6. Agents will be placed in larger and more complex worlds where they need to transfer their learning from smaller worlds in order to remain competitive. This chapter will reveal the setbacks with state-space explosions and why they result from the state-spaces they do.

Chapter 7 will introduce real world conditions such as sensor noise and failure, comparing once again the performance of selected state-space representations. This chapter will show how sensor accuracy can degrade over distance, and how an adaptive agent deals with inaccuracies quite admirably. The benefit of learning is revealed in this chapter, showing that adaptation trumps intuitive policies in

unpredictable conditions.

Finally, Chapter 8 will revisit and summarize the results found in Chapters 4–7 as well as the contributions and relevancy of this work. In closing, there will be a brief discussion of ways in which this research may be advanced in the future.

Chapter 2 – Background

In the field of robot coordination/navigation, it is often the topic of research to successfully explore an unknown environment. Some cases involve a large team of robots in a complex world whose objectives are the measurement of specific points of interest [25]. The robots must either individually or collectively come within a certain distance of a target in order to accurately measure it. The predator-prey concept is rather similar, except instead of 'tracking' and 'measuring' a static target, it is the agents' objective to 'track' and 'measure' a moving target. Using the vernacular of predator-prey, one could instead say the predators are 'pursuing' and 'capturing' rather than 'tracking' and 'measuring.' This process of capture still requires that the predator-agents come within a certain distance of their prey. The major difference is that the target in this case is, sometimes, also given an objective: run away.

While it is the goal of the predators to be where the prey is, it is the goal of the prey to be where the predators aren't. It is this aspect that makes the pursuit domain such a challenge, teams of predators must learn tactics such as intercepting, flanking, and trapping; simply heading straight towards the prey will almost always allow it to escape [55]. Likewise, the prey must learn to do more than elude the closest predator, it must also take into account the other predators trying to corner it. The craftier the predators become, the craftier the prey must

become in order to survive. It is this back-and-forth building of tactics that makes Predator-Prey solutions such a powerful tool in modern day applications discussed in [49, 11, 30], it is also the reason why these problems are no trivial task to solve.

2.1 Pursuit Domain Approaches

There are several ways in which pursuit domain research is already being implemented in modern applications. Several institutions use teams of robots to show how buildings can be explored [19] and targets acquired using predator-prey algorithms [51]. Conceptual models of Pursuit-Evasion scenarios have ranged from hunting scenarios [37, 1] to a force of police officers searching multiple rooms for criminals [52]. Even in the Art Gallery Problem [12] where one must find the minimum number of guards necessary to keep valuables secured uses predator-prey methods [36]. Several of these models follow a pure mathematical graph solution and are quite abstract in concept, while others are modeled in physical simulations, generating empirical solutions.

A more functional form, one whose dynamics come from electrostatics in physics, says to treat enemy agents as repelling potential wells and target agents as attracting potential wells. This idea originates from the integration into the environment of biological pheromones distributed by insects to coordinate swarm behavior[32]. The extension comes in by allowing pheromones of a polar nature, meaning one type of pheromone is meant to be followed and the other is meant to be avoided. The electrostatic predator-prey approach modifies this by giving pheromones radi-

ally propagating intensity, so that the closer an agent finds itself to that pheromone source, the more it is repelled/attracted. This is a conditioned approach that gives the state-to-action mapping a mathematical formula to follow. It has the advantage of having quite low computational cost, but the drawback is that it must be heavily tuned in order to achieve the desired results in agent decisions.

Many approaches to the predator-prey domain use policies as a way to achieve focused agent behavior [51, 26, 52, 33] where the agents are presented with a series of if-then statements that define the action to take for whatever state is observed. This is a solid way of designing agents to do exactly what is wanted of them in controlled settings. However, this approach does not give agents the capacity to learn and adapt which, when faced with unforeseen circumstances, places them at risk for catastrophic failure. For example, an autonomous car that has recently gotten a flat tire may drive off a cliff because its policy always assumed there would be four, fully inflated tires. This problem may have been avoided had the car been given the ability to adapt to its new conditions. It is robust-conscience designs like this that put adaptive agents at an advantage. With this in mind, our goal is to create a robust agent that is affected by its interactions so that it can be introduced to new environments and conditions and still be able to learn useful behavior. A more exhaustive survey of the applications and environments (both physical and virtual) in which this research is found is conducted in [11, 49, 41].

Whether a designer opts to integrate a policy or adaptive style into the autonomous agent, the agent's state must first be defined. Recall from Section 1.2 that a state can be thought of as the agent's perception of world, or what informa-

tion that agent has to make decisions. Too much or too little relevant information can render decision making a difficult, and sometimes impossible, chore. Imagine eating at a restaurant that had a 2000 page menu containing every dish with every ingredient imaginable. Patrons would most likely have a difficult time extracting the information they need to order an entrée with which they were content. The same is true with computing agents, they are not required to know everything about their surroundings, in fact knowing too much can be detrimental to learning[9]. Likewise, too little information can cause agents to make incorrect associations which may develop undesirable behavior [55]. For example, say there is an autonomous agent in charge of making and stocking up ice cubes for refreshing drinks. If this 'refriger-agent' is only able to sense day and night, it would not understand why on some days there was high demand for ice and for other days there was almost none. Were the 'refriger-agent' equipped with a temperature sensor, it would correlate high demand to hotter days and thus be able to more efficiently make and stock ice. If a designer wishes to see efficiently performing autonomous systems, then agents must be given not only the right information, but the right amount of information.

As stated in Section 1.2, it can be a challenge to determine what the right information is and how it is presented to the agent as a state. In predator-prey scenarios, a popular state definition seems to be that of a global one[9, 38, 21, 40]. A global state is simply a vector containing every agent's position in the environment. For instance, in a discretized $n \times n$ gridworld containing N cells, a single agent would have N possible states. If agents are allowed to occupy the same position at

the same time, adding one agent to the gridworld makes the orientations increase to N^2 states. This is the "curse of dimensionality"[34] mentioned in Section 1.2, small increases in population and environment cause exponential increases in the state-space [3]. One can see that as the size of the gridworld and number of agents grow, the state-space can become quite unmanageable.

There are many clever ways out there that attempt to control the state-space size by reducing the field of view so that the state information is more local[55, 20, 19, 33]. Another is choosing a distributed or hierarchal form that reduces the dimensionality consequences, but does not extinguish them completely [3, 27]. As long as agent positions are included with the state generation, there will always be problems with scaling and learning due to the inherent 'curses of dimensionality.' The approach taken in this paper is to rethink how predators use their surroundings and senses to capture their prey.

When observing predator-prey scenarios in real life, a couple things become rather apparent. Let's take, for example, a soccer game in which three defenders are moving in to steal the ball from an opponent. If these players are experienced and competent defenders, they can achieve strategic positions around their opponent without the need for communication between themselves, or direction from their coach. If we were to treat these soccer players as computing agents, we could say that their state is what and who they see around themselves. Their positioning strategies are dependent upon their experience (learning), and the relative positions of their teammates and opponent (state). And based on their surroundings, they each make a decision of where to go next. What's more, is that these decisions

are made quickly, without debate or hesitation. This implies that their actions are a response to their state, and that response is built by experience. If they accomplish in stealing the ball, then they are psychologically rewarded for their efforts. If, however, they fail and the opponent scores, they are psychologically penalized and will probably try something different next time.

The soccer players behave in such a way that it implies their choices are purely reactionary, in other words, they see their surroundings and immediately map those surroundings to a learned action. There is no reason, then, to not believe that a computing agent can do this also. In situations where other strategies are not necessary (being near goals, etc.), the soccer players do not care how big of a field they are on (independent of environment), it does not change their objective of obtaining the ball nor augment how they are going to accomplish it. They also do not take into account all of their other teammates and opponents on the field, they only acknowledge those involved in the play (agent scalability). Already the players have eliminated two dimensionality problems by localizing their surroundings. A thorough explanation as to how we will attempt in mimicking this state-space representation is found in Section 3.2.3.

2.2 Reinforcement Learning

Recall that the soccer players mentioned in Section 2.1, above, do the following four things when attacking an opponent:

1. s: Evaluation of state

2. a: Choosing action based on state

3. $T(s, s')$: Transition to new state

4. $r(s, a)$: Resultant reward or penalty

Where $s \in S$ and $a \in A$. The reward function $r(s, a)$ where $(r(s, a) \in R(S, A))$ returns a reward to an agent that has chosen to take action a from state s. This 4-tuple decision framework is known as a *Markov Decision Process* (MDP) and is utilized quite extensively in Reinforcement Learning problems[35, 40, 22, 4]. The significance of this is that the predator-prey domain fits nicely into a Reinforcement Learning framework [43]. Thus, a basic 1-step Q-Learning algorithm will be chosen and implemented using several state-representations. Performance will then be attributed to the selection of state definition rather than how well each learns, as is done in most research. The basic 1-step Q-Learning algorithm is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.1}$$

Equation 2.1 is essentially a table that contains an agent's memory of the values, both expected and received, for all state-action pairs $\{S, A\}$ it's experienced. To explain how agents 'learn' utilizing Equation 2.1, we will look at a brief example from an agent's perspective.

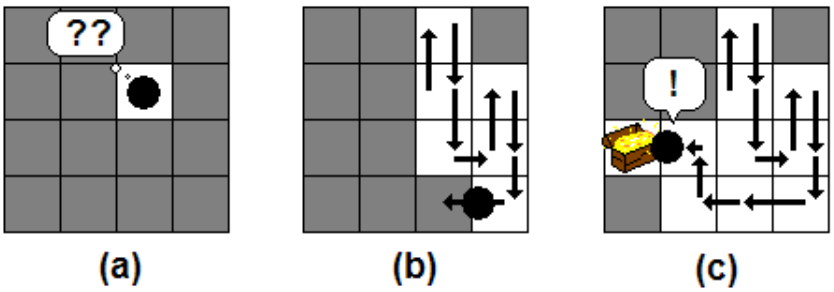


Figure 2.1: Agent in a New Environment - agents choose random actions while they are inexperienced. Rewards help shape their behavior to make coherent decisions in the future.

(a) An agent is placed inside an environment and is given no directions or instructions. It will consult its Q-table and find it has no experience whatsoever, and so will not know what to do.

(b) It will proceed to wander around aimlessly, all the while updating its Q-table to take note of the states it's been in, the actions it's taken, and the reward it's received.

(c) Eventually, an action from a specific state leads to a high reward. The agent will store this knowledge and use it for the next episode. Over several episodes, the agent will find the state-action pairs that yield the highest expected rewards, allowing it to quickly find the goal in subsequent episodes.

In Equation 2.1, the parameters α and γ dictate how an agent learns from past experience and estimation. A low value for α will mean that the agent does not rely heavily upon new information and rewards, this will result in slow learning. If $\alpha = 0$, no learning occurs at all. A low value of γ will result in the agent

not looking ahead to see what its expected rewards are from the new state it has selected. Increasing γ should increase the learning rate and hasten performance improvement. Refer to Figure 4.1 in Section 4.0.1 to see how different values of α and γ affect performance.

Though the agent has a method for storing its experience at this point, it still needs a way to select the best action from its Q-table. Intuitively, one would think that it is best to choose the action that corresponds to the highest expected reward. This is known as a *greedy* approach and is suspectable to settling for less-than-ideal decision chains because of its tendency towards local optima[2]. The greedy method can also be thought of as an *exploitation* of what it has learned to be rewarding actions. This means that the agent is potentially ignoring a large part of its state-action pairs, preferring to stay with actions it knows to be good rather than trying new actions to see if they might be better. Trying new things can be thought of as *exploration*, and an "exploitation/exploration" trade off ensues [28]. A way to break out of exploitive habits and encourage exploring is to occasionally pick a random action with probability ϵ, known as an $\epsilon - greedy$ approach.

Yet another way of selecting actions from the Q-table is by applying to each action probabilities that are proportional to their respective Q-values. The probability function P(s,a) for selecting a specific action i given the state follows the softmax selection equation:

$$P(s, a_i) = \frac{e^{Q(s,a_i)}}{\sum_{j=1}^{n} e^{Q(s,a_j)}} \tag{2.2}$$

```
while episode < episode_MAX
    do { Initialize Agent Positions
         while t < time_MAX
           do { Calculate state s_t
                q_t ← Q(s_t, a) Sample Q table
                a_t ← P(s_t, a) Select action
                Take action a_t
                New state s_t'
                r_t ← R(s') Receive reward
                q_t ← q_t + α(r_t + γmax_a Q(s', a) − q_t)
                Q(s_t, a_t) ← q_t Update Q-values
                if Prey = captured
                  then End episode
```

Figure 2.2: Q-Learning Update Process - pseudocode for an episodic learning process.

A comparison of these action-selector methods (greedy ,ϵ-greedy, and softmax) and their use in Equation 2.2 is found in Section 4.0.2.

In summary, the way agents utilize the rewards given and adjust their decisions for future iterations will be determined by their learning algorithms. Using a Reinforcement Q-Learner, agents will calculate their current state, consult the Q-values remembered for that state, and select an appropriate action based on that information. Q-values will then be updated using rewards the agents receive for taking the chosen action. The flow of this episodic learning iteration may be found in Figure 2.2.

Chapter 3 – Setup and Simulation

Because of the mutually exclusive nature of the agent objectives in pursuit-evasion problems, it would not be entirely useful to assess the system with an overall performance. Since either the prey is caught or it escapes, only one team of agents will receive an end reward and so the two objectives cannot both be met; one group must succeed while the other fails. For the purposes of this research, we will focus solely on the control of the predator and assess its performance on how quickly it is able to capture the prey. If the prey will behave in one of two ways: it will remain stationary, or pick random actions and wander. Either way, it will have no perception of the world and make no decisions, it will simply act as a target for the predator to track and capture.

3.1 System Dynamics and Environment

Choosing the environment in which the agents exist determines a considerable amount regarding the possible forms their action- and state-spaces will take. A continuous environment, 2D Euclidean e.g., allows agents to have continuous actions, 2D vector states, etc. Due to the digital nature of computers, world information must be discretized to some degree, but one can approximate a continuous environment by creating a gridworld with small enough cells, similar to what was

done with UC Berkeley's BEAR project [51]. One could also discretize a continuous environment into finite areas, like a building being divided into rooms and hallways [20, 19].

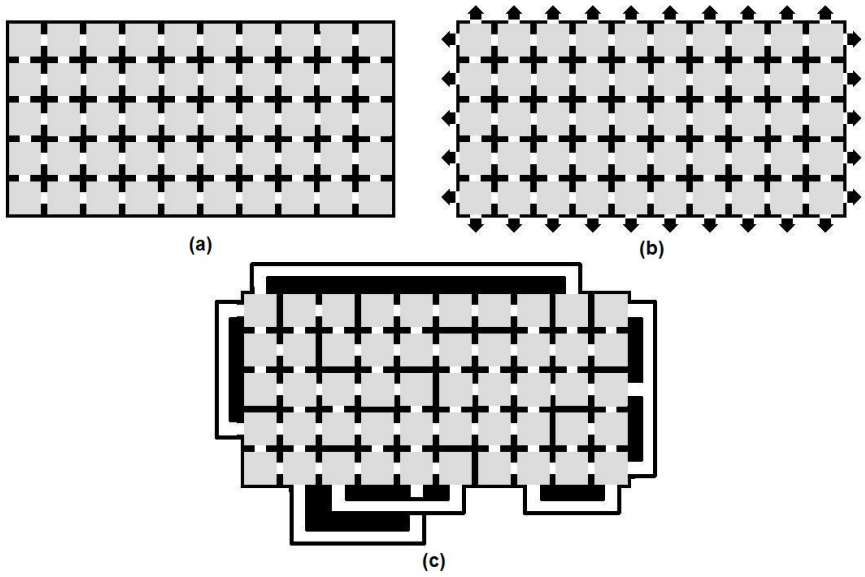


Figure 3.1: Possible Instantiations of the Proposed 'Gridworld' - (a) shows a finite gridworld with boundaries, while (b) removes the boundaries and gives uniform connectivity (creating a toroidal environment). (c) is still a finite world but has introduced a large degree of asymmetry into its connectivity.

For this paper, the agents will occupy a world that is discretized similar to those found in Figure 3.1. In this world, an agent may only occupy one cell at any given time, and it may not share its cell with any other agent while it is there. The spatial layout of (x, y) coordinates is convenient for us to understand, connections between cells are obvious, and we can quickly find paths from one cell to another. Computing agents, however, are not blessed with this intuition and need the world and all its properties presented to them in numbers alone, without any inferred

relationships. Thus, the world will be translated into a list of all the possible cells, and the cells to which they are connected. The symmetric 10×10 environment shown in Figure 3.2 shows how this translation works.

Node	Action			
	$\Leftarrow$	$\Uparrow$	$\Rightarrow$	$\Downarrow$
1	0	0	2	11
⋮	⋮	⋮	⋮	⋮
9	8	0	10	19
10	9	0	0	20
11	0	1	12	21
12	11	2	13	22
⋮	⋮	⋮	⋮	⋮
100	99	90	0	0

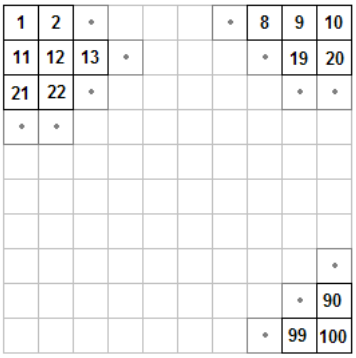


Figure 3.2: Environment Data Presented to Agents - an array containing all 100 cells in a 10×10 gridworld with a list of all connections by action. Note that spatial position is not conveyed to the agent in this form.

3.2 Defining States

For the simulations in this paper, we will explore numerous $1v1$ scenarios in order to answer the question: *which state-space representation offers the highest scalability and robustness?* These competing representations are the popular Position- and Relative-based state-spaces described in Sections 3.2.1 and 3.2.2 respectively, and the Condensed Observation of Locale and Agents (COLA) state-space briefly which will be discussed at length in Section 3.2.3. All representations will have access to the same environmental and agent information, but they will each interpret that

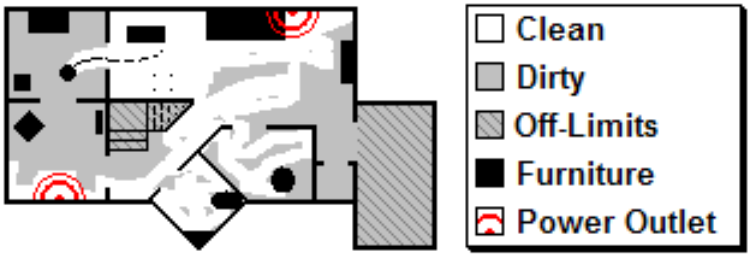


Figure 3.3: Example Environment for Position-Based Representation - a static world where rewards are only dependent on the robot's location is ideal for a representation that is defined by its global coordinates.

data differently in defining what their state variables are. And, as stated in Section 2.2, all representations will use the exact same learning algorithm, Equation 2.1, to show that the way a state is constructed can make a dramatic difference in how well one performs.

3.2.1 Position State-Space Representation

One of the more well-known and first to be thought of state-space representations is the Position-Based. This representation uses the environment data provided to determine its location in the world which it uses for its state. In static environments where conditions are slow to change, there is little need for anymore than just this since the outcome of any state-action pair will not change much from the last several episodes.

Imagine a robotic vacuum that needs to clean a large house, Figure 3.3. There are several rooms that are dirty and need cleaning, and a few rooms that the robot is not to enter. The robot will receive a penalty when it enters forbidden areas,

and rewards when it covers parts of the house that are still dirty. The robot also needs to recharge every once in awhile and so must learn the locations of power outlets around its environment. Since this house is unchanging, once the robot has learned the locations of good areas, bad areas, and power areas, it will be able to complete its task quite effectively. However, if the homeowners frequently redecorate and shift heavy furniture around, the robot will have to relearn some areas because the rewards it is now receiving are different from past experience. We should subsequently expect that this representation will perform well in static environments with static goals, but will have trouble in more dynamic situations.

From a gridworld domain, the representation could look something like what is seen if Figure 3.4. Because this particular representation takes into account its global position, it will possess pertinent environmental information. However, it will not be able to account for any sort of dynamic presence, such as environmental changes or other agents. Mathematically speaking, the state will take the following form:

$$s_{POS} = \{X, Y\} \tag{3.1}$$

If an agent finds itself in a rectangular gridworld of dimension $m \times n$, the total number of positions within that world is therefore $m \times n = mn \equiv N$. As s_{POS} is dependent only on its own set of coordinates, the resultant state-space size will increase proportionally to the total number of nodes in the world N, and therefore is on the order $O(N)$. Consequently, the number of agents present will not affect

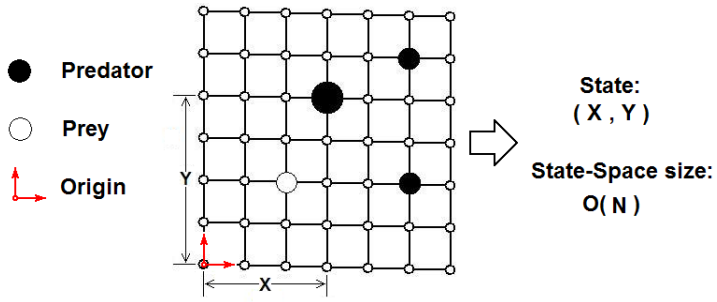


Figure 3.4: Position-Based Representation in Gridworld - an agent's state is found by calculating its location relative to the global coordinate system. It's state-space size is therefore on the order of N, the number of locations (nodes or cells) in the world.

the size of the state-space, thus will scale well with agent populations.

3.2.2 Relative State-Space Representation

Some stray away from the global state but still treat relative-agent positions as the state inputs [55, 20, 19, 33] and this is shown to produce the desired behavior in discrete worlds. A slight extension of its position-based counterpart, the relative state-space representation differs by measuring all agent coordinates relative to its own position. This gives it the advantage of being able to sense the proximity and direction of other agents, it also puts the state in a localized form that is independent of global position. The drawback is that it contains no information about the environment itself and so will not be able to accommodate a world that is dynamic. In most static environments, however, it should perform rather well.

If an agent with this representation is place in an $m \times n$ gridworld, then it will observe agents that are close (directly adjacent to its cell) to far away (on the other

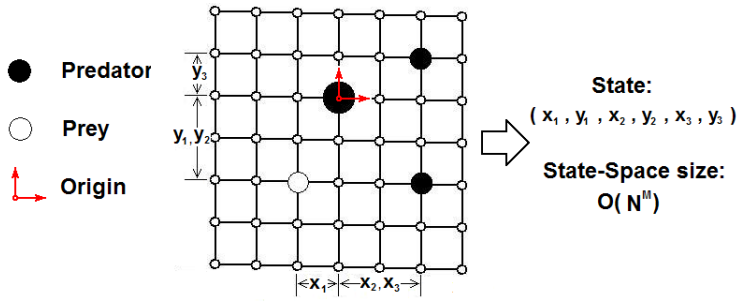


Figure 3.5: Relative-Based Representation in Gridworld - an agent's relative position to all other agents is the in which this state-spaced is compiled. Unlike a Position-based representation, the origin is set to its own position and so all global information is lost. The size of the state-space in this particular case is dependent upon both the size of the world and number of agents, making it on the order of $O(N^M)$ where M is the number of agents and N is the number of locations in the world.

side of the grid). In an $m \times n$ grid, the distance from one agent to another can be anywhere between $[-(m-1), +(m-1)]$ in one direction and $[-(n-1), +(n-1)]$ in the other. This means that the full interval is:

$$([-m, \ldots, -1, 0, +1, \ldots, m], [-n, \ldots, -1, 0, +1, \ldots, n])$$

Totaling $(2m-1)(2n-1) = 4mn - 2n - 2m + 1$ states. If N is the number of cells in the world, this state-space is on the order of $O(4N)$, roughly four times bigger than the Position-based size. What's more, is that for every agent added, a whole new set of states is available, increasing the state-space size to $O((4N)^{M-1})$ where M is the total number of agents. In a 10×10 gridworld with 3 agents, the state-space will have 160,000 possible states. Adding a single agent increases this further still, to 64,000,000.

This Relative state-space representation will have more difficulties in scaling up than a position-based because its size is dependent on both the world and the

number of agents present. It should, however, be able to track moving targets, something the position-based representation lacks the information to accomplish.

3.2.3 Condensed Observation of Locale and Agents (COLA)

In Section 2.1, we introduced an example from soccer that led us to some possibilities about predator/prey needing some exploration:

1. **Decisions are reactionary -** both predators and prey make decisions which do not require a high amount of processing time, thus choices may not be actions, but *re*actions.

2. **Little to no communication is required -** a group of predators is capable of coordinating without issuing auditory instructions or orders, using only the basic visual information available to form strategic positions and approaches.

3. **Immediate surroundings are most important -** other predators and prey which are far away may be ignored for most decisions made, meaning local information is recognized while much of world may be dismissed.

These three things imply that predators may be able to coordinate well with a simple state-to-action map. With this in mind, our objective will be to create this simple map using reinforcement learning and a state-space representation that attempts to estimate surroundings in a simplified form.

The Concensed Observeation of Locale and Agents (COLA) state will be the estimated predator and prey densities an agent observes around its current posi-

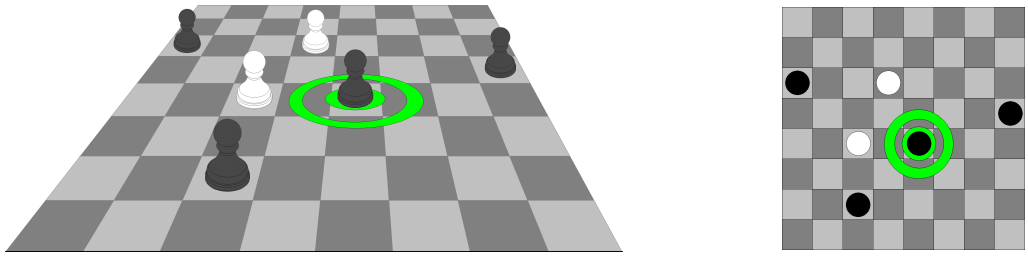


Figure 3.6: Chessboard with Dark and Light Pawns - pawns are allowed to move one space to any adjacent square, the objective of the dark pawns is the capture of the light pawns. The center dark pawn is highlighted to indicate that it is now its turn to calculate its state and choose an action. The image on the right is of the same board, but from the top view.

tion. Take for example the 8×8 board in Figure 3.6 with 2 types of pieces, dark pieces (predators) and light pieces (prey). If we were to look at the board from the perspective of the highlighted dark pawn, it would look something like what is shown in Figure 3.7

These pieces have the choice of moving to any adjacent square (either up, down, left, or right) and these will be referred to as the agents' options. Each option will have a value assigned to it equaling the estimated density of agents it observes following that action path. There are a couple rules that the COLA calculation follows in order make the representation more realistic:

1. **Observation terminates at objects and agents -** If an observation path intersects with something other than an empty node, it will terminate and return its value to the option with which it is associated.

2. **Non-terminating nodes are observed once -** Nodes may be observed by more than one observation path, but once they have been observed, they may not be observed again for that time step.

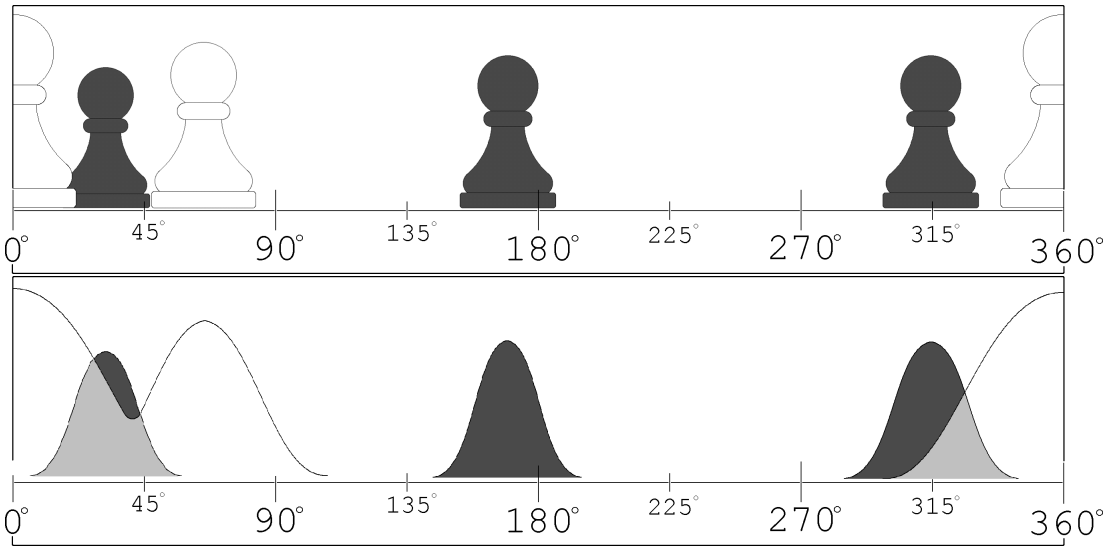


Figure 3.7: Environment as Viewed by a Predator - a 360° view of a predator's surroundings. The 0° heading is aligned with due west. Note that closer agents appear larger and take up more field of view.

The first rule mimics line of sight phenomena in that objects cannot be observed if an obstacle is blocking that path of observation. This is not to say that those objects are unobservable, it simply means that vision is blocked for that specific observation path. If, for instance, a mirror were introduced, the object could be seen down the observation path that travels to the mirror first. The second rule is meant to prevent unnecessary observation paths like those shown in Figure 3.12(a).

The value for each option is determined by summing the number of agents found along any observation path originating from one of the agent's set of actions, or options. Of course, the farther down a path an agent is, the lower impact it will have since it is technically farther away. Consequently, for every successive node the observation path takes, agents found there will have their value divided by a

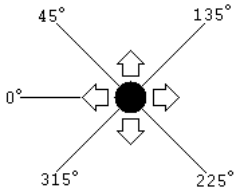


Figure 3.8: Actions and Their Associated Subfield of View - an agent's full field of view may be divided into its actions (*left*, *up*, *down*, and *right*).

distance factor, making them less noticeable.

To better explain this concept of condensing agents into option values and using observation paths to do so, we will explore the state created by the highlighted pawn in Figure 3.6. To reiterate, Figure 3.7 shows the world from this pawn's perspective, giving a 360° view of its surroundings. This view can be translated into agent densities seen at any given heading, with closer agents filling up more field of view, and farther agents taking up less.

Since the pawn only has four directions down which it can travel, it may be prudent to turn this 360° view, into a four action view. Noting that certain directions are associated with a subrange of the total field of view (for instance, the

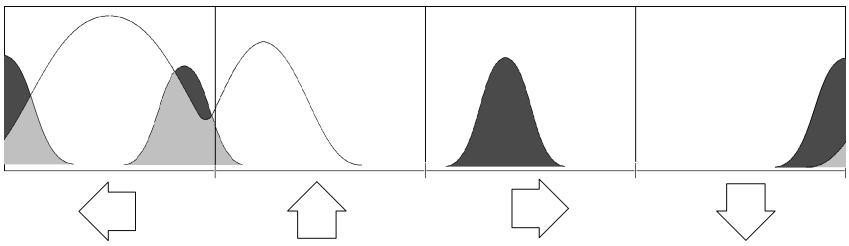


Figure 3.9: Agent Densities from an Action-Based Perspective - an agent may now see the densities of predators and prey down any action.

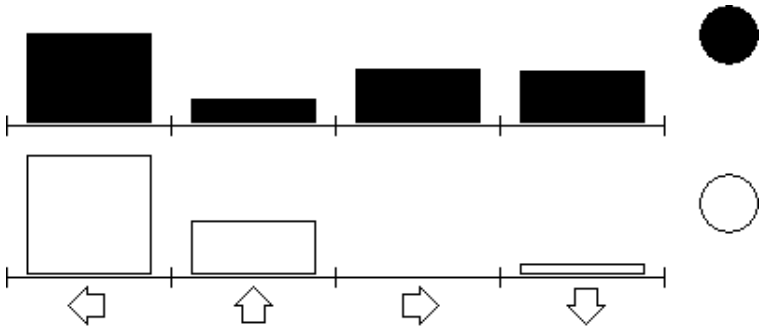


Figure 3.10: Condensed Observation of Locale and Agents (COLA) - state representation showing relative agent densities viewed down actions. This state is scalable in both gridworld size and number of agents.

action *left* is associated with the interval $[-45°, 45°]$), we can allocate equal parts of the pawn's view to its possible actions. Figure 3.8 shows what range will be associated with which action. The subsequent view, now oriented by action rather than degree heading, is what we see in Figure 3.9.

We can now sum the values within each action, for each type of agent (predator and prey) to give us the surrounding populations. Figure 3.10 shows how this representation now appears. The calculation of this state is fairly simple and can be shown using the pawn from Figure 3.6. We can translate the board into a grid of nodes which are connected to one another by lines. These lines represent vertices by which agents can travel and are therefore the movements associated with agent actions.

The world is now in a nodal representation, with agents occupying certain nodes and capable of traveling to other nodes via vertices, Figure 3.11 shows the transition to a nodal form. In order to follow the progression of vision, we will

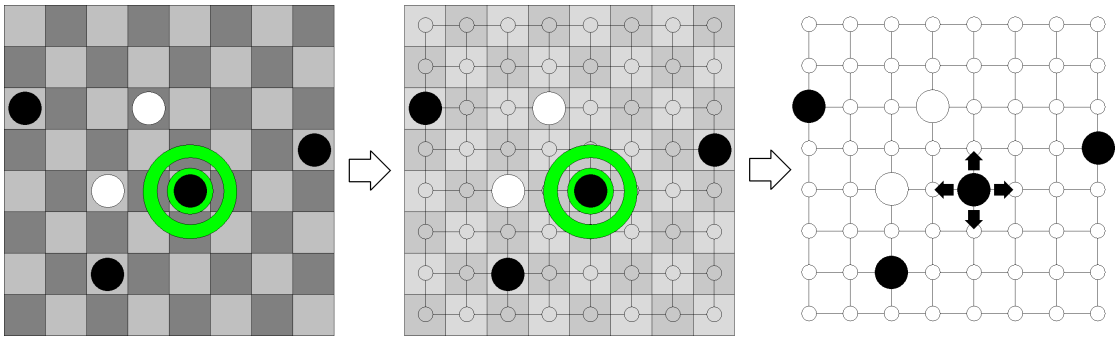


Figure 3.11: A Visualization of the Cell-to-Node Transition - cells can be treated as nodes, defined positions in a world, and connected to other nodes by vertices. These vertices are the actions an agent can take from the node it occupies.

use Figure 3.13. The observing agent will first sample all nodes (primary nodes) directly connected to its present location. The number of primary nodes determines how many options the agent has, and so a values will be found for each option showing what the estimated predator and prey densities are in that direction.

Let us follow the progression that takes this predator to the left. The primary node is empty and so the agent observes nothing to its immediate left. From there, all nodes (with the exception of the agent's current node) connected to this primary node are sampled. We see a prey agent to the left and so have observed another player on the board. At this point, the observation path: *Left* - *Left* terminates and returns its observation to the left-option value. This value will simply be a prey agent divided by how many steps away it is, in this case, 2. The final value for any option will then be:

$$V_a = \sum \frac{A_i}{d_i} \tag{3.2}$$

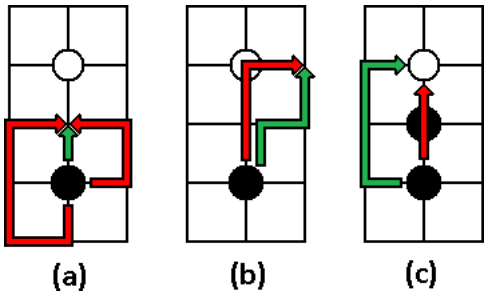


Figure 3.12: Unallowed Observation Paths - line of sight examples in a gridworld domain. (a) shows that there is more than one direction from which a node may be observed, only the shortest path(s) will be acknowledged. Both (b) and (c) show that observation paths may lead to other agents, but may not pass through them. Observation paths terminate at agents, walls, and objects.

Where V_a is value associated with an action, and the summation is over all valid paths belonging to that option. A_i denotes the agent type and d_i is simply the number of steps away the observed agent is.

Note that the observation paths *Left - Up* and *Left - Down* have not terminated and so continue in their progression. The sampling will continue with the next set of nodes (tertiary nodes) and the values will propagate back to their respective option value. If at any point a node is sampled and it contains an agent, that visibility path terminates and vision cannot extend past that node. Also, if an empty node was observed at a previous step, it is not allowed to be observed at a later step. This prevents redundant and infinite observation paths forming like those shown in Figure 3.12.

Each agent will have two sets of sensors, one set that locates predators and the other that locates prey. Both sensor sets will respectively create 'agent awareness' by generating estimates for the surrounding populations. Once all nodes have been

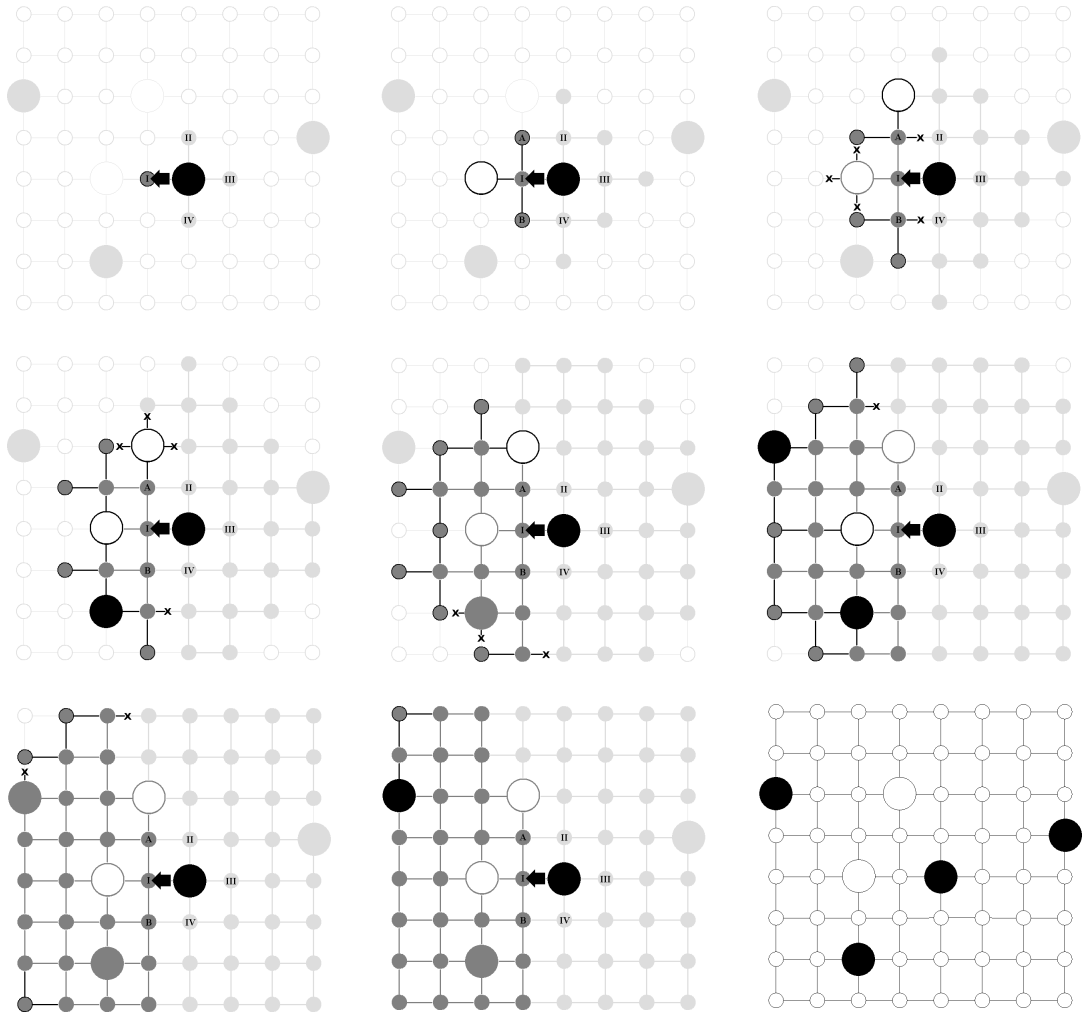


Figure 3.13: The Progression of Vision into Surrounding Nodes - observation begins at the observing agent, extends to all connected nodes, and then progresses to subsequent connections until the entire world is observed.

observed, our state looks quite similar to what we proposed in Figure 3.10.

Using this information, the predators should learn that regions of high prey densities means a reward is waiting in that direction. These density patterns will be used to find the desirable actions that return large rewards; it is the goal of this research to show that these state representations can allow agents to perform quite well in highly dynamic environments.

With this state representation, we have managed to create something that is scalable with both the number of agents and size of the world. The final goal we had was to make it localized so that it only focused on its immediate surroundings. We will therefore bin our state into a discretized form obeying the following:

$$s_{COLA}(a) \begin{cases} 2, & V(a) = V_{max} \\ 1, & V(a) = V_{min} \\ 0, & V_{min} \leq V(a) \leq V_{max} \end{cases} \tag{3.3}$$

Where $V(a)$ is the value associated with an option and agent type, V_{max} is the maximum value of V, and V_{min} is the minimum. We now have a state-representation that is localized, and scalable in world size and number of agents. Our state has eight state values applied to it, a set of two (one for each agent type) for each direction the agent can go. Since each value can be one of the following: [0, 1, 2], the size of our state is $3^8 = 6561$. This means no matter the world's size, shape, connectivity or the number of agents involved, our state space will remain this size. In the following simulations, we will be running $1v1$ scenarios and so

only half the state will be used (since the predator will observe no other predators on the board, those inputs will always be zero). This cuts our state space from 6561, to $3^4 = 81$ states.

Upon first inspection, it appears as though we've created a manageable state-space representation that is both localized and scalable. We will test the performance of this representation to those discussed in Sections 3.2.1 and 3.2.2. First, the simulator used needs to be explained.

3.3 Simulator

The simulations will be comprised of episodes, where each episode is another chance for the predator to catch the prey. At the beginning of an episode, agent locations will be initialized in the world; at no point may they occupy the same node at the same time. Depending on the simulation, the prey's starting location may be held constant from episode to episode, whereas the predator's starting location will always be random. Each agent will calculate their respective states, make a decision based on their observation, and execute that action. This will be defined as a *time step*, and every episode will have 50 time steps (t_{max}) before a new episode begins. All agents in the world will move simultaneously, then recalculate their states, take an action, and repeat till the episode is over or until the prey is caught.

Rewards: At each time step, the Predator will be penalized minutely for moving. This causes a lower overall reward the longer it takes to catch the prey,

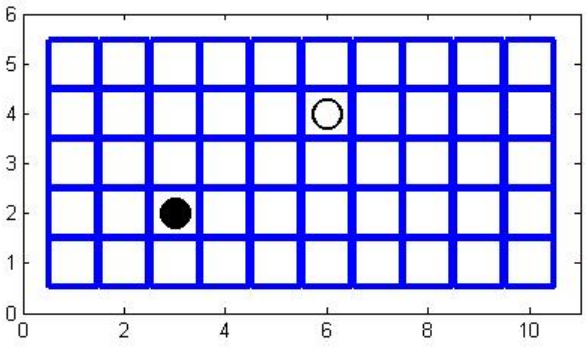


Figure 3.14: Simulator Screen for 1 v 1 - simulator screenshot showing a single prey (white) and a single predator (black).

influencing the predator to catch the prey as quickly as possible. Upon the successful capture of the prey, the predator will receive the reward R, 100 points was used in all simulations.

R : Awarded to the predator upon successful capture of the prey.

$-\frac{R}{t_{max}}$: Penalty for moving.

Collisions & Capture: When agents are adjacent to a wall or environmental obstacle, they still have the option of moving in that direction. If either the predator or prey runs into a wall or obstacle, it will remain in its original position and be penalized the move penalty. If however, the predator and prey run into each other, this will be defined as the *capture*. It is as this point the predator receives the capture reward R and the episode will end.

Chapter 4 – Q-Learning Parameter and Method Choice

Before the Position-, Relative-, and COLA-based state-space representations are compared, we will check our learning algorithm, the 1-step Q-Learner from Equation 2.1. There are two parameters that need to be chosen, α and γ, in addition to the action selection methods discussed in Section 2.2. To do this, we will run a simulation of a single predator and prey, measuring the performance for different values and selection methods until we have found the top performer.

4.0.1 α and γ Selection

The parameter α determines the ratio of kept memory to replaced memory. α exists on the interval $[0, 1]$, but if it is zero, no reward is saved to the agent's Q-table and so no learning occurs. γ also exists on the interval $[0, 1]$ and it sets how much of the look-ahead estimation is saved to memory.

$$Q(s_t, a_t) \Leftarrow Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$
$$Knowledge \Leftarrow (1 - \alpha)Memory + \alpha Reward + \alpha\gamma(Estimation)$$

For these simulations, a predator used the COLA representation and was given 1500 episodes over which to learn the location of a randomly placed target in a 5×10 gridworld. Figure 4.1 shows the average capture time to which each $\alpha - \gamma$ pair converged. Note that as expected, the convergence time for $\alpha = 0$ is quite

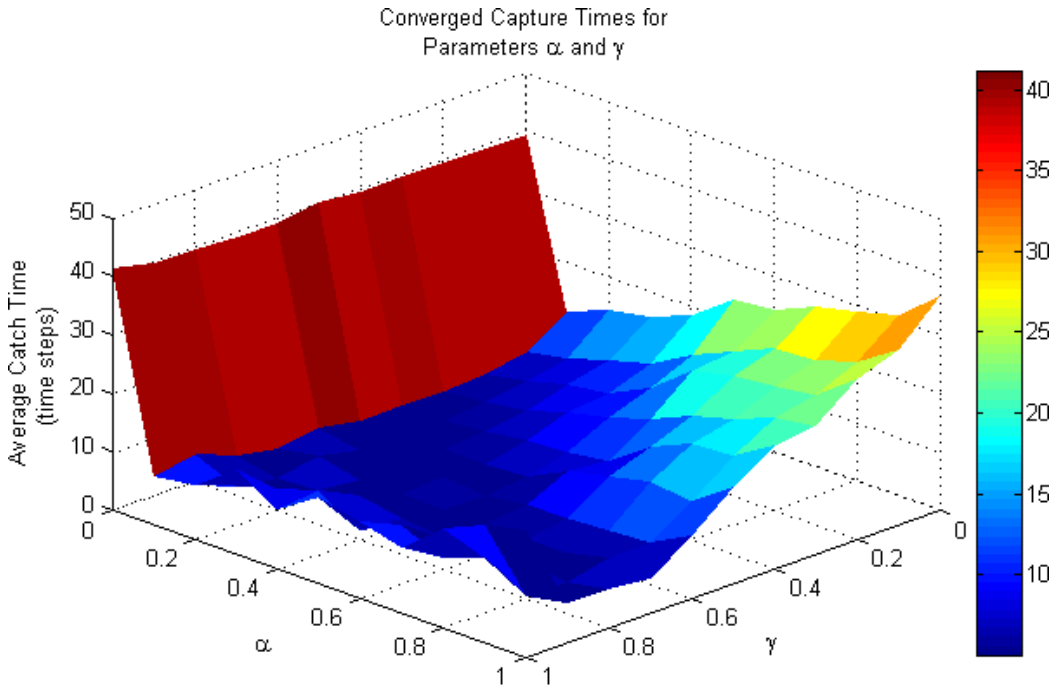


Figure 4.1: Performance Comparison for α and γ - several values for α and γ were chosen and their converged capture times are compared. All $\alpha - \gamma$ pairs converged by episode 500, and so the above values are averaged over episodes 500–1500.

poor due to no learning. We see that there is a comfortable region for us to select our two parameters, and so we will choose a pair that holds a low convergence time and is not close to pairs with drastically greater times. For the remainder of our simulations, we will use $\alpha = 0.4$ and $\gamma = 0.5$.

4.0.2 Action Selction Methods

We will now determine the primary action selection method that yields the lowest capture times. As discussed in Section 2.2, our options will be greedy, ϵ-greedy,

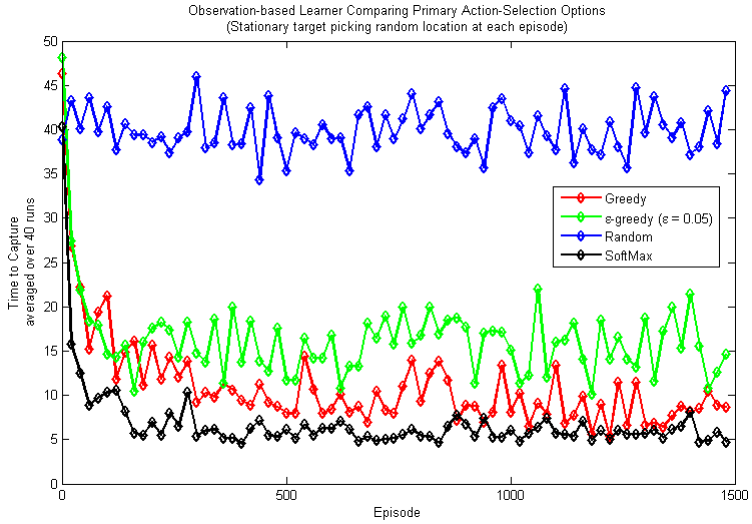
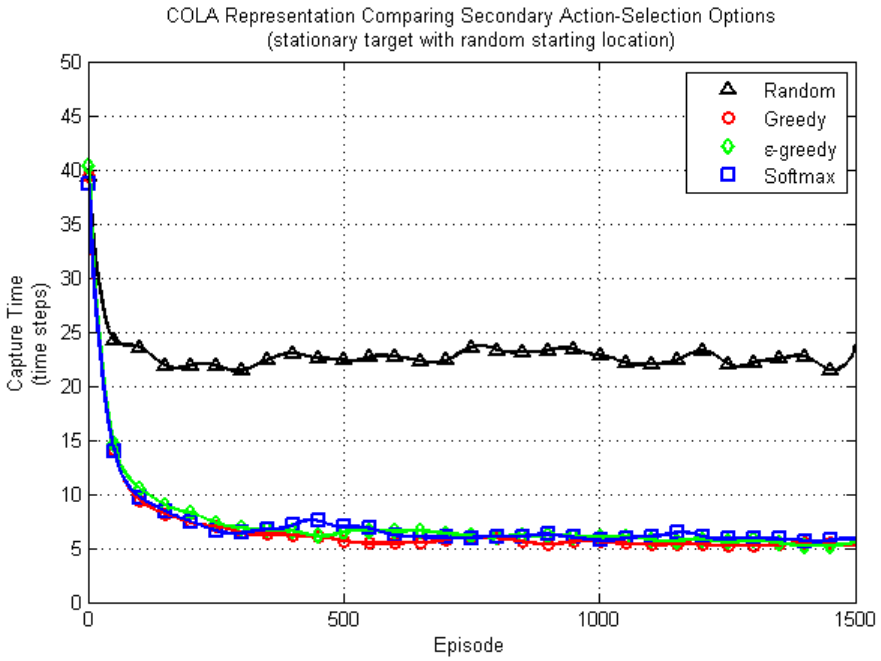


Figure 4.2: Performance Comparison for Primary Action Selection - from the Q-table, actions are selected from the values stored for each state using greedy, ϵ-greedy, and softmax methods.

and softmax. As in the simulation for finding α and γ, a single predator will do its best to find a randomly placed target. The agent will consult its Q-table and from those values choose an action by one of our proposed selection methods. Once again, the predator will have 1500 episodes to optimize its behavior. For statistical purposes, we will run this simulation 20 times and average the results. The performance for each method can be found in Figure 4.2. It is apparent that the softmax action selection method outperforms both the greedy and ϵ-greedy methods by an appreciable amount. Therefore, softmax will be used to select actions from agents' Q-tables.

From the Q-Learning Equation 2.1, we see that there is a Q_{max} term which means the maximum Q-value from the subsequent state is transferred to Q-value of the present state. This is essentially a greedy selector and we will verify if it



Converged Capture Time	
greedy	5.50
ϵ-greedy	5.98
softmax	6.06

Figure 4.3: Performance Comparison for Secondary Action Selector - from Equation 2.1, we compare performance when Q_{max} is replaced by $Q_{\epsilon-max}$ and $Q_{softmax}$.

is indeed the best. Figure 4.3 presents the capture times for a greedy, ϵ-greedy, and softmax secondary action selector. Our results show that the three perform equally with the greedy selector slightly ahead; meaning Q_{max} is indeed the best term to use.

These results are supported by Watkins' introduction and proof for the 1-step Q-Learning algorithm which uses the Policy Improvement Theorem found in Bell-

man and Dreyfus (1962). Their theorem states that the best possible performance is obtained by a policy of selecting a maximum value, all other policies will result in performance that is less than or equal to this optimum [6, 53].

We have shown that our selection of the parameters α and γ, and the method by which actions are chosen can have significant effects on how quickly and how well our agents learn. From these results we have determined that our agents perform well when our parameters α and γ equal 0.4 and 0.5 respectively, when our primary action selector follows a softmax probability method, and when our secondary action selector is kept at a greedy Q_{max}. We will therefore use these as our parameters and methods for the remainder of experiments.

Chapter 5 – Static and Dynamic Prey Results

In this chapter, we will compare the performance for the three state-representations described in Section 3.2. There are a couple terms which will be used throughout the results sections and they will be discussed here.

Prey/Target Start Location - refers to the beginning position of the prey/target at the start of each episode. If the start location is '*fixed*' then the prey's location remains the same from episode to episode. Otherwise, start location changes depending upon the change rate.

Static vs. Wander - describes the motion of the prey/target. Static indicates the prey makes no movement, wander allows it to pick random actions and move accordingly.

Convergence - when performance stabilizes and does not decline nor improve. The value of this performance asymptote will be referred to as the converged

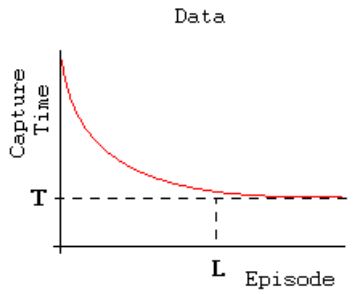


Figure 5.1: Convergence Plot - arbitrary data showing convergence to time T with a learning time of L.

capture time. (*shown as* ***T*** *in Figure 5.1*)

Learning Time - the number of episodes required for performance to converge.[1](*shown as* ***L*** *in Figure 5.1*)

5.1 Static Prey Starting Location

The first set of simulations will test how each state-space is equipped in finding a stationary target in a fixed world. For all episodes, the prey will be assigned to the same starting location, and throughout the episode, the prey will not be allowed to move.

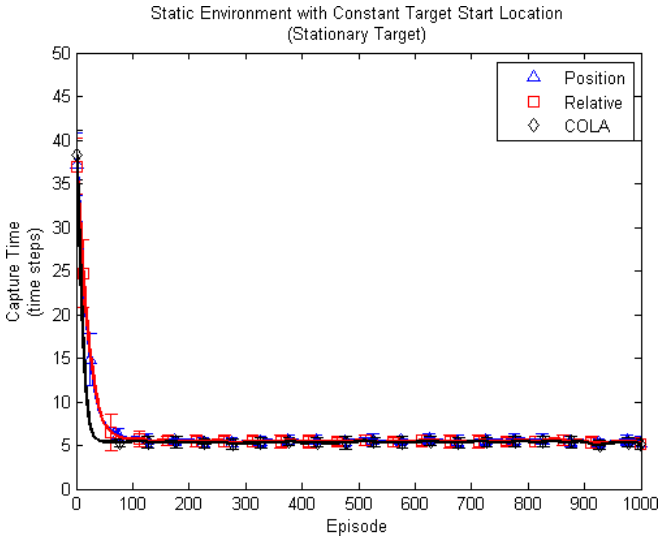


Figure 5.2: Stationary Target - the three-state space representations perform equally when finding and capturing a static target. Performance was averaged over 20 runs.

Figure 5.2 shows that all representations perform equally well, all converging

[1]For this paper, the learning time will be used to describe things qualitatively, and so does not need a precise mathematical definition.

to an average capture time of approximately 5.5 time steps. This was an expected result as it was stated in Section 3.2 that all representations would handle static environments/targets without much problem.

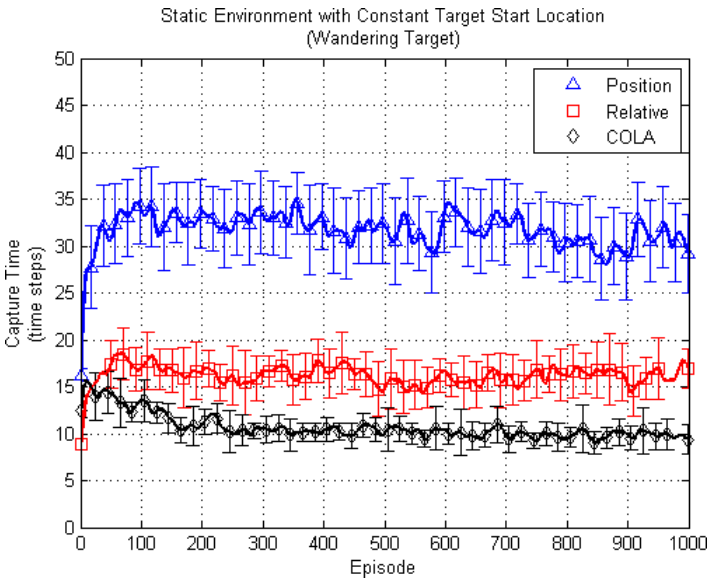


Figure 5.3: Wandering Prey with Constant Start Location - the prey picks random actions at every time step, but begins each episode in the same location.

We will now make the task slightly more challenging by allowing the prey to wander. Like last time, the prey will have a fixed starting location that remains the same from episode to episode, but after the episode begins, the prey will pick random actions to take, making it more difficult to catch. Figure 5.3 shows that the average capture time does indeed increase across all state-space representations, but the COLA in particular manages to keep its time 7 time steps below Relative and 20 time steps below Position. Position, of course, is not well-suited for a dynamic prey and so it comes as no surprise that it performs the worst of all.

Relative, however, is capable of tracking a moving target, as is COLA. The only reason Position as catching the prey at all is that it learns to go to the region where they prey starts because that is where it was accustomed to receiving the most reward. The prey does not wander far and so the Position-based predator randomly runs into the prey at times, it is not tracking the prey at all.

5.2 Dynamic Prey Starting Location

To further push the agents' capabilities, we will see how they fare when the prey is not required to have a fixed starting location. All three state-representations will learn on a stationary target which has a fixed starting location.

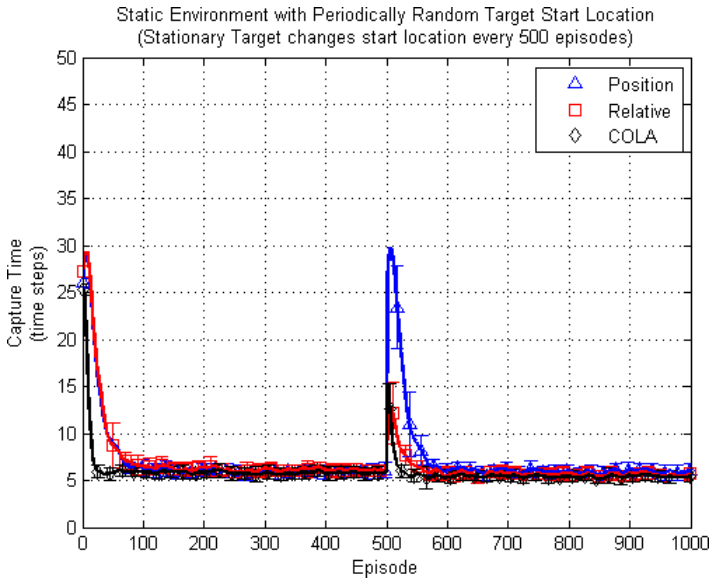


Figure 5.4: New Target Location every 500 Episodes - the prey remains fixed for 500 episodes until it is assigned a new starting location at episode 500 and remains there for the duration of the simulation.

After they have been learning for 500 episodes, the prey's starting location will be placed somewhere else and held there for the remainder of the simulation. Figure 5.4 shows that all three state-representations are confused by the prey's sudden change and must readjust their behavior. It is not long before they all learn the prey's new position and proceed to catch it quickly every time.

Now the prey will change locations every 250 episodes, twice the frequency. In this case, we see in Figure 5.5 the same result; all three experience a sudden increase in their capture times but quickly recover. We also see that the Position-based representation is most affected by the altered conditions, meaning that it is the least suited for change.

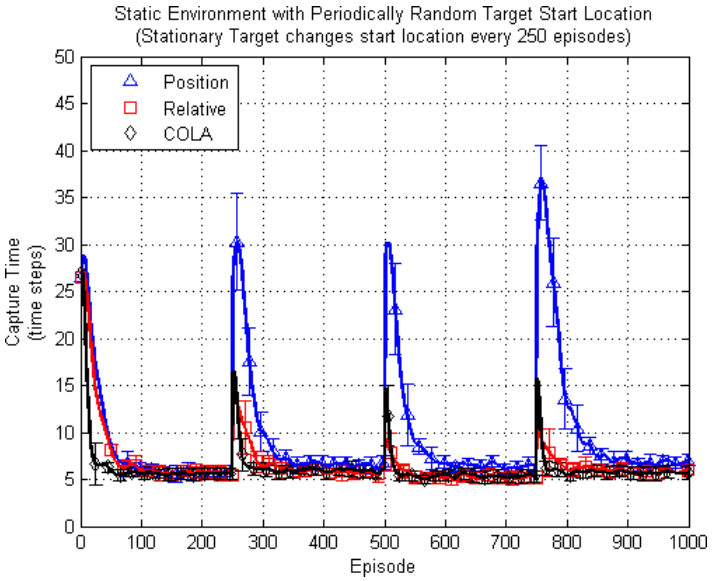


Figure 5.5: New Target Location every 250 Episodes - the prey is assigned a new starting location every 250 episodes. Agents must relearn their target's location at every new assignment.

When we increase the frequency even further to a prey location change every

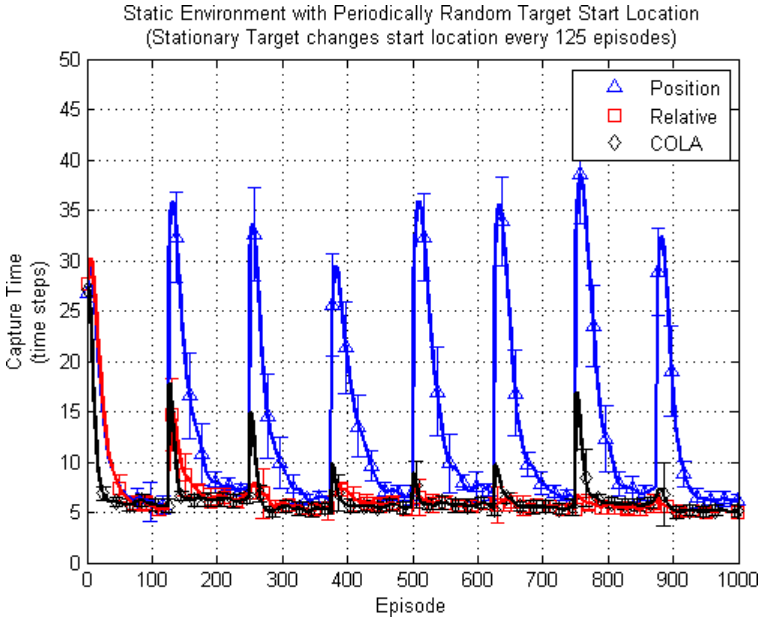


Figure 5.6: New Target Location every 125 Episodes - the Position based-representation shows that it is still, significantly affected by the prey's random changes while the COLA and Relative representations are becoming less and less affected.

125 episodes, something interesting begins to happen as seen in Figure 5.6. The Position-based representation continues to be thrown off by the prey's new location, but the COLA and Relative representations are thrown off less and less as the episodes increase. This shows that they are now learning to track a moving target and are beginning to show that changes in the prey's location will not disrupt their performance anymore.

As the frequency of random start locations grows to once every 50 episodes, we observe in Figure 5.7 that the progression continues. The position-based representation is now unable to recover and is not performing as well as COLA and Relative. The latter two have now become quite accustomed to the periodic changes and are

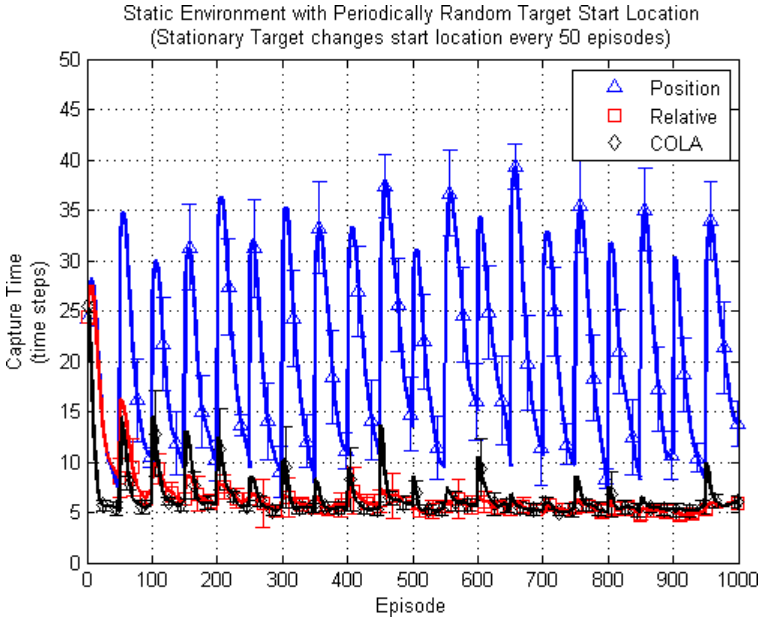


Figure 5.7: New Target Location every 50 Episodes - target location changes faster than a Position-based representation is capable of learning, and so results in poor performance. Both COLA and Relative are affected by subsequent changes less and less as episodes go on.

relatively unaffected by the target's new locations. Note that the learning time (the number of episodes required to converge) is consistent for each state-space representation. No matter the periodicity of the target's change in start location, the Position-based representation needs about 100 episodes to achieve convergence. The COLA- and Relative-based representations, however, require 25–50 episodes to converge, enabling them to perform their best in, at most, half the time. The reason we see the Position-based representation do so poorly is that the world is changing twice as fast as the representation is capable of learning.

Finally, we will randomize the target's starting location at the beginning of every episode. Figure 5.8 shows the Position-based representation completely in-

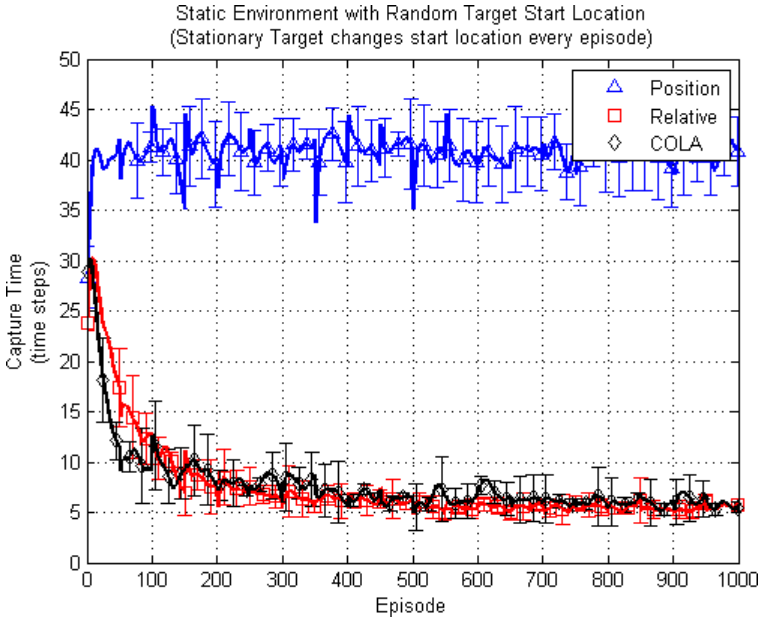


Figure 5.8: New Target Location every Episode - in a dynamic target environment, the COLA- and Relative-based representations are able to converge to a low capture time, whereas the Position-based performance destabilizes completely.

capable of finding the target. Both COLA and Relative, track the target quite well, achieving the same average capture time of 5.5 time steps as was done in the completely static case. Note, however, that the learning time drops to roughly 400–500 episodes, meaning that it takes significantly longer (approximately 10 times longer in this case) for these state-representations to learn a dynamic target than a static one.

We have seen that the frequency at which the prey changes locations can affect the learning time and convergence of the predators. Figure 5.9 consolidates this information, showing the converged capture times as the prey location change rate increases. COLA and Relative converge to about the same value of 5.5 time steps,

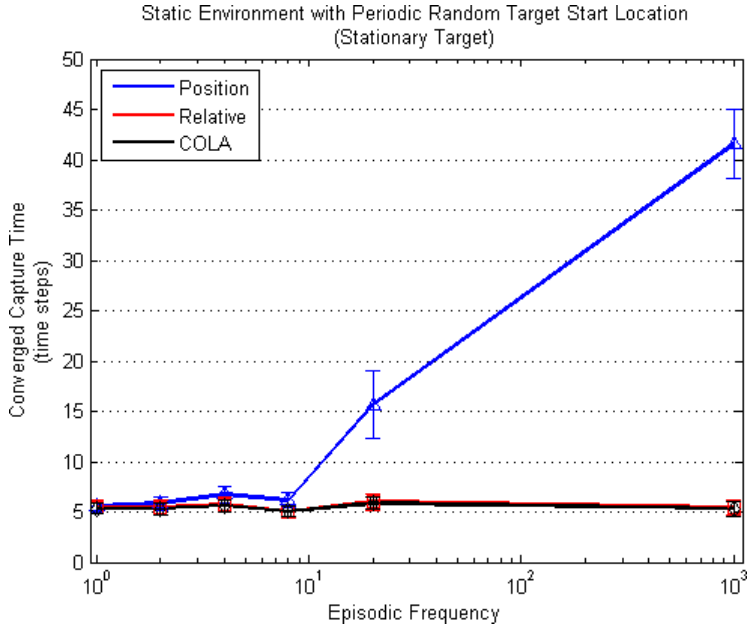


Figure 5.9: Frequency of Random Prey Start Location - the converged capture times when prey changes its starting location every 500, 250, 125, 50 and 1 episodes. Only the Position-based representation is affected.

and this is consistent no matter the periodicity of the prey's randomized starting location.

We will take the agents that have learned on prey changing starting locations every episode and test them on a world in which the prey is no longer forced to remain stationary. Similar to the simulation conducted in the previous section, Section 5, the prey will choose random actions and wander through the environment. We will compare how well the agents transition from their learning environment to this new, test environment. In Figure 5.10, all agent performance curves begin where they left off in Figure 5.8 because that was their training domain. Once the prey is allowed to wander, the Relative-based representation slightly drops in

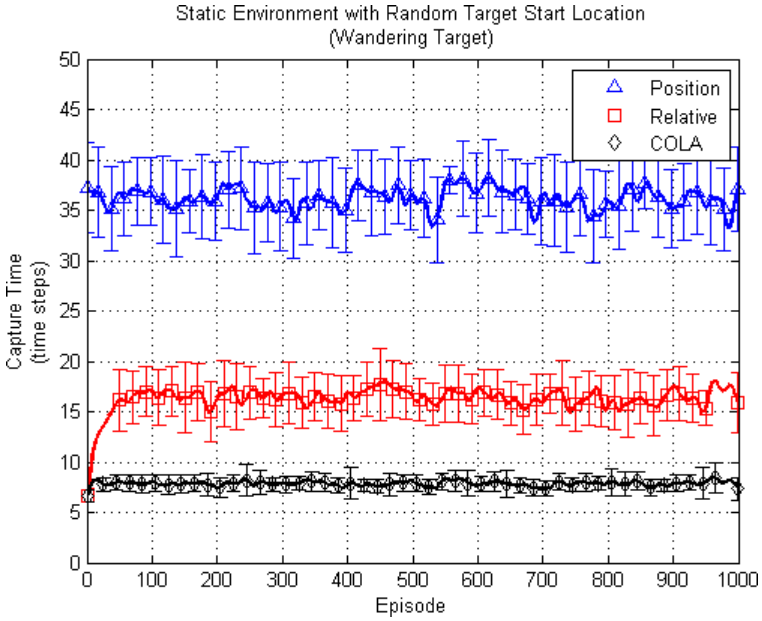


Figure 5.10: Wandering Prey with Random Start Location - the prey is now allowed to wander, learning on randomly placed prey seems to improve performance over learning on fixed prey.

performance, meaning it is having a more difficult time tracking the prey. The COLA representation is able to perform somewhat the same with only a minor increase in capture time, 2–3 time steps.

We can compile the converged capture times into a table to see how much the times change when conditions are modified. When the prey's starting location is fixed, all agent types do well on static prey, when the prey is allowed to wander, the performance declines for all. The Position-based state-space representation does not have sufficient information to track dynamic prey and so is not showing low capture times. Faring much better is the Relative-based representation which still suffers a drop in performance, but not to the extent of the Position-based. And

in the lead is the COLA representation, which only increases its capture time by 2 time steps transitioning from a static to wandering prey. Trends are consistent for when the prey is assigned a random location.

Starting Location	Fixed		Random	
Prey Motion	Static	Wander	Static	Wander
Position	5.55±0.6	30.46±4.1	40.79±3.5	36.11±3.9
Relative	5.49±0.6	16.33±2.7	5.35±0.8	16.31±2.6
COLA	5.37±0.6	9.81±1.4	5.69±0.7	7.84±0.9

One thing to point out, though, is when a comparison is made between the capture times for a wandering prey with fixed starting location and a wandering prey with random location. The COLA-based representation manages to improve its capture times by 2 time steps when the start location is random. The reason for this is that in a more dynamic setting, the agent is exposed to more conditions and therefore observes more states. This allows the agent to learn more and results in better performance.

Chapter 6 – Transfer Learning: Extension to New Environments

When agents are placed in new environments (larger worlds, sparsely connected worlds, worlds with obstacles, etc.) they must be capable of preserving their performance lest they need to relearn their behavior all over again. Transfer Learning [43] is the concept of being able to take what was learned in one domain, and apply it to one that is different. For these simulations, agents will learn in a static environment where the prey changes starting location every time. After achieving convergence, they will be placed into a new environment and are charged with finding the prey.

6.1 Expanded Environment

The first test will be to see how these agents deal with a larger environment. As previously stated, the agents will learn in the 5×10 gridworld and then be placed in a larger, 30×30 world. This new environment now has 18 times more nodes than the learning environment. Since both the Position- and Relative-based representations have sizes dependent upon the number of nodes in a gridworld, their state-space will increase proportionally, and therefore be at least 18 times larger as well[1]. The COLA-based representation does not increase in magnitude

[1]Recall from Section 3.2.2 that the Relative state-space size is on the order of $O(4N)$ and so will be closer to 72 times larger

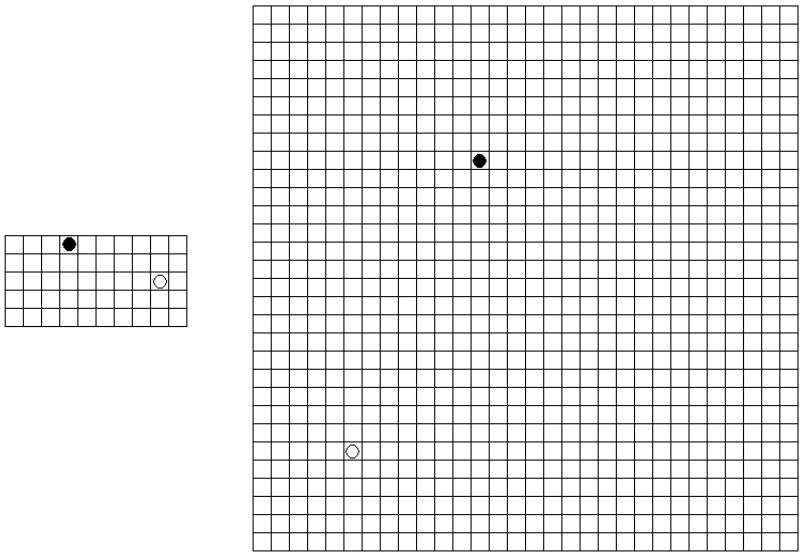


Figure 6.1: Expanding a 5×10 Gridworld to 30×30 - an increase in gridworld size results in potentially more states and larger state-spaces. This particular world is now 18 times bigger than the original 5×10.

as it is independent of world size.

In Figure 6.2, we see similar initial performance as was observed in Section 5.2. At episode 500, the world is expanded and the maximum time allowed to agents is doubled to 100 ($t_{max} = 100$). As a result, we witness the immediate decline in Relative-based performance. This is due to the 18-fold increase in state-space size. Like usual, the Position-based representation is not much of a contender. It is now taking them both the full 100 time steps to to end the episode meaning that they are failing to capture the prey most of the time, while the COLA-based representation manages to keep below 30 time steps. If the simulation were allowed to continue for hundreds of episodes, both the COLA- and Relative-based representations would converge to a new time, the COLA is already at an advantage and would have a

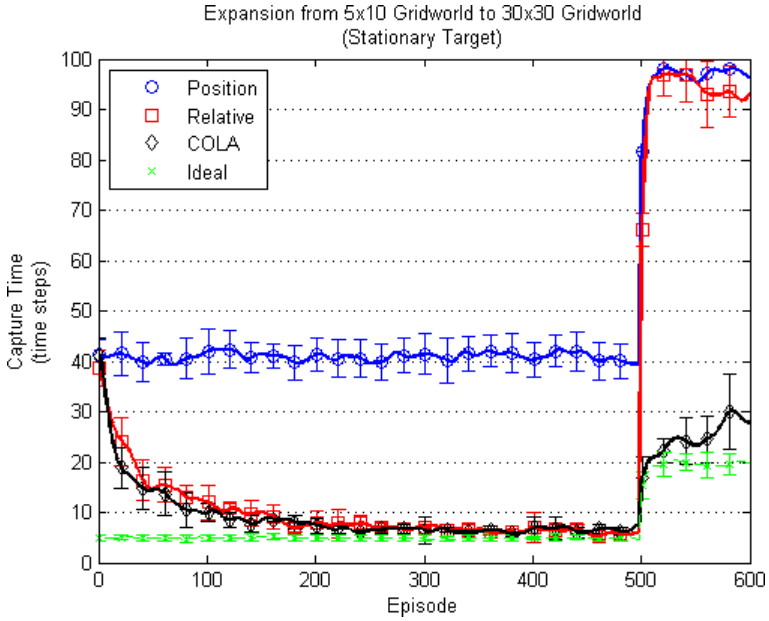


Figure 6.2: An Expanded Gridworld - this results in an increase in state-space for both the Position- and Relative-based representations, but not the COLA-based one. Performance is reflected by this fact.

much faster learning rate. Position will not converge do to this being a dynamic simulation.

This result shows that the COLA-based representation truly does scale with world size quite well and should be able to handle any sized environment with little additional learning.

6.2 Environment with Obstacles

As it was shown in Sections 5.2 and 6.1, the Position-based representation is incapable of dealing with dynamic environments and is not much of a contender. It

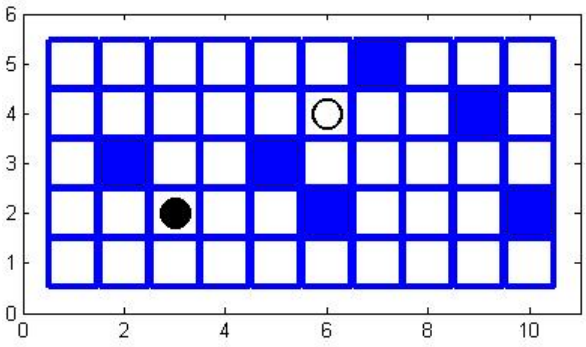


Figure 6.3: Simulator Screen with Obstacles - a screenshot showing a single predator (black), prey (white), and several filled in cells which represent obstacles.

will be excluded from future results as it is more of the same. In its place, we will introduce a new contestant.

Policy - this agent will use the same COLA state-space representation, but will not use a learning algorithm to determine its behavior. Instead, it will follow a hand-coded policy written to achieve the quickest and most efficient capture times.

The COLA representation will now be compared against the Relative representation and this new Policy agent. For the next simulation, the agents will be placed within an environment that has obstacles (nodes that are occupied by objects other than agents). Figure 6.3 shows the simulator layout for this circumstance. Like before, agents will learn in a static environment with a randomly placed prey. At episode 1500, 6 obstacles will be randomly placed in the 5×10 gridworld. Figure 6.4 shows that the hand-coded policy is almost unaffected by the introduction of this new environment. Both learning agents, however, take a considerable hit in

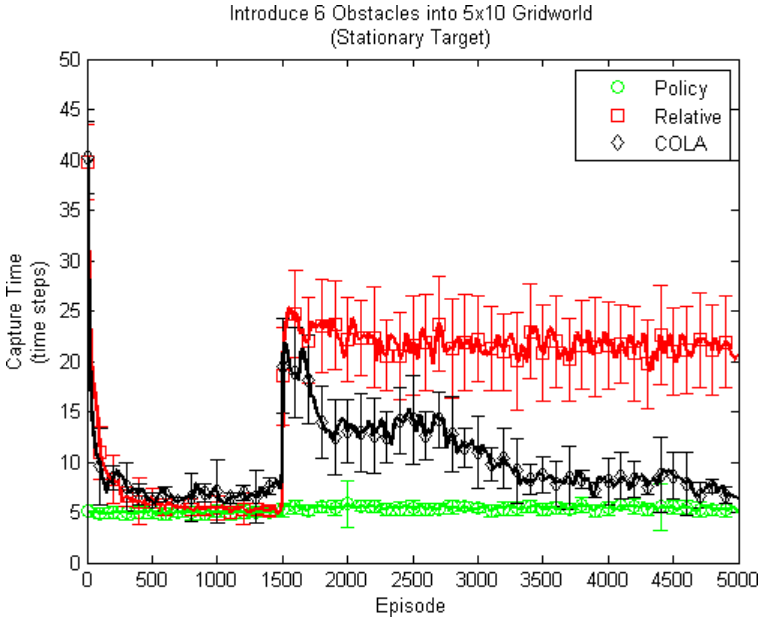


Figure 6.4: Introduction of Obstacles into Environment - when the environment contains obstacles, the Relative-based representation has difficulty in maintaining its lower capture times.

performance, a hit from which the Relative representation cannot recover. Although it takes 3500 episodes, the COLA representation is able to account for the environment, eventually performing as well as the Policy.

As was the case with the Position-based representation, the Relative-based representation is unable to compete any further as it has been shown to fail in transfer learning. And so, only the COLA and Policy agents will remain for the duration of our simulations.

We will take the previous simulation which introduced an environment with obstacles, and modify it so that the environment becomes dynamic. The 6 obstacles will remain in place for 10 episodes, at which point they will pick new random

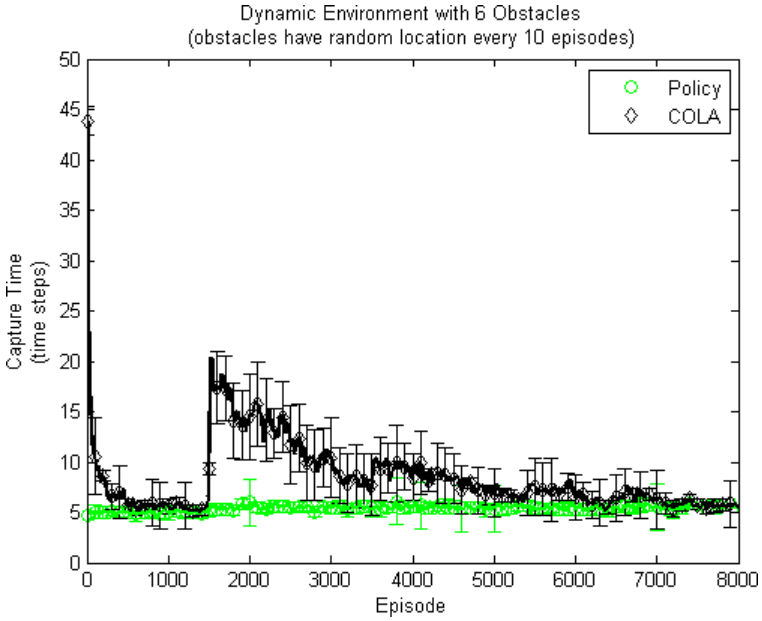


Figure 6.5: Dynamic Environment with Obstacles - 6 obstacles are placed randomly within the world every 10 episodes, where they remain until new locations are selected. This simulates a dynamic environment.

locations. This will simulate an every-changing environment to test these agents in seeing how they perform in a dynamic world.

Figure 6.5 shows that even though the world has become quite dynamic, the COLA-based representation is still able to track and capture the target. This time, however, it takes 4500-5500 episodes for it to converge and match the Policy. At this point, one must ask the question: *Why even use a learning agent if a hand-written policy will always do at least as well or better?* The answer to this question will be found in Chapter 7 as we continue our investigation into the COLA state-space representation.

Chapter 7 – Algorithm Robustness

In the real world, no sensor is perfect. And as sensor resolution, precision, and compactness increase, so does cost. As such, robots must learn to deal with less than the best and perform well despite their technological disadvantages. All state-space representations must be generated by data gathered from sensors. The unfortunate catch with sensors is that they have noise, and can sometimes malfunction. This results in the robotic agent perceiving a deception of the world because the information it's received from its sensors is faulty. Robust algorithms and state-space representations must be able to account for this otherwise they will fail along with their sensors.

In this chapter, we will augment the COLA values to simulate noise. Recall from Section 3.2.3 that the state-space is constructed by measuring the distances of nearby agents. In practice, however, a series of sensors would conduct the measurement and from that the state would be calculated. Any error in the sensor measurements will result in noise and thus affect the perceived state.

7.1 Performance with Sensors of Varying Noise

To see how significantly the perception changes as noise increases, we can conduct a simple simulation by measuring several states with noise and without and

comparing how they match up. A predator agent will be placed inside a 10×10 gridworld along with a single prey agent. For every possible instantiation of the predator's and prey's location, we will measure the COLA state using the technique described in Section 3.2.3. We will then take a new measurement with some added noise and see if the agent can determine the correct state.

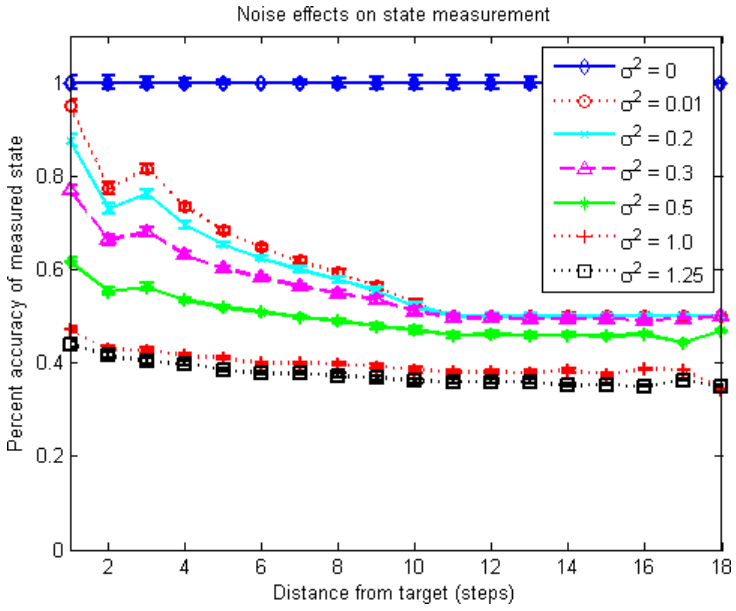


Figure 7.1: Noise Amplitude Effects on Perception Accuracy - perception accuracy degrades with both noise amplitude and distance from target. Even at $\sigma^2 = 1.25$ ($noise_{amp} \approx 100\%$), we see the accuracy does not degrade much past 40% for the distances shown.

For the sensor noise, we will use random white noise generated from a Gaussian distribution:

$$S_{noisy} = S_{pure}(1 + noise) \tag{7.1}$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{7.2}$$

Figure 7.1 shows that the perception accuracy is certainly affected by noise, even at the slightest amount. As noise increases, we see that the accuracy degrades as expected. We also observe that the perception accuracy is affected by the distance; this is a reasonable finding as it makes sense that the farther away something is, the more difficult it will be to measure it accurately. Although we have found noise to affect perception quite a bit, it may not have quite as drastic an effect on performance.

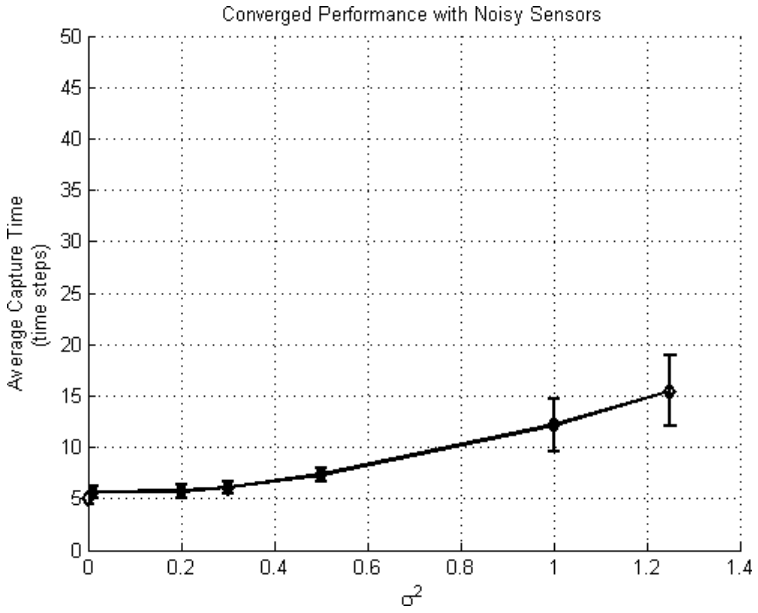


Figure 7.2: Converged Capture Times for Various Noise - the time taken to capture the prey increases as the noise amplitude increases. For noise levels up to $\sigma^2 = 0.5$, the COLA agent is able to keep capture times below 8 time steps.

To test this, the $1v1$ scenario will now be conducted with noise. Figure 7.2

shows that the effects on performance are not as drastic as Figure 7.1 would imply. Even though there is roughly 50% perception accuracy at $\sigma^2 = 0.5$, the capture time associated with that noise level increases by less than 2 time steps. These results show that a COLA-based state representation is robust to sensor noise.

7.2 Performance with Sensor Failure

Now that we've shown that a COLA representation is even capable of handling sensor noise, we can now see how the Policy and COLA learner fare in a situation where their sensors are noisy, and one of their sensors fail. When we say "sensor," one of the four state values is meant. Essentially, we will modify this value to simulate a failed sensor.

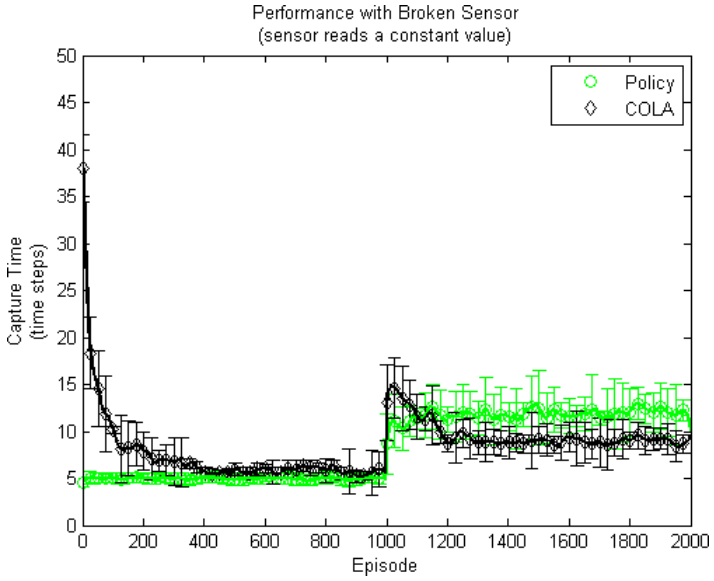


Figure 7.3: Performance with Broken Sensor - one of the four direction sensors outputs a constant value within interval of [-0.5, 1.5].

In this first simulation, one of the four sensors will go dead and output a constant value no matter what state is observed. At any given moment, a high agent density would be represented by an observed value of ≈ 1, meaning roughly 100% of that specific field of view is occupied. Likewise, no agents results in a value of zero. The broken sensor's value will therefore be chosen from this range, slightly expanded to $[-0.5, 1.5]$.

Figure 7.3 shows that the Policy and COLA learner both are affected when their respective sensors break at episode 1000. The Policy is unable to improve its performance because its behavior is hardwired into it. The COLA learner, however, is capable of adjusting for its failed sensor and it eventually does better than the Policy, finally converging to a capture time which is 25% faster than the Policy.

Now we will see what happens when a sensor malfunctions and outputs a random number within the sensor range of $[-0.5, 1.5]$. This simulation differs in that the unreliable sensor now changes its value every time step, making it complete noise with no preservation of original signal. Figure 7.2 shows the Policy is greatly affected by this failure and cannot recover from it. Its capture times increase to an average value of 31.8 time steps, while the COLA learner is able to keep its times at around 13.6 time steps, 134% faster than the handwritten Policy.

These examples have shown that adaptive algorithms are quite powerful, being able to experience unpredictable circumstances and still capable of extracting necessary information to make useful decisions. Sensor failure and noise are quite common and severely alter an agent's evaluation of its state. Given the above re-

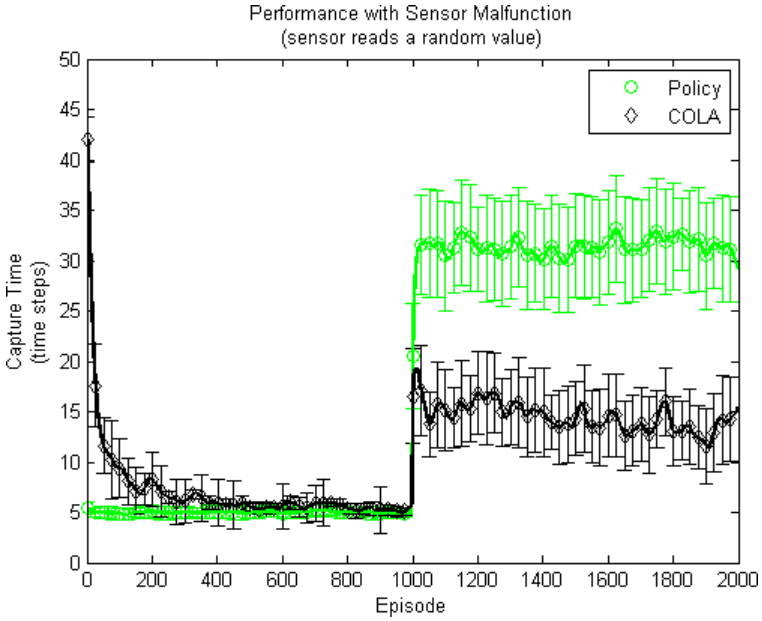


Figure 7.4: Performance with Malfunctioning Sensor - one of the four direction sensors outputs a random value within interval of [-0.5, 1.5] at every time step.

sults, the difference between a learning agent and a policy agent very well could be the difference between a successful and failed mission. This final robustness check for the COLA state-space representation shows that when coupled with a basic reinforcement learning algorithm, this is potentially a powerful way to perceive the surroundings in the predator-prey domain.

Chapter 8 – Discussion

The selection of state information for agents in a learning situation can heavily influence how efficiently good behavior is obtained. Designers must keep agent states simple enough to reduce computational time but also deliver enough information so that agents may make intelligent decisions. The challenge is in identifying what information is pertinent, and how to represent said information in a numerical format.

In this paper, we have investigated the preliminary performance of a state representation that follows an agent's action-, or vision-, paths. It is a state-space representation that has used nature as its motivation. In Section 2.1, we argued that predatory agents make decisions as a reaction to their immediate surroundings. In Section 3.2.3 we explored this idea and developed our own state-space representation that accomplishes the following in simulation:

1. Can track a dynamic target
2. Independent of number of agents
3. Independent of world size
4. Transferable to different environment types
5. Robust to noise

The Condensed Observation of Locale and Agents (COLA) representation observes the surrounding areas and condenses the agent presence into values that occur down each of its actions. Through a simple simulation containing one predator agent and one prey agent, we have shown that a COLA state representation can continuously track a moving target while a Position- and Relative-state representation have a great deal of difficulty with dynamic scenarios (either with prey, environment, or both). We have also shown that the COLA-representation has the inherent ability to transfer what it has learned to numerous domains such as larger gridworlds, dynamic gridworlds, and noisy gridworlds. Finally, in Section 7.2, we presented two scenarios in which an adaptive algorithm performs over 100% better than a hand-coded policy privy to the same state information.

The work presented in this paper suggests the COLA Learner is a promising platform from which to approach these more complex and demanding Predator-Prey problems. In multi-robot coordination scenarios where environments change and the number of agents may vary, it is necessary to enable these agents with enough versatility to remain functional. We have proposed that problems due to scaling may be better solved by approaching them from a state-representation rather than learning. This paper has shown that given the same learning algorithm, performance can be quite different depending upon what information is included in an agent's state.

Future work will address scenarios containing multiple predators hunting a single prey that is capable of evading. It will also introduce the need for allocating agent rewards to promote coordinated behavior. This work can also be extended

to the continuous domain with limited observability in which case agents would have to couple their COLA learning with search and communication algorithms.

Bibliography

[1] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized Pursuit-Evasion in Graphs," *Combinatorics, Probability and Computing,* Vol. 12(3), pp. 225–244, 2003.

[2] J. Bang-Jensen, G. Gutin, and A. Yeo, "When the Greedy Algorithm Fails," *Discrete Optimization,* Vol. 1(2), pp. 121–127, 2004.

[3] A.G. Barto and S. Mahadevan, "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dynamic Systems,* Vol. 14(4), pp. 341–379, October 2003.

[4] A.G. Barto and R.S. Sutton, *Reinforcement Learning: An Introduction*, MIT Press Cambridge, MA, USA, 1998.

[5] A. Bazzan, "Opportunities for Multiagent Systems and Multiagent Reinforcement Learning in Traffic Control," *Journal of Autonomous Agents and Multi-Agent Systems,* Vol. 18(3), pp. 342–375, 2009.

[6] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ, USA, 1962.

[7] M. Benda, V. Jagannathan, and R. Dodhiawala, "On optimal Cooperation of Knowledge Sources - An Empirical Investigation," Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, 1986.

[8] G. Boone, "Concept Features in *Re:Agent*, an Intelligent Email Agent," In *Proceedings of the Second International Conference on Autonomous Agents*, pp. 141–148, 1998.

[9] B. Bouzy and M. Métivier, "Multi-Agent Model-Based Reinforcement Learning Experiments in the Pursuit Evasion Game," CRIP5 Université Paris Descartes, Paris, France, 2008.

[10] Y. Chen, H. Qi, and S. Wang, "Multi-Agent Pursuit-Evasion Algorithm Based on Contract Net Interaction Protocol," In *Advances in Natural Computation*, pp. 482–489, 2005.

[11] P. Cheng, "A Short Survey on Pursuit-Evasion Games," University of Illinois, Urbana-Champaign, IL, 2003.

[12] V. Chvatal, "A Combinatorial Theorem in Plane Geometry," *Journal of Computorial Theory,* Vol. 18, pp. 39–41, 1975.

[13] J. Dean, J. Macker, and W. Chao, "A Study of Multiagent System Operation within Dynamic AD HOC Networks," In *IEEE Military Communications Conference*, pp. 1–7, 2008.

[14] K. Dresner and P. Stone, "Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism," In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Vol. 2, pp. 530–537, New York, NY, 2004.

[15] K. Dresner and P. Stone, "Mitigating Catastrophic Failure at Intersections of Autonomous Vehicles," In *AAMAS Workshop on Agents in Traffic and Transportation*, pp. 78–85, Estoril, Portugal, May 2008.

[16] A. Farinelli, L. Iocchi, and D. Nardi, "Multi-Robot Systems: A Classification Focused on Coordination," *IEEE Transactions on System Man and Cybernetics,* Vol. part B, pp. 2015–2028, 2004.

[17] F. Fernández, D. Borrajo, and L. E. Parker, "A Reinforcement Learning Algorithm in Cooperative Multi-Robot Domains," *Journal of Intelligent and Robotic Systems,* Vol. 43, pp. 161–174, 2005.

[18] B. Gerkey, S. Thrun, and G. Gordon, "Visibility-Based Pursuit-Evasion with Limited Field of View," In *Proceedings of the National Conference on Artificial Intelligence*, pp. 20–27, San Jose, CA, USA, July 2004.

[19] G. Hollinger, J. Djugash, and S. Singh, "Coordinated Search in Cluttered Environments Using Range from Multiple Robots," *Field and Service Robotics,* Vol. 42(3), pp. 433–442, 2008.

[20] G. Hollinger, A. Kehagias, and S. Singh, "Probabilistic Strategies for Pursuit in Cluttered Environments with Multiple Robots," In *Proceedings of the IEEE*

International Conference on Robotics and Automation, pp. 3870–3876, Rome, Italy, 2007.

[21] N. K. Jong, T. Hester, and P. Stone, "The Utility of Temporal Abstraction in Reinforcement Learning," In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Vol. 1, pp. 299–306, Estoril, Portugal, May 2008.

[22] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research,* Vol. 4, pp. 237–285, 1996.

[23] J.-H. Kim, H.-S. Shim, H.-S. Kim, M.-J. Jung, I.-H. Choi, and K.-O. Kim, "A Cooperative Multi-Agent System and Its Real Time Application to Robot Soccer," In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pp. 638–643, Minneapolis, MN, USA, 1996.

[24] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawai, and H. Matsubara, "RoboCup: A Challenge Problem for AI and Robotics," *Lecture Notes in Computer Science,* Vol. 1395, pp. 1–19, 1998.

[25] M. Knudson, "Neuro-Evolutionary Navigation for Resource-Limited Mobile Robots," Doctoral Dissertation, Oregon State University, Corvallis, OR, 2008.

[26] B. Lenzitti, D. Tegolo, and C. Valenti, "Prey-Predator Strategies in a Multiagent System," In *Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception*, pp. 184–189, 2005.

[27] J. Liu, S. Liu, H. Wu, and Y. Zhang, "A Pursuit-Evasion Algorithm Based on Hierarchical Reinforcement Learning," *International Conference on Measuring Technology and Mechatronics Automation,* Vol. 2, pp. 482–486, 2009.

[28] J. G. March, "Exploration and Exploitation in Organizational Learning," *Organization Science,* Vol. 2(1), pp. 71–87, 1991.

[29] "Massive Software," Autonomous agent-based, AI software used in motion picture and video game production, Massive Software website: www.massivesoftware.com.

[30] M.J. Mataric, "Coordination and Learning in Multi-Robot Systems," *IEEE Intelligent Systems,* Vol. 13(2), pp. 6–8, 1998.

[31] L. Panait and S. Luke, "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems,* Vol. 11(3), pp. 387–434, 2005.

[32] H. Van Dyke Parunak, S. Brueckner, and J. Sauter, "Digital Pheromones for Coordination of Unmanned Vehicles," In *Environments for Multi-Agent Systems*, pp. 246–263, 2005.

[33] D. Pellier and H. Fiorino, "Coordinated Exploration of Unknown Labyrinthine Environments Applied to the Pursuit Evasion Problem," In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 895–902, The Netherlands, 2005.

[34] W.B. Powell and B. Van Roy, "Approximate Dynamic Programming for High Dimensional Resource Allocation Problems," In A.G. Barto, J. Si, W.B. Powell, and D.W. II, editors, *Handbook of Learning and Approximate Dynamic Programming*, IEEE Press, New York, 2004.

[35] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley and Sons, Inc., 1994.

[36] B. Sachs and D. L. Souvaine, "An Efficient Algorithm for Guard Placement in Polygons with Holes," *Journal of Discrete and Computational Geometry,* Vol. 13(1), pp. 77–109, 1995.

[37] A. Sánchez and A. Weitzenfeld, "Multi-Agent Formations in a Herd of Wolves Hunting Model," Technical report, Instituto Tecnológico Autónomo de México, Mexico City, Mexico 2004.

[38] J. Schrum, "Competition Between Reinforcement Learning Methods in a Predator-Prey GridWorld," Technical Report: AI08-9, University of Texas at Austin, Austin, TX, 2008.

[39] G. Settembre, P. Scerri, A. Farinelli, D. Nardi, and K. Sycara, "A Decentralized Approach to Cooperative Situation Assessment in Multi-Robot Systems," In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Vol. 1, pp. 31–38, Estoril, Portugal, May 2008.

[40] P. Stone and R.S. Sutton, "Scaling Reinforcement Learning toward RoboCup Soccer," , pp. 537–544, Williamstown, MA, USA, June 2001.

[41] P. Stone and M. Veloso, "Multiagent Systems: A Survey from a Machine Learning Perspective," *Autonomous Robots,* Vol. 8(3), pp. 345–383, 2000.

[42] T. Takahashi, S. Tadokoro, M. Ohta, and N. Ito, "Agent Based Approach in Disaster Rescue Simulation - From Test-Bed of Multiagent System to Practical Application," In *RoboCup 2001: Robot Soccer World Cup V*, pp. 63–74, 2002.

[43] M. Taylor, "Autonomous Inter-Task Transfer in Reinforcement Learning Domains," Doctoral Dissertation, University of Texas, Austin, TX, USA, 2008.

[44] Adil Timofeev, "Adaptive Routing and Multi-Agent Control for Information Flows in IP-Networks," *Information Theories and Applications,* Vol. 12, pp. 295–299, 2005.

[45] K. Tumer, "Designing Agent Utilities for Coordinated, Scalable and Robust Multi-Agent Systems," In P. Scerri, R. Mailler, and R. Vincent, editors, *Challenges in the Coordination of Large Scale Multiagent Systems*, pp. 173–188 Springer Science + Business Media, Inc., 2005.

[46] K. Tumer and A. Agogino, "Distributed Agent-Based Air Traffic Flow Management," In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 330–337, Honolulu, HI, 2007.

[47] K. Tumer and A. Agogino, "Analyzing and Visualizing Multi-Agent Rewards in Dynamic and Stochastic Domains," *Journal of Autonomous Agents and Multi-Agent Systems,* Vol. 17(2), pp. 320–338, 2008.

[48] K. Tumer and A. Agogino, "Learning Indirect Actions in Complex Domains: Action Suggestions for Air Traffic Control," *Journal of Advances in Complex Systems,* Vol. 12(4), pp. 493–512, 2009.

[49] P. Valckenaers, J. Sauter, C. Sierra, and J. Rodriguez-Aguilar, "Applications and Environments for Multi-Agent Systems," *Journal of Autonomous Agents and Multi-Agent Systems,* Vol. 14(1), pp. 61–85, 2006.

[50] M. Veloso and D. Vail, "Dynamic Multi-Robot Coordination," *Multi-Robot Systems: From Swarms to Intelligent Automata,* Vol. 2, pp. 87–100, 2003.

[51] R. Vidal, O. Shakernia, H. J. Kim, D. Shim, and S. Sastry, "Probabilistic Pursuit-Evasion Games: Theory, Implementation and Experimental Evaluation," *IEEE Transactions on Robotics and Automation,* Vol. 18(5), pp. 662–669, 2002.

[52] M. Vieira, R. Govindan, and G. Sukhatme, "Optimal Policy in Discrete Pursuit-Evasion Games," Technical Report 08-900, University of Southern California, CA, 2008.

[53] C.J.C.H. Watkins, "Learning from Delayed Rewards," Doctoral Dissertation, Cambridge University, Cambridge, England, 1989.

[54] E. Yang and D. Gu, "Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey," Technical Report CSM-404, Dept. of Computer Science, University of Essex, Colchester, Essex, UK, 2004.

[55] C. Yong and R. Miikkulainen, "Cooperative Coevolution of Multi-Agent Systems," Technical Report: AI01-287, University of Texas at Austin, Austin, TX, 2001.