Multi-Agent Cooperation using Trickle-Down Utility

Steven McLean

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2003

Abstract

Trickle-Down Utility, an original method for reward distribution in a multiagent reinforcement learning system is investigated. This method leaves it to the agents to find the optimal reward distribution between all the agents in the system, where agents are able to pass on a portion of any reward received to other agents in order to encourage mutually beneficial cooperative behavior. This is done by establishing benevolence levels between agents which dictate the amount of reward an agent will give to another, and this level is capable of being changed dynamically by the agent in hopes of selfishly optimizing its individual reward. This system is useful in designing artificially intelligent computer game agents that will interact with other agents and human players in a realistic manner to give the illusion of emulating a real world society of agents, as it is done using the Torque Game Engine in this research. It is found that selfish agents can perform better and receive more individual long-term reward by sharing immediate reward with other agents. Homogeneous agents benefit from being fully benevolent, sharing reward equally. Whereas, heterogeneous agents might benefit most from being partially benevolent, though system performance may suffer as a result.

Acknowledgements

I would like to thank Gillian Hayes for her advice and guidance throughout this dissertation, which was essential in keeping me on track and with keeping the big picture in focus. I also must thank Jay Bradley for his insight into the potential of AI in computer games, and for his support and willingness to help through whatever practical issues arose.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Steven McLean*)

Table of Contents

Chapter 1

Introduction

This dissertation investigates an original method for reward distribution in a multiagent reinforcement learning system, called Trickle-Down Utility, which is designed with the intention of allowing agents with local perspective to learn to share reward to further benefit individual interests. This new multiagent system (MAS) utility function will be applied towards creating realistic artificial intelligence in a computer game environment. This method allows agents to redistribute reward amongst themselves in order to encourage cooperation towards accomplishing tasks.

Computer games are designed with the sole purpose of entertaining the player, which includes designing artificially intelligent agents that interact with the player, other agents, and the world in a realistic manner. Realism is never achieved by giving agents omniscient knowledge of the system, but rather by designing them to sense, think, and act like human players. Trickle-Down Utility is designed to allow agents to always interact with other agents in a manner that is always in their own best interests.

This research was done using the Torque Game Engine [20] to create game agents to interact with one another inside of a virtual world. The agents use reinforcement learning to choose actions, using reward received for completing certain tasks within this virtual world. Each agent maintains benevolence levels towards each of the other agents, which determines how much reward to share with the other agents whenever it is received for accomplishing desired tasks. In this research, the performance of the agents under various levels of benevolence is compared to visualize the expected utility landscape across all benevolence levels. Then the feasibility of making this level

variable is investigated, in order to allow agents to dynamically determine their own level of benevolence towards other agents.

The results from these experiments show that agents perform better and receive more cumulative reward by sharing some amount of reward with other agents. When all agents are equally capable of receiving the reward, it is shown that the entire group of agents benefit from being entirely benevolent with one another. When one agent is superiorly capable of receiving the reward, it is shown that this agent benefits the most by sharing a small portion of any reward. By giving away some of its immediate reward, an agent can ensure that it receives more cumulative reward in the long run. Making the benevolence levels variable has not been implemented completely, but suggestions are made for future effort towards this goal.

This chapter is intended to give an introduction to the rest of this dissertation, which is laid out as follows. Chapter 2 overviews some background material and relevant previous literature on this topic. Chapter 3 outlines the new form of reward distribution called Trickle-Down Utility, and outlines a number of questions to be investigated in this research. Chapter 4 describes the chosen task and how it is to be implemented. Chapter 5 discusses the experiments and results found using this implementation. Chapter 6 introduces a revised task and implementation and examines the new results. Chapter 7 examines the results as to how it applies to the investigation questions and the significance of the research, and outlines some future directions to be taken. Chapter 8 concludes this dissertation by summarizing what we set out to do, how we did it, and what we found.

Chapter 2

Background & Literature Review

In this section, we will outline some useful background information and overview some of the most relevant papers towards this topic. First, we will discuss the current state of artificial intelligence in current computer games and the directions in which it could be beneficial for us to further research, with much emphasis on papers from John E. Laird and Michael van Lent. Next, we will do a quick overview of some of the relevant issues in reinforcement learning necessary to understand all of the techniques used later in this paper, with particular focus on the work of Richard S. Sutton and Andrew G. Barto. Then, we will cover some of the major aspects of designing multiagent systems, much of which is well covered in the work of Peter Stone and Manuela Veloso. Finally, we will discuss a couple of different methods of reward distribution within multiagent systems, which differ in the overall complexity of the system and the side-effects of using different utility functions, as described well in the work of David Wolpert and Kagan Tumer.

2.1 Computer Games

2.1.1 What AI can bring to Computer Games

As the computer games industry continues to grow, the focus of game developers, and buyers' interest, is going to shift from graphics-intense games to games that offer superior gameplay and realistic AI. The popularity of some online computer

games such as *Half-Life:Counter-Strike* [22], a first-person shooter where dozens of human players join into teams and try to defeat the other team in such missions as hostage-rescue, bombing a target, or simply neutralizing all members of the other team, suggests that avid gamers may be growing bored of current AI in games, because they can easily be defeated and aren't dynamic enough to keep the game interesting. These sorts of multiplayer online games, which include role-playing games such as *Everquest* [16] and *Ultima Online* [14], strategy games such as *Starcraft* [2], and action games such as *Quake 3 Arena* [7], are rich in interesting agent interactions, because the other characters that are encountered in the game are controlled by other human players. If an AI system were able to emulate a human player in all of the ways necessary to keep a human player interested in the game, then the door is open to enhancing current games and to creating new and interesting games not requiring the networked connection to hundreds of other players.

There is a wide-range of roles an AI agent can provide to improve the gameplay. In the case of multiplayer online games, AI agents could be a substitute for human-controlled agents when necessary or they could be used to flesh out the world by assuming the roles of characters that human-controlled agents would have little desire to play, such as the storekeeper, the peasant farmer, or the fair maiden trapped in the tower. Ideally, we would like it if AI agents were able to adapt their goals and behaviors based upon the actions of the human players, allowing for dynamic storylines that will continue to intrigue and entertain the players. In action or adventure games, AI agents are very prevalent as enemies to be fought, or perhaps as partners and companions to aid the player. AI agents also play a big role in strategy and sports games, where they can be utilized to control the opposing team or army playing against the player or can be used to control the other members of the player's own team.

Some games, such as *Black & White* [10], are already being marketed with AI as a primary selling point for the game, indicating that some companies are actively trying to incorporate established AI techniques. As computer hardware systems and video game consoles become faster and more capable, more system resources will become available for the AI systems, allowing more realistic agents and bringing us closer and closer to human-level AI.

2.1.2 What Computer Games can bring to AI

We can define human-level AI as a robot or computer system that can think and interact with the world around it with the same capabilities as humans, such as the fictional AI systems like the Terminator, C3P0, or HAL, which we have all grown up expecting to be possible to create in the not so distant future. But so far, we do not have such systems, because currently there is no need for a human-level AI systems to solve most problems. The vast majority of AI research is focusing on very specific issues and techniques. Computer games might be the first application to significantly benefit from human-level AI, where artificial agents need to interact with a complex world and with human players in a realistic and engaging way. Laird and Lent claim that computer games "can provide the environments for research on the right kinds of problems that lead to the type of incremental and integrative research needed to achieve human-level AI" [9].

Brooks believes that human-level AI will be achieved in the future only after we develop insect-level AI and incrementally increase the system complexity up to human-level AI using the subsumption architecture [3]. Through his behavior-based robotics approach, Brooks is striving to achieve this goal, although human-level AI robots still look to be a long way off. Robotic systems are plagued by the fact that they must exist and interact in the real world, making it difficult to create realistic and reliable behavior in such an unpredicatable and complex environment. Computer games, however, allow agents to exist in an artificial world with more predictable physics and environment parameters.

With the gaming industry recognizing the need for more research in AI, and with academia gradually recognizing games as a promising platform on which to conduct original research, Artificial Intelligence research is at the stage where both institutions would benefit greatly from more collaboration.

2.2 Reinforcement Learning

One of the most promising research areas in artificial intelligence today is that of reinforcement learning, which is discussed much more comprehensively in Sutton

& Barto's book [19] and the algorithm can be seen in Section 4.2. "Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal" [19]. Given a current state space and a set of goals, the learner chooses an action based upon that state and observes the resulting state and the benefits of that action in helping it achieve its goals. A numerical value is associated with the states and/or the actions so that the learner is able to evaluate which action to choose when any particular state is encountered in order to maximize reward.

Reinforcement learning lends itself wonderfully to agent-oriented systems, where agents "have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments" [19]. The virtual agents in virtual worlds found in computer games are no different than robotic agents when it comes to programming reinforcement learning techniques, which means reinforcement learning research can be conducted using games, but with the advantage that the algorithms can be run much faster than real-time robotic systems. The primary reason that reinforcement learning techniques have not been implemented in game agents in the past is due to the computational demands of the learning, forcing the state space to be fairly small. But, as more resources become available, there will be many demands for reinforcement learning techniques, because they allow for the game agents to adapt to the style of play of the players, providing a more entertaining experience.

The implementation described in this paper uses Q-Learning, a control algorithm developed by Chris Watkins, where agents will learn the optimal policy while performing actions from a different policy [23]. In each state, an agent can choose from one of it possible actions, and each state-action pair is assigned a Q-value which represents the expected reward the agent will receive if it chooses that action while in that state. Over time the learner will converge to the optimum Q-values, allowing the agent to follow the optimal policy to receive the most reward.

One of the most popular policies to follow when using Q-Learning is the ε-greedy action selection method. Here, the agent will usually select the best action available in a given state, but some percentage ε of the time, it will select a random action. This allows the agent to fully explore the state space in order to accurately find the corresponding Q-values for each state-action pair. By decaying the ε gradually, the

agent will exploit its learnt knowledge once sufficient exploration has been done in order to maximize the amount of reward it will receive in the long run.

2.3 Multiagent Systems

Multiagent Systems (MAS) can be defined as "the subfield of AI that aims to provide both principles for construction of complex systems involving multiple agents and mechanisms for coordination of independent agents' behaviors" [18]. Some tasks require distributed agents, such as multi-robot systems, while some tasks that can be implemented using a single-agent system, might prove simpler or benefit from improved performance when implemented using a distributed agents approach.

Peter Stone outlines some of the key advantages of using MAS over a single-agent system, such as parallelism, robustness, scalability, and simpler programming [18]. Parallel computation with multiple agents can improve a system's performance by allowing separate agents, with separate computation processing power, to perform certain subtasks of the overall task, and together form a complete system. Alternatively, a robust system has individual agents performing the same overall task, and the system will continue functioning even if there are failures in some of the redundant agents. MAS scale much easier than one large system, because the agents are designed to interact with other agents, and so it shouldn't involve much work to add additional agents. Also, MAS should be much simpler to program than one large system because of the inherent modularity of the individual agents. Each agent can be individually programmed for its role in the system

One of the most major factors to consider when designing a MAS is the amount of knowledge each agent has about the system. The designer may need to decide whether to give agents a local or global perspective. Sometimes having a local neighborhood perspective can be beneficial for the system to allow it to take advantage of the distributed nature of the agents. In some situations, such as a group of agents seeking out under-utilized resources, it may be counter-productive for the agents to have global knowledge, because they will all continue to choose the same resource, instead of choosing resources in their respective vicinities, which could greatly benefit system

performance [15].

In addition, the designer needs to consider how much knowledge an agent should have about the other agents in the system and how they might model their states. The defining nature of a MAS is that an agent's environment is dynamic because it is affected not only by its own actions, but also by the actions of the other agents. A single-agent system benefits from the fact that it knows what all elements of the system are to do at any given time, but in order for a MAS to be as effective, an agent would need to know what the other agents were going to do. To do this, an agent tries to predict the other agents' actions based on previous experience and modelling of their states.

The three most defining categories of a MAS are into heterogeneity, communication, and benevolence.

2.3.1 Homogeneous vs Heterogeneous

Homogeneous agents are a group of agents with the same state space knowledge, goals, and set of actions. Homogeneous agents are useful when robustness is needed in a system, since all the agents are the same, so no one agent is pivotally important to the system, plus extra agents can be added with minimal extra programming and work. Homogeneous agents might be especially useful for foraging tasks, where the agents need to spread out to explore and gather any necessary resources found.

Heterogeneous agents are a group of agents that might have different state space representations, allowing for more or less knowledge of the environment; differest sets of actions so that one agent might be more capable of accomplishing certain tasks better than other agents; or different goals, possibly inducing agents to work at cross-purposes, where they might impede each other's progress to achieving their respective goals. Heterogeneous agents are quite useful for tasks that can be divided into subtasks and the agents can specialize in one of the subtasks so as to improve the performance of the entire system.

An example, for comparison, might be a task where a group of agents need to move a set of objects, such as crates, from point A to point B, one side of the river to the other side of the river, as quickly as possible. A system of homogeneous agents

might involve each agent picking up a crate and fording the river with it, then going back and getting the next one. A system of heterogeneous agents might involve agents that are extra capable of moving the crates, such as a crane, and agents that are extra capable of crossing the river, such as a boat. Here, the cranes could easily load the crates onto the boats, the boats cross the river fairly quickly, then more cranes take off the crates, and the boats return for another load. In this situation, it is more than likely that the system of heterogeneous agents would perform far better overall and far more efficiently from an individual agents perspective.

In general, systems of heterogeneous agents might be more complex to design and more prone to problems if one agent breaks, but could offer better performance and are more useful for studying social agents.

2.3.2 Communicating vs Non-Communicating

Communicating agents have the ability to share information between one another in a direct way. They are able to share their sensed state, allowing for distributed sensing, where an agent can learn more about the environment beyond that which its own sensors can detect. Or agents are able to share their individual goals or share what they are going to do next, in order to prevent agents from working at cross-purposes or trying to accomplish the same task, when it may only require one agent's effort. With communication, agents are capable of planning group maneuvers that might be otherwise impossible or require lots of training and modelling within the group.

Non-communicating agents can effectively work together through the effects of stigmergy, where an agent can affect other agents by affecting the environment. Owen Holland designed a team of non- communicating robots to mimic a colony of ants moving dead ants into a pile [5]. The robots were reactive robots that simply moved in a straight line, turning to avoid walls, and occasionally picking up one of the pucks that had been scattered around the arena. When it picked up three pucks, it would backup and turn slightly and continue moving. The pucks would start to be put into small piles and then gradually all shifted into one large pile. Although the agents had no knowledge of this goal and were not working together explicitly, the desired result was achieved by the system.

In a self-contained computer game like the one being used in this dissertation, it is easy to allow agents to communicate since the system of agents is contained within a higher level system, as opposed to a robotic MAS or a networked computer game, where communication is a much bigger concern. Communication is likely necessary or essential for redistributing reward as it is done in this research and described later in this paper.

2.3.3 Benevolence vs Competitiveness

All research in MAS is essentially studying the interaction of agents, and one of the most interesting aspects of this is how agents view and treat other agents. The biggest distinction between types of agents in this aspect is whether they are benevolent or competitive towards one another. A benevolent agent chooses actions which may benefit other agents more than itself, under the premise that the other agents will act in a similar manner towards it. A competitive agent chooses actions which will fulfill its own goals, and may purposely choose actions to impede other agents from fulfilling their goals.

Many researchers feel that all agents in a MAS should be exclusively self-interested. A selfish agent chooses actions which will fulfill its own goals, though we can build into selfish agents benevolence or competitiveness by further specifying what their goals are, in that selfish agents will work together only if they share a common goal.

A popular example to illustrate the benefits of being competitive or cooperative is the Prisoner's Dilemma Problem. Two agents are in prison and are given the opportunity to confess against the other agent. If one agent confesses and the other does not, then the confessing agent will receive one year in prison while the other receives four years. If both confess, they will both receive three years, and if neither confesses, then they will both receive two years in prison. From an individual rational agent's perspective, it is better to confess, because if the other agent confesses, you get three years in prison as opposed to the four years if you don't confess, and likewise, if the other agent does not confess, then you will only get one year in prison.

However, what isn't taken into account is the fact that the other agent is also a

rational agent and will likely come up with this same strategy, meaning both agents will end up confessing and both will receive three years in prison. Considering the total number of years that will be spent in prison, it is actually best for the system if both agents cooperate by not confessing so they both receive two years, for a combined total of four years, as opposed to the six years if they both confess against one another. But how are agents supposed to come up with this strategy? When iterating this problem repeatedly with the same agents, such as in a genetic algorithms approach, after many episodes or generations eventually cooperative behavior does emerge [12].

Game theory provides other approaches towards optimizing the global utility of the system, while optimizing the individual utility of the agents. Some utilize Nash Equilibrium, where the agents are still selfish, but take into account the likely actions of the other agents in the system in order to find an equilibrium where all agents are as satisfied as possible [6] [11] [13].

2.4 Utility Functions

2.4.1 Team-Game Utility

Team Game Utility (TGU) is definitive of a team of benevolent agents. Whatever utility the team receives gets distributed evenly to the entire team, where utility is the another term for reward. This directly aligns the utility function of individual agents and of the group. We will be using TGU as a basis for comparison against the results found from other experiments later in this paper.

The advantage of TGU is that the agents truly are acting in the best interests of the team, because individual reward is directly proportional to team reward. Agents in the team never purposely work against one another, since there is no competitive aspect within the team. Also, TGU is computationally simple and easy to implement.

The disadvantage of TGU is that it requires a well defined notion of the team in the system. This requires prior knowledge of the members and the system must be designed with the team in mind, but in certain situations this might be difficult or impossible. Also, within the team, TGU encourages bad behavior in agents which are not directly involved in completing the group task. In the example of the football team,

a midfielder might, by chance through an exploratory action, run to the sideline to get a drink of water while at the same time the team scores a goal. Should he be rewarded for his actions? The next time the team looks like it might be getting close to scoring he might all of a sudden run to the sideline to get that drink of water, and then the opposing team might steal the ball and the midfielder will be out of position.

2.4.2 Wonderful Life Utility

One alternative approach uses the "COllective INtelligence" (COIN) framework, where COIN is a "large multi-agent system where there is a well-defined 'world utility' function which rates the possible dynamic histories of the collection and where there is little to no centralized control" [21]. This framework focuses on making the individual utility functions aligned with the global utility function, so as to eliminate the agents from working at cross-purposes. The utility function that is used is the Wonderful Life Utility, where an agent determines its utility based upon its worth in the system. Essentially, in the spirit of the film, each agent receives reward for what it accomplished which wouldn't have been accomplished without his presence.

The advantages of this method are that it solves the problem of encouraging bad behavior seen in the TGU, and if agents have global knowledge of the system their actions will work towards the benefit of the team, even though they are selfish agents.

The disadvantages of this method is acquiring this global knowledge. To calculate an individual's utility, it calculates what the global utility would be if all the other agents did their same actions but the individual was not present. The system's performance improves as more knowledge of the system is included in each agent, with optimal performance occurring only with every agent having global knowledge of the system. This makes us consider how difficult is it to get this much knowledge, or what might happen if this information were unavailable or impossible to get. This approach is also computationally complex and so no effort was put forth to implement it in this dissertation project.

Chapter 3

Trickle-Down Utility

I propose a MAS utility function called Trickle-Down Utility, that is designed in the spirit of human-level AI. The system is comprised of selfish agents, but each agent has the ability to share utility with others, in order to further benefit its own best interests. There is no well defined notion of team, and an agent has the ability to dynamically determine its own benevolence or competitiveness towards other agents. Humans reguarly, if not always, have in mind their own best interests, so designing a human-level AI MAS should take this into account.

As mentioned before, selfish agents can be made to be benevolent by sharing common goals. By using Reinforcement Learning, goals are simply the source of reward, and agents will learn to choose actions that maximize their reward over time. An agent's goals become synonymous with rewarding the actions that achieved the goal. So perhaps a better method than sharing goals amongst selfish agents might be to encourage them to become benevolent by getting them to share reward amongst the other selfish agents. By receiving reward when other agents accomplish their goals, new goals indirectly emerge, such that an agent learns to assist the other agents in order to receive more personal reward. Through this, agents might form into mutually beneficial alliances, which we can refer to as a team, that will not only maximize individual utility, but also system utility.

Thinking in terms of a multiplayer role-playing game, this is a more realistic system to use than Team-Game Utility or Wonderful-Life Utility, because player characters and game-controlled charaters have the option of working with a group of other agents

or working alone to complete tasks. By sticking together they are more likely to successfully complete tasks, but will receive less reward for completing the task than if they completed it alone. Perhaps, though, as a group it becomes possible to complete tasks which are far too difficult to complete individually and thus each agent will receive more reward as part of the group than if they each worked alone.

Team formation using this method is desirable, especially when we want the system to emulate tasks where teams are distinctive elements of the task. A preliminary example of this could be two footballers both wanting to move the ball up the pitch and get it into the opponent's goal. If the midfielder passes the ball to the forward, the team would have a better chance of scoring than if he tried to dribble it all the way up himself. But how will this bring him any personal reward? If the forward were to give him credit for the pass after him scoring the goal, then that could be how he gets his reward. If the forward does not 'thank' the midfielder, then the midfielder will receive no reward for that action and will be less likely to pass the ball again to that forward, thus hurting the team's chances of scoring. By the forward learning to share some of its reward it is not only benefitting the team, but also itself because it is ensuring that it will continue to get the ball in the future and be given opportunities to score more goals.

Thinking in terms of an economic system, we could say that agents are always trying to maximize the amount of currency in their own pocket, so they will only cooperate under the assumption that they will get paid in return for their help. And they are willing to pay other agents that help the agent to accomplish his task. This might be a great method for establishing subtask allocations. Given a group of agents and an episodic ongoing task that requires many subtasks to be completed, such as building houses, certain agents could gradually prove themselves to be better carpenters, plumbers, or electricians. These agents could then gradually be given more and more reward if the house gets built faster, until they have established themselves as the best at their craft. The system, or more specifically, the agents in the system, will balance overall performance with optimal reward distribution to ensure that each agent gets as much reward as it deserves.

3.1 Benevolence Levels

In order to accomplish this, we need to establish a numerical value indicating how much reward should be given to other agents in order to maximize individual utility over time. We establish this as benevolence levels, a sliding bar between zero and one, where zero indicates purely competitive agents, where no reward is passed, and one indicates purely benevolent agents, where reward is shared evenly.

The grey area in between will lead to complex agent relationships that may not fall into the category of a strict team-structured situation. It will be interesting to investigate this area as to how well the group performs, and how well each agent performs individually.

One advantage of this approach is that it is possible for an agent to only include the agents in its immediate proximity when considering them for reward sharing. This could be extremely beneficial in cases where omniscient knowledge of the entire system and the other agents is not necessary, or impossible to get, such as in the case of a robotic MAS. An agent can simply keep a record of these benevolence levels for every other agent it has encountered and only update them when in contact with that agent.

Prior knowledge can be built into the system by initializing the benevolence levels appropriately, such as when teams are an inherent element of the task. Plus, we can decide whether the agent should be initially trustful, or distrustful, when encountering new agents.

3.2 Investigation Questions

We now want to outline the investigation questions for the rest of this research and paper.

3.2.1 Question 1: Investigate Static Trickle

The first thing to investigate is the difference in performance based upon what the benevolence levels are set to. We would like to investigate whether it is possible to

predict what the expected utility landscape will look like for individual agents and for the system. Here are a few questions to guide us in our research.

- How does the performance of Full Trickle compare to Team-Game Utility?
  Where Full Trickle uses fully benevolent agents. See Section 5.1 for some results.
- In what kind of task is the optimal solution going to be for agents to be entirely benevolent, entirely competitive, and partially benevolent to some agents?
- How will the results vary between homogeneous agents and heterogeneous agents?
  See Section 6.2 for some results.

3.2.2 Question 2: Investigate Variable Trickle

The next topic to investigate is the feasibility of making the benevolence levels variable, so that the agent can alter its interaction with others in order to improve overall performance and personal reward. The dynamics of the system are quite unclear as to how easily implemented this might be, but here are a few questions we hope to answer.

- What is the best technique to use to find the optimal benevolence level?
  See Sections 5.3 and 6.3 for some discussion.
- How quickly can the benevolence level converge to the optimal value?

3.2.3 Question 3: Investigate Agent Interactions

Here we hope to take a look at our agents as a model for the social world, and that we might learn something from their interactions.

- How are the benevolence levels of one agent effected by the benevolence levels of other agents?
- In the case of heterogeneous agents, will more capable agents become more competitive than the others?

Chapter 4

Implementation

4.1 Task

The task chosen to conduct this series of investigations into using Trickle Utility, is for a group of three agents, game bots named Andy, Bob, and Chuck, to learn to go to a randomly chosen goal, a predefined location within a virtual world. There is a negative reward for every step that the agent takes until it has reached the goal, and once near the goal they will receive a reward of zero. With these rewards, the agents will use Q-Learning reinforcement to find the optimal policy as to how to get to the goal along the shortest possible path in order to minimize negative reward. Once the last agent reaches the goal, a large positive episodic reward is given to the team using either the team-game utility or Trickle methods, and the goal randomly switches to another goal and learning continues for 1000 episodes.

When using the Trickle method, the last agent receives all of the reward and is capable of redistributing that reward amongst the other agents as he sees fit. What we hope to find is that under certain circumstances, agents will learn a policy that goes against the optimal individual policy of getting to the goal as quickly as possible. Instead, this policy will keep agents out of the goal, willingly accepting additional negative reward, in order to try to be the last agent to enter the goal to receive the large episodic reward. As agents' policies become more dependent upon this episodic reward, the overall utility of the system will decrease as the performance of the group

of agents moving towards the goal becomes suboptimal. We hope to show that in order to maximize system utility and individual utility, agents will need to learn to become benevolent among the other agents.

4.2 Q-Learning

4.2.1 State Space

The state space was set up with the intention of having the location of objects in the environment spatially relative to the agent. The state space should not include any absolute coordinates, because thinking in terms of human-level intelligence, this is not the most realistic representation we desire. The inspiration for this came from Peter Stone's RoboCup work [17].

4.2.1.1 SubStates

The state vector for an agent is comprised of a set of substate vectors, each of which corresponds to a specific object in the environment. The first substate vector, in all of my experiments, referred to the current goal, and the other two substate vectors, if present, referred to the other two agents. The substate for the goal contained the distance and direction from the agent to the goal. The substates containing the other agent information consisted of the distance of that agent from their own goal, which in all of my experiments corresponded to the same goal, but this was done so that agents could be assigned separate goals in the future. The direction from the other agents to the goal is irrelevant information for the task introduced in Section 4.1, and so was not included, but can be added if necessary.

Separating the state space vector into substate vectors proved useful when allowing or disallowing agents access to the knowledge of other agents simply by providing the corresponding substates when needed. In general, the fewer substates given to the agent, then the smaller the state space would ultimately grow to. As a point of comparison, in my tests, when an agent did not have knowledge of the other agents, the saved Q-values file tended to be about 9MB, and when the agent did have knowledge

of the other agents, the Q-values file tended to grow as large as 400MB. Obviously, computation time might become an issue as the state grows in size.

4.2.1.2 Proximity

To find the proximity used in the substate vectors for the goal and the agents, the program finds the distance between the three-dimensional coordinates of the agent and goal, which are easily accessible within the game engine. Then it classifies the distance into the proximity categories as listed in Figure 4.1 below, where the units are in meters.

Proximity	Distance (m)
Next	0.0 - 2.0
Near	2.0 - 15.0
Far1	15.0 - 30.0
Far2	30.0 - 75.0
Away	75.0 - 150.0
Gone	150.0+

Fig. 4.1: Proximity Categories

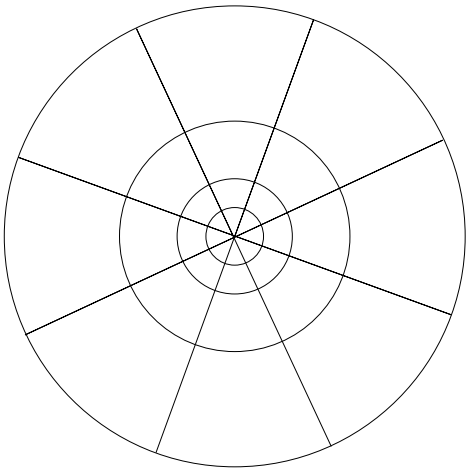

Fig. 4.2: State Space Coverage

These proximity categories are used instead of the actual distances in order to make the state space as small as possible because of memory and computational issues. These values could easily be changed by altering the distances between the different proximities, or by adding or deleting additional categories. The intention was that the inner rings should be closer together and then the space between them gradually increase as they get further from the goal.

The Next proximity was intended to be used for tasks where agents needed to physically interact, such as tagging or passing an object. However, for all the experiments conducted, I purposely left out the Next proximity so as to slightly reduce the size of the state space, and because I felt it wouldn't have too large an effect on the

learning. No tasks involving tagging or object passing were studied.

The Near proximity is used to indicate when an agent is at a goal. It was necessary to have an area of space to be considered the goal as opposed to a single point, because agents couldn't stop moving, as explained below, and ideally once in the goal an agent would learn to simply run around within this area without leaving.

4.2.1.3 Direction

Next, the angular bearing was found between the agent and the goal with zero facing east. This was likewise categorized into the eight basic directions: North, South, East, West, NorthEast, NorthWest, SouthEast, SouthWest. Again these values could easily be altered to allow for additional directions, if more information was needed, and a larger state space was computationally feasible and beneficial. The complete state space represents an area similar to that shown in Figure 4.2.

When including knowledge of other agents into the state space, the direction is not kept, because it doesn't necessarily provide useful information, but rather just increases the size of the state space. In the end, a typical state vector for Andy might look like: [Goal: Away, NE; Bob: Near; Chuck: Far1]

4.2.2 Actions

The possible actions an agent can take at a given moment are to move in the eight primary directions: North, South, East, West, NorthEast, NorthWest, SouthEast, and SouthWest. This is done by taking the bot's current location and adding a set distance to the x and/or y coordinate, then telling the bot that this is his destination. When moving diagonally, such as NorthWest, the x and y are both changed by a distance determined by the Pythagorean theorem, such that the diagonal distance is the set distance, instead of adding that set distance to both the x and y. The advantage of this is so that the agents are not stuck moving in a grid-like pattern and can add to the stochastic nature of the environment, making them more beneficial for use in actual game AI bots.

Initially, this distance was set at 10 meters, but it proved beneficial to reduce this to 2 meters for all of the experiments, allowing for the agent to more constantly check

its state and decide if it should choose a different action. The game engine takes responsibility for moving the agents once a destination is specified, and when an agent reaches its set distance destination, this is considered to be a completed action step, and the agent will choose another action.

4.2.2.1 Action Steps vs Learning Steps

Due to the nature of the state space, very often, especially when far away from the goal, an agent will discover after moving a step that his state is exactly the same because the distance he moved is relatively small compared to its distance from the goal and the proximity hasn't changed, for example. So, to improve learning and prevent an agent from running around randomly, after it completes its previous action, the agent checks if its state has changed and if not, then it chooses the same action until the state does change. Once the state does change, the Q-values are updated and a new action is selected. A learning step may consist of many action steps.

The benefit of this comes is in not skewing the Q-learning. If an agent were to choose the best action available to him and find himself in the same state, with the same negative reward, then the next time he picks the best action in the same state, the best action may have changed. The agent might just run in circles in the same state and not learn how to move out of this state effectively.

4.2.2.2 DoNothing Action

Originally, I intended to have an action called DoNothing, which essentially did just that, the agent would choose to remain idle where it is. This would be very helpful when the agent reaches the goal and decides to stop moving and continue receiving non-negative reward for its action. However, there were a number of flaws when I first tried testing with this action. While choosing the same action until the state changes, as described above, if an agent were to pick the DoNothing action, its state would never change, and thus would continue to choose the DoNothing action forever. While this could be fixed by checking for this special case, there continued to be problems when the agent did choose this action, because the agent would instantly complete this action, unlike the other actions which had a certain time involved when moving from

one location to another.

If an agent found the DoNothing action to be a good thing, such as when it was in the goal, it would continue to pick it over and over again, thus making the game slow down dramatically, because there was constant computation at each learning step reassuring the agent that the DoNothing action was beneficial, making it select it even more often. Some form of time delay is necessary in order to make this action useful. So for this experiment, I did not include this action, and just forced the agents to keep moving at all times.

4.2.2.3 Allowable Actions

When an agent's state indicates that it is Gone from the goal, then it is not allowed to pick actions that may take it further away from the goal. It must pick instead an action which will bring it closer to the goal, thus creating a boundary limit around the goal, preventing the agents from wandering off and not learning effectively. This also prevents the instance where the agent would choose to move away from the goal and then discover its state never changes, and thus continue running away from the goal forever.

So, if an agent's state was [Goal: Gone, SW], then it would only be allowed to move in the three directions of South, SouthWest, and West.

4.2.2.4 ε-Greedy Action Selection

As the agent is learning the optimal policy, the agent follows a policy determined using the ε-greedy action selection method. The agent will take the best action available, except it will choose a random action some percentage ε of the time. This parameter ε gradually decays as the actions are all fully explored while in each particular state. This forces the agent to begin to exploit the knowledge it has learned in order to receive the maximum amount of reward possible. A lower threshold limit of .001 is set upon this parameter to ensure that the agent does explore on rare occasions.

4.2.3 Rewards

After every action there is a -1 penalty, to encourage an agent to move to its goal as quickly as possible. If an agent's proximity is 'Near' to their goal, then there is no negative reward. If all three agents are 'Near' to their goals, then an episodic reward of +100 is given out. This reward is either split up and given to all three agents, or given entirely to the last agent that enters the goal. This is determined by the experiment and the form of utility distribution, which will be explained later in Chapter 5.

4.2.4 Q-values

In each state, an agent is capable of taking one of its possible actions, and each of these state-action pairs is associated with a numerical value called a Q-value, which indicates the amount of reward an agent can expect to receive by following that action while in that state and then continuing with the policy. An agent's policy is determined by taking the action with the highest Q-value in every state, and over time these Q-values should converge to the optimal policy, given a suitable learning algorithm.

This convergence is ensured by decaying the learning parameter α, which is stored within each Q-value. This parameter, α indicates how much the Q-value should be affected by the current reward, and how much it should be affected by the previous Q-value. After each learning step, this parameter is decremented slightly, with a lower threshold limit of .01.

These Q-values are updated according to the algorithm below, which corresponds to the case where agents are learning without knowledge of the other agents, and this learning is done by all three agents.

4.3 Redistributing Reward

Whenever an agent wins an episode and receives episodic reward, he is allowed to share some of the reward with the other two agents. The actual amount is determined by a set of benevolence levels, with each agent maintaining a level for each other agent it has encountered in the system. These levels can be fixed or can be made variable so

Algorithm 1 Q-Learning Outline - Learning Individually

s ε [Goal:(Next,Near,Far1,Far2,Away,Gone),(N,S,E,W,NE,NW,SE,SW)]
a ε [(moveNorth,moveSouth,moveEast,moveWest,moveNE,moveNW,moveSE,moveSW)]
reward ε [($-1.0 \equiv$ Not Near Goal),($0.0 \equiv$ Near Goal),($100.0 \equiv$ All Agents Near Goal)]
$\alpha(s,a) = 1.0$
$\gamma = 0.9$
$\varepsilon(s) = 0.5$
for episode = $1 \ldots 1000$ **do**
  Set Goal: $goal \leftarrow RandomGoal$
  while all agents are not Near or Next to $goal$ **do**
    Set State: $s \leftarrow currentState$
    Take Action: $a \leftarrow \varepsilon$-$GreedyAction$
    if $\varepsilon(s) > .001$ **then**
      $\varepsilon(s) \leftarrow \varepsilon(s) * .99$
    end if
    Observe: $reward, s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha * [reward + \gamma * optQ(s',a') - Q(s,a)]$
    if $\alpha(s,a) > .01$ **then**
      $\alpha(s,a) \leftarrow \alpha(s,a) * .99$
    end if
    $s \leftarrow s'$
  end while
end for

as to find an optimal level that maximizes reward. It includes itself in this list of agents, but simply considers its benevolence level towards itself to be 100% at all times.

For redistributing reward, an agent simply finds the percentage that each other agent should get by dividing the benevolence level for each corresponding agent by the total benevolence level of all known agents.

$$Percentage(A,B) = \frac{Benevolence(AgentA,AgentB)}{\sum_{X=1}^{n} Benevolence(AgentA,AgentX)}$$

where $Percentage(A,B)$ is the percentage of Agent A's reward that he intends to give to Agent B, and where:

$$\sum_{X=1}^{n} Percentage(A,X) = 100\%$$

This reward distributed from other agents is added into the reward an agent finds the next time they receive environmental reward.

4.4 Implementation Specifics

4.4.1 Game Engine

To implement these techniques and experiments, the Torque Game Engine [20] was chosen as a good testbed. Torque is the AAA rated game engine that formed the backbone to the Tribes engine, and is distributed by GarageGames with the intention of being used by independent game developers to produce original games.

The engine takes care of the interface, animation, and physics necessary for any game. Built-in world editors allow for the developer to quickly design virtual worlds, and GUI editors also allow for the user interface to be easily manipulated. Within the engine, it is simple enough to create bots, which can be controlled by the programmer, making it easy to implement bots with AI to choose their actions based on reinforcement learning. This programming can be done in scripts or in original C++ code that can be easily integrated with the existing engine code.

4.4.2 Running a test

Tests could be conducted in either game mode or in server mode. In game mode, the game loads creating a player character which can be controlled by the keyboard and mouse to move around and interact with the virtual world, and we are able to watch our agents moving within the virtual world and learning to choose the appropriate actions based on what rewards they receive. Commands to control the learning could be issued by typing them into a console window that is available upon request. We can set the game speed to be slightly faster than normal, but as the speed increases the system is unable to smoothly display the agents moving within the game.

Alternatively, we are capable of running the game in server mode, where everything is loaded except for the graphics engine to allow the user to watch what is happening in the virtual world. However, in server mode, the game speed can be set much higher because there are far fewer computational demands. For all of my experiments, I ran the game in server mode at 10,000 times the normal speed.

As the program runs, current data is outputted to files to be analyzed later. Every one hundred learning steps, each agents saves its current Q-values to a file. After each

episode completes, the system outputs the current goal and location, how many action and learning steps each agent took, the current cumulative reward totals of each agent, and which agent won the episode. This data is then analyzed using Matlab, to create the many graphs seen in the results.

4.4.3 Problems & Difficulties

4.4.3.1 Agent Collisions

Game engines are designed in a fashion to emulate the real physical world as much as possible, and that includes giving agents corporeal form. This means that agents will collide with objects in the environment and with each other. While this is ideally how computer games should be designed, this brought my implementation some difficulty, because agents often got stuck against things or one another. Some AI implementations take the effort to build into agents obstacle avoidance so that they do not collide with anything, but this seemed beyond the necessary scope of this implementation. So, all of the objects in the virtual world that could cause agents to get stuck, such as trees, rocks, and buildings were removed. Plus, the collision detection software was altered slightly to no longer detect collisions between agents, so they act like ghosts by being able to move straight through other agents.

4.4.3.2 Agents falling to Oblivion

One major setback, which delayed all data collection significantly, was a bug in the Torque Game Engine that would cause the vast majority of the test runs to crash. Sporadically during a test run, an agent would simply stop responding to all of the commands the program would give it. It turns out that when an agent stopped responding, it was because it had somehow fallen through the ground in the virtual world and was falling helplessly. This situation never happened when I would run the learning tests in game mode, but only in server mode, when the game at a greatly faster pace. This could suggest that either I never experienced it in game mode because I didn't give it enough time for this fluke to occur, or it is because the game engine couldn't handle the game speed running this quickly, while still checking for

all collisions and ensuring the physics of the game worked properly.

Not willing to do an extensive study of this problem or try to fix the bug, I simply set about a way to get around this problem. Whenever an agent is found to be falling, the program respawns that agent and restarts the current episode. All important data such as Q-values, states, goals, etc. are saved from the falling agent, who is then removed from the game, and a new agent is added to the game back at the initial spawnpoints with all the previous agent's data loaded into the new agent. All the learning counters are reset in all the agents, they are all moved back to the previous goal, the starting point of the current episode, and learning is resumed.

To detect the problem, the program looks at the coordinates of the agents whenever an agent saves its Q-values, which is after every one hundred learning steps. Currently, the only way for a trial to crash is when all three agents fall at about the same time since they are unable to finish any learning steps and check on the other agents. Though occasionally multiple agents do fall at times close to one another, very rarely do all three fall so this wasn't a great hindrance.

I feel that the validity of the data isn't compromised terribly by this process. On average, there are about 3 or 4 restarted episodes in every trial, so it would be much more counter-productive to just wait for the occasional trial with zero restarts. However, if errors occur in the first few episodes, I do throw away that data because it does seem to skew the learning curves since agents are doing the majority of their initial explorations during this time.

Chapter 5

Experiments & Results

To compare these different methods of reward distribution, several different experiments were set up to examine the performance of each system, first varying the type of reward distribution. We compare the TGU against the Trickle method, using both static and variable benevolence levels. The static levels chosen are 0% (No Trickle), 50% (Half Trickle), and 100% (Full Trickle).

Secondly we vary the amount of knowledge the agents have at their disposal about the other agents. We look at the cases where all three agents have no knowledge of each other, where one agent has knowledge of the other two, and where all three agents have knowledge of the others. The second case, where one agent has more knowledge than the other agents, is a case of using heterogeneous agents, which should provide some useful insights into how different agents might benefit differently from cooperating.

5.1 TGU vs Full Trickle

The first investigation compares the results of TGU against the Full Trickle method. In the TGU experiments, the episodic reward is split evenly among and distributed to all of the agents by the system, once all three agents are near the goal. In the Full Trickle experiments, the episodic reward is given to the last agent who reaches the goal, and he is forced to redistribute the reward evenly amongst all of the other agents. This is done by setting all benevolence levels at 100%.

In both cases, this creates an uncompetitive task, because an agent's personal utility is not dependent on the actions of the other agents. Once an agent reaches the goal, he begins receiving zero reward instead of negative reward, so it is in his best interest to move as quickly as possible to the goal. At the same time, he does not gain anything by being the last agent to reach the goal since he would be afflicted by more negative reward and gets the same amount of episodic reward as all the other agents.

What we hope to see is that the performance should be identical between these two cases. The only difference between the two cases is the source of the reward, whether it comes from the system or another agent. This is just a difference in implementation, and should have no effect on the results. And from the results of the testing, we find that this is indeed the case. In every test, the results were nearly identical, with the only variation being a result of the inherent randomness of the task. Below are shown representative graphs showing these similar results.

5.1.1 TGU vs Full Trickle - Learning Curves

The learning curve in the case where all three agents have knowledge about the other agents using the TGU distribution can be seen in Figures 5.1 and 5.2. The learning curve in the case where all three agents have knowledge about the other agents using the Full Trickle distribution can be seen in Figures 5.3 and 5.4. The x-axis of the learning curves corresponds to the episode number from one up to one thousand. The y-axis refers to the relative directness with which an agent moves towards the goal, which is found by dividing the number of action steps to reach the current goal by the distance between the previous goal and the current goal.

These learning curves are typical of the learning curves presented in the rest of this dissertation. When one curve is shown, it is the average of the three curves from each agent, otherwise the three curves are shown. The learning curves generally start out very high as the first few episodes take the most amount of time to complete, then gradually decrease. The curves are fairly noisy, but these would smooth out if many more trials were conducted, for the vast majority of my experiments these curves are averaged over ten trials. Since each action step is 2 meters, if the agent moves in as direct a path as possible to the next goal, then there will be half as many steps as there

are meters between the goals, so the optimal value would be 0.5, which most of these curves approach.

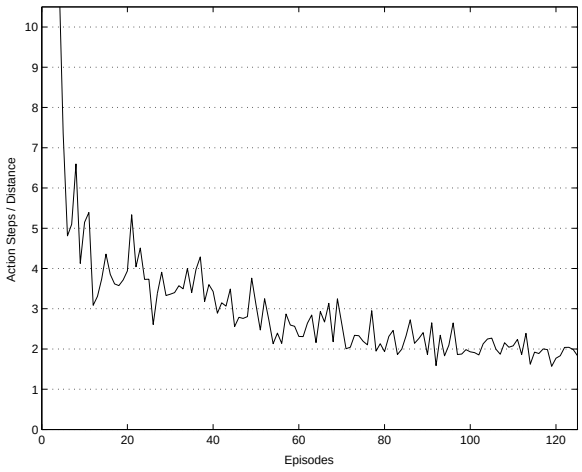

Fig. 5.1: TGU - Full Knowledge, Learning Curve Start

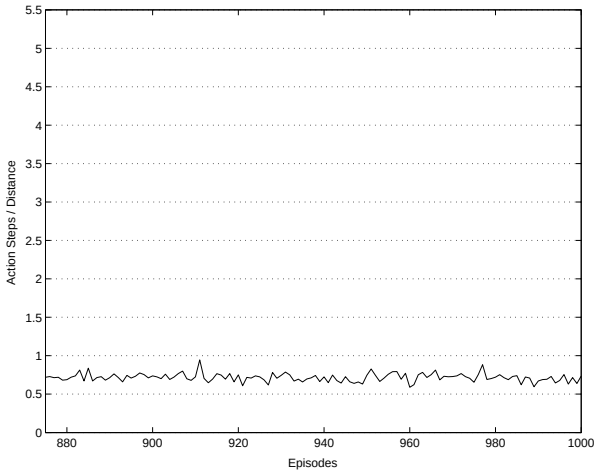

Fig. 5.2: TGU - Full Knowledge, Learning Curve End

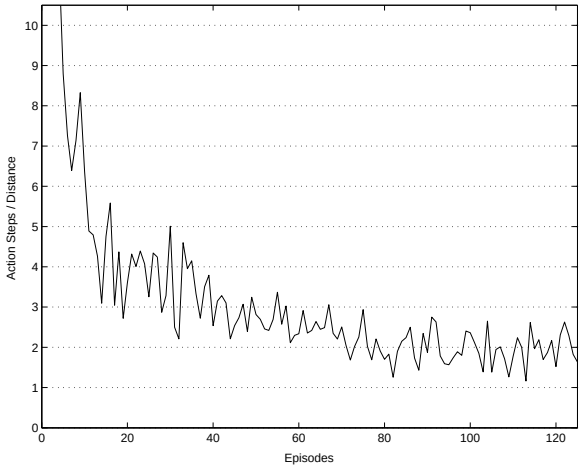

Fig. 5.3: Full Trickle - Full Knowledge, Learning Curve Start

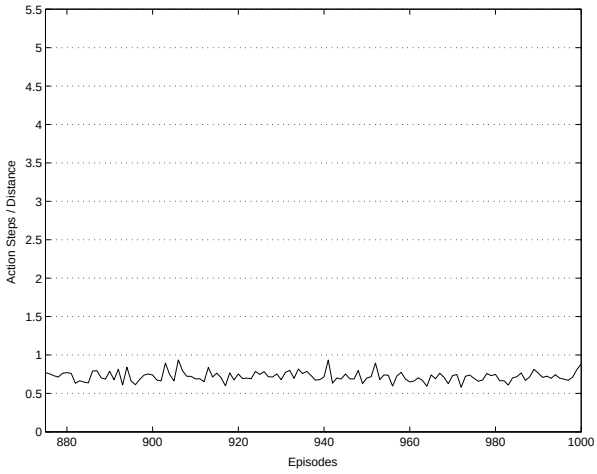

Fig. 5.4: Full Trickle - Full Knowledge, Learning Curve End

5.1.2 TGU vs Full Trickle - Reward Curves

The reward curve in the case where all three agents have knowledge about the other agents using the TGU distribution can be seen in Figure 5.5. The reward curve in the case where all three agents have knowledge about the other agents using the Full Trickle distribution can be seen in Figure 5.6. The x-axis corresponds to the episode number, while the y-axis corresponds to the cumulative reward received by an agent so far.

These reward curves are typical of the reward curves seen in the rest of this dissertation. These curves are averaged over ten trials usually, and in this case, the three agents' cumulative reward is averaged together to give us extra data, but in all of the other graphs the three agents' curves are shown seperately. These agents are losing reward due to the negative reward of the task, and then after learning the task, the agents begin to receive more reward than they are losing. The U-shaped curve is more prominent in this case because it is the case where the agents have full knowledge of the other agents so the state space is much larger and require extra time to learn the optimal policy.

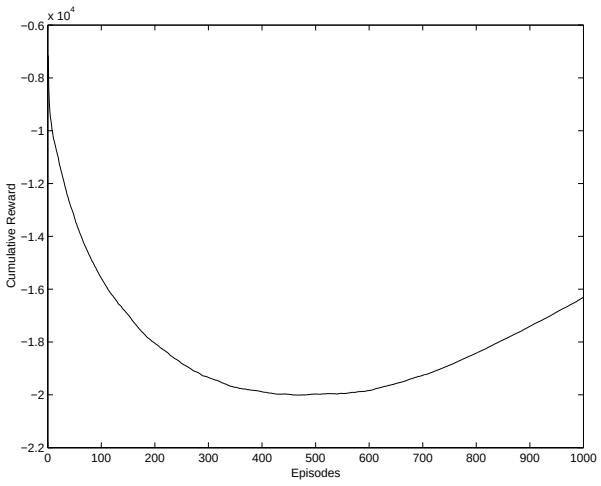

Fig. 5.5: TGU - Full Knowledge, Average Reward Curve for Three Agents

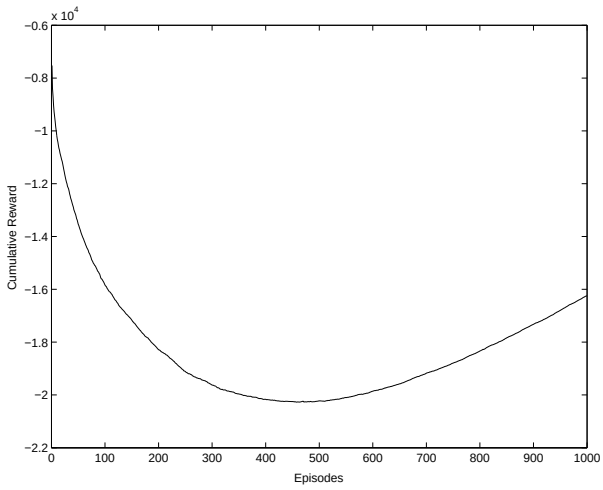

Fig. 5.6: Full Trickle - Full Knowledge, Average Reward Curve for Three Agents

5.1.3 TGU vs Full Trickle - Discussion

For this particular task, the performance of TGU shows that it is intuitively the best method to use, because it does not encourage any competition among the agents. And as these results prove, the Full Trickle method can perform just as well as TGU. That being said, TGU will not always be the best method, because of the disadvantages mentioned in Chapter 2, where it encourages bad behavior in agents not responsible for accomplishing the goal. So, the Trickle method may prove more beneficial in the cases where TGU falters, because the Trickle method should allow for different amounts of reward to be given to different agents. By not giving reward to agents that are not helpful towards accomplishing the goal, their behavior will not be encouraged, and thus we can avoid the disadvantage of using TGU. This, of course, is after the issues of determining the variable benevolence levels are solved.

5.2 Static Trickle

For these experiments, we compare how differing benevolence levels affect the performance of the group and the performance of the individual agents. In the No Trickle case, the benevolence levels are set to 0%, so the last agent to reach the goal receives all of the reward and does not redistribute any of it to the other agents. This creates a competitive task because all of the agents want to be the last agent to reach the goal. Agents are trying to learn to balance staying at the goal so as to minimize lost utility from the negative reinforcement of the world with staying out of the goal to try to grab the opportunity to receive the large episodic reward of being the last agent to reach the goal. This should have the worst performance of the three static benevolence levels, at least for this task.

As discussed before, the Full Trickle case creates an uncompetitive task since an agent benefits nothing from being the last agent to reach the goal. This should have the best performance.

In the Half Trickle case, the benevolence levels are set to 50%, and we can expect that the performance results should fall somewhere between the other two cases.

Below we compare the cases where all agents have no knowledge of one another,

where one agent has knowledge of the others, and where all agents have knowledge of the others.

5.2.1 No knowledge of others

5.2.1.1 Learning Curves for different Static Trickle levels with no knowledge of other agents

The learning curves for the three different benevolence levels, in the case where all agents have no knowledge of the other agents can be seen in Figures 5.7 and 5.8 for No Trickle, Figures 5.9 and 5.10 for Half Trickle, and Figures 5.11 and 5.12 for Full Trickle.

When comparing the three different cases, it can be said that as the benevolence increases, the performance of the entire group improves. Looking at the convergence value at the end of 1000 episodes, it can be seen that as the benevolence increases, the curve approaches the optimal value of 0.5. The agents are learning the optimal policy that will get them to the goal as quickly as possible. It can also be seen, that the group seems to be learning slightly faster in the first hundred episodes as the benevolence increases, because there is no competition, so there is a constant source of reward in each episode. Additional trials would be very useful to reduce any uncertainty here when comparing the different graphs.

5.2.1.2 Reward Curves for different Static Trickle levels with no knowledge of other agents

Another useful graph to study is the reward difference curve, which is the derivate of the reward curve and indicates the amount of reward an agent receives in an episode. In most cases, the three reward curves and reward difference curves of the three agents are very similar and are superimposed over one another, but in the case where one agent has more knowledge than the others, a difference in the curves becomes more prominent. The reward curves and reward difference curves for the three different benevolence levels in the case where all agents have no knowledge of the other agents can be seen in Figures 5.13 and 5.14 for No Trickle, Figures 5.15 and 5.16 for Half

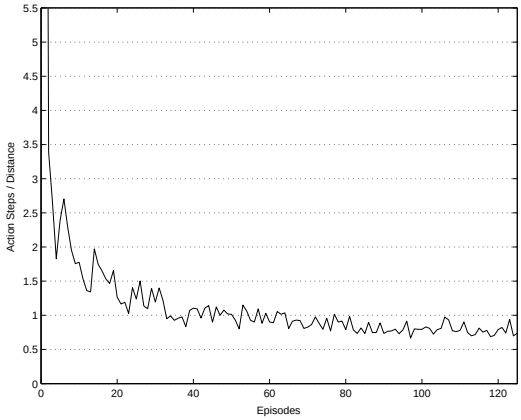

Fig. 5.7: No Trickle - No Knowledge, Learning Curve Start

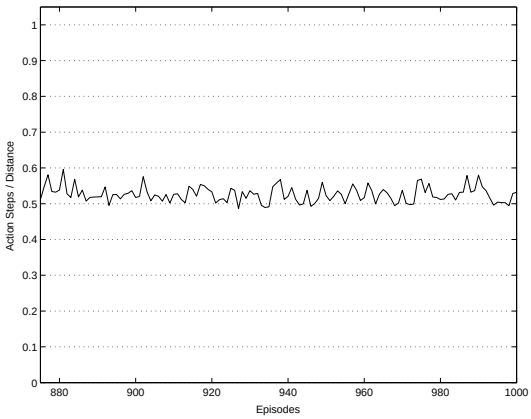

Fig. 5.8: No Trickle - No Knowledge, Learning Curve End

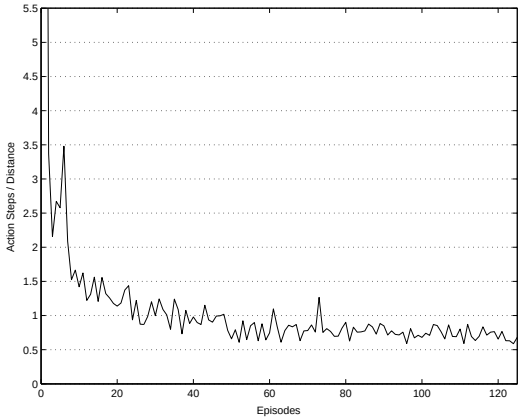

Fig. 5.9: Half Trickle - No Knowledge, Learning Curve Start

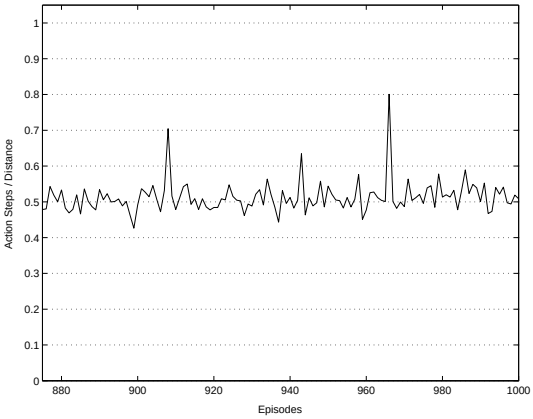

Fig. 5.10: Half Trickle - No Knowledge, Learning Curve End

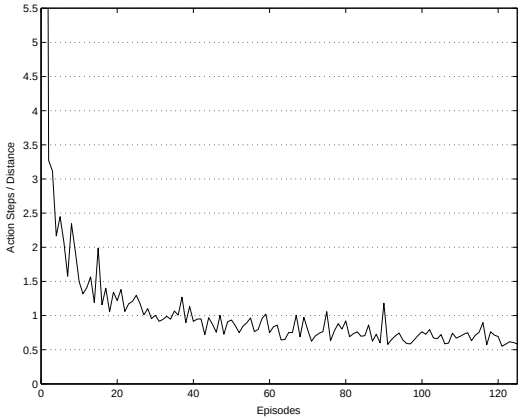

Fig. 5.11: Full Trickle - No Knowledge, Learning Curve Start

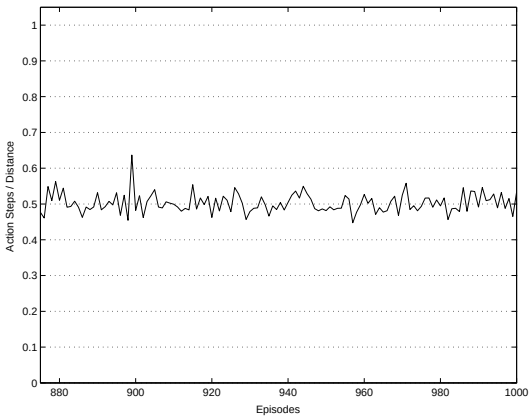

Fig. 5.12: Full Trickle - No Knowledge, Learning Curve End

Trickle, and Figures 5.17 and 5.18 for Full Trickle.

As can be seen in the reward difference curves, as the benevolence decreases, the variance increases. This can easily be linked to the competition for reward. Each of the agents would, on average, win one out of every three episodes, so without sharing, the individual reward received in each episode will fluctuate much more than the case where there is sharing. Though, when looking at trials individually, it seems that usually one of the three agents tends to become dominant over the other two, because they might learn a policy which tended to get them into the goal last.

When comparing the cumulative rewards in the three cases, it can be seen that as the benevolence increases, the total reward of the individual agents, and of the system increases. This coincides with the fact that the group is reaching their goals when they have higher benevolence levels.

5.2.2 One Agent has knowledge of others

5.2.2.1 Learning Curves for different Static Trickle levels with one agent having knowledge of other agents

The learning curves for the three different benevolence levels, in the case where one agent has knowledge of the other agents, can be seen in Figures 5.19 and 5.20 for No Trickle, Figures 5.21 and 5.22 for Half Trickle, and Figures 5.23 and 5.24 for Full Trickle. These learning curves are separated by the different agents, because there is a slight difference in performance for the agent with additional knowledge versus the other two agents.

Again, we find that as the benevolence increases, the performance increases in all the agents, though clearly, the agent with more knowledge is performing worse and learning slower. This is a result of his state space being much bigger, so there is much more exploration before a suitable policy can be established. There is a concern that because the agent has a much larger state space, he suffers from the additional computation needed to search through his state space and Q-value vectors, which may be an issue with asynchronous agents. So we also look at the number of learning steps, instead of action steps, the agents take to finish an episode and find that the agent with

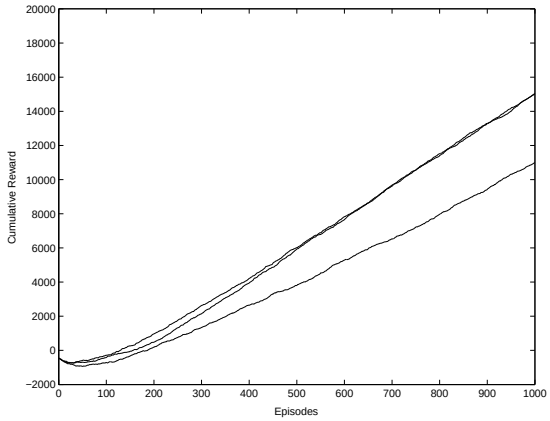

Fig. 5.13: No Trickle - No Knowledge, Reward Curve

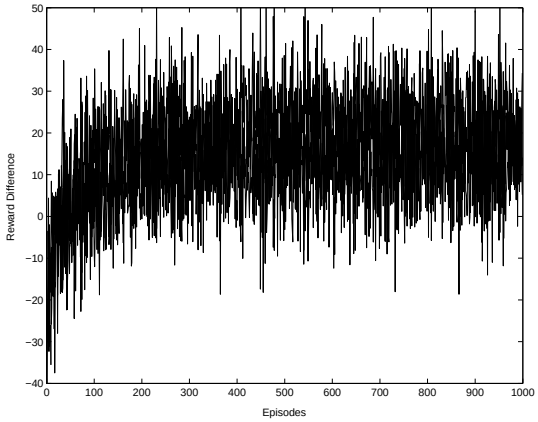

Fig. 5.14: No Trickle - No Knowledge, Reward Difference Curve

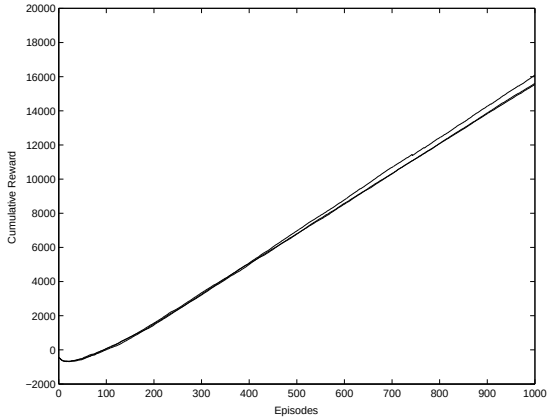

Fig. 5.15: Half Trickle - No Knowledge, Reward Curve

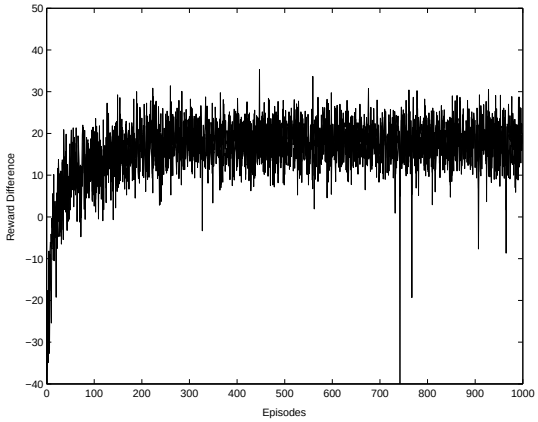

Fig. 5.16: Half Trickle - No Knowledge, Reward Difference Curve

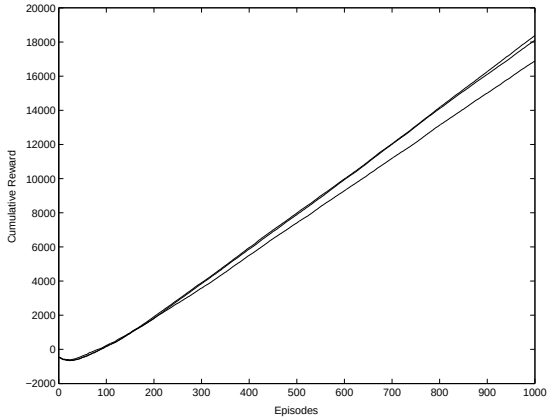

Fig. 5.17: Full Trickle - No Knowledge, Reward Curve

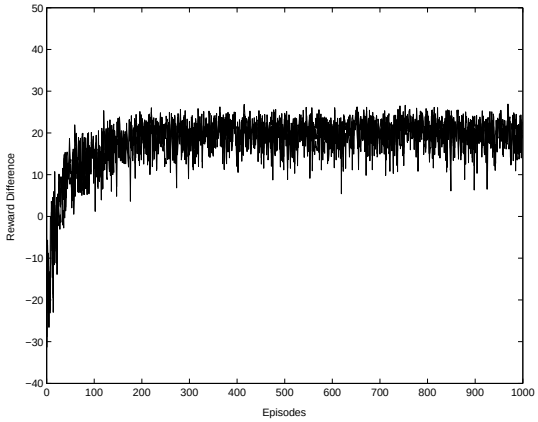

Fig. 5.18: Full Trickle - No Knowledge, Reward Difference Curve

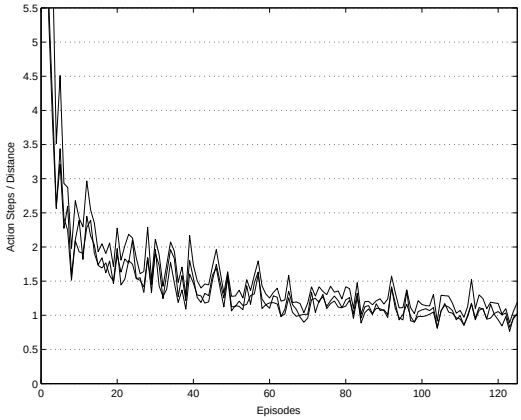

Fig. 5.19: No Trickle - One Has Knowledge, Learning Curve Start

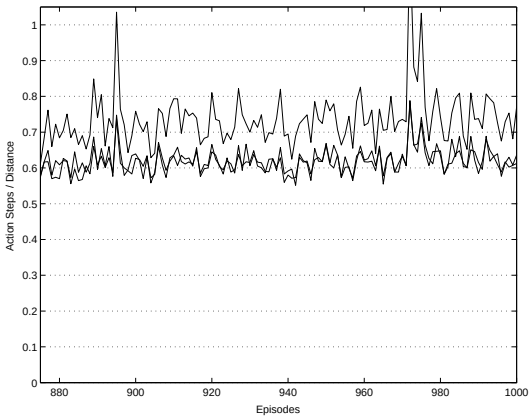

Fig. 5.20: No Trickle - One Has Knowledge, Learning Curve End

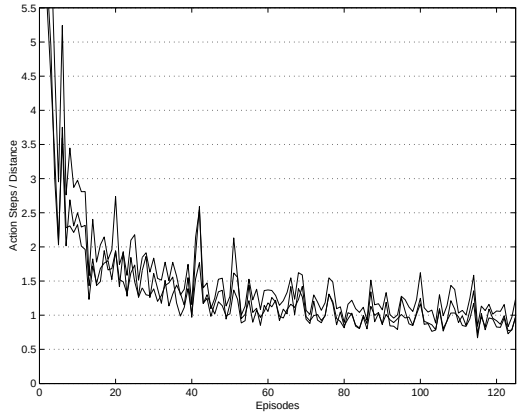

Fig. 5.21: Half Trickle - One Has Knowledge, Learning Curve Start

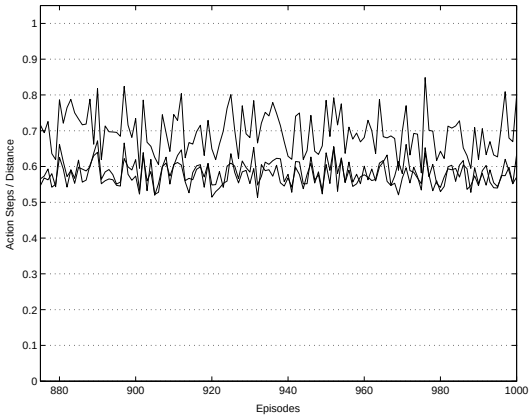

Fig. 5.22: Half Trickle - One Has Knowledge, Learning Curve End

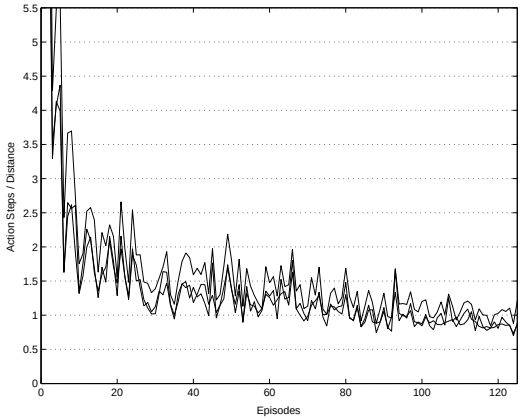

Fig. 5.23: Full Trickle - One Has Knowledge, Learning Curve Start

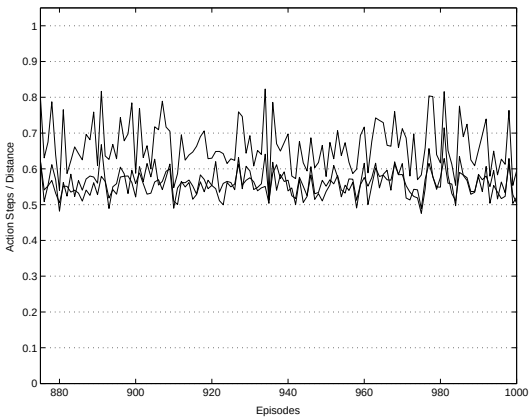

Fig. 5.24: Full Trickle - One Has Knowledge, Learning Curve End

extra knowledge takes fewer learning steps. These learning curves with the learning steps on the y-axis can be seen in Figures 5.25 and 5.26 for No Trickle, Figures 5.27 and 5.28 for Half Trickle, and Figures 5.29 and 5.30 for Full Trickle.

The agent with extra knowledge should experience a state change whenever any one of the agents' proximity to the goal changes, so initially it was expected that this agent would be forced to experience more learning steps than the other two agents. Perhaps, the extra computation is indeed slowing the agent from experiencing as many states as the other agents.

Alternatively, the agent learns that it is best to stay away from the goal until the other two agents are in or close to the goal, so he might spend more time further away from the goal, where the states cover a much larger area due to the state space setup of using proximity and direction. This is backed by the fact that the agent is taking more action steps. There are many more action steps in a learning step when agents are further from the goal. At the same time, the other agents will be experiencing many more learning steps as they move within the goal waiting for the third agent to enter, because of the nature of how the state space is setup. But since the other agents' proximity won't be changing as often as their direction, the knowledge within the other agent is not being changed, and thus his state is not changed even though the other agents' states may have changed. Several observances of the actual behavior while in game mode support this idea, since this agent tended to stay further away from the goal than the other agents, but a more thorough policy analysis could provide some helpful insight.

5.2.2.2 Reward Curves for different Static Trickle levels with one agent having knowledge of other agents

The reward curves and the reward difference curves for the three different benevolence levels in the case where one agent has more knowledge than the other agents can be seen in Figures 5.31 and 5.32 for No Trickle, Figures 5.33 and 5.34 for Half Trickle, and Figures 5.35 and 5.36 for Full Trickle.

The immediate result that is evident is the difference in reward between the agent with additional knowledge versus the other two agents. It turns out that the agent with

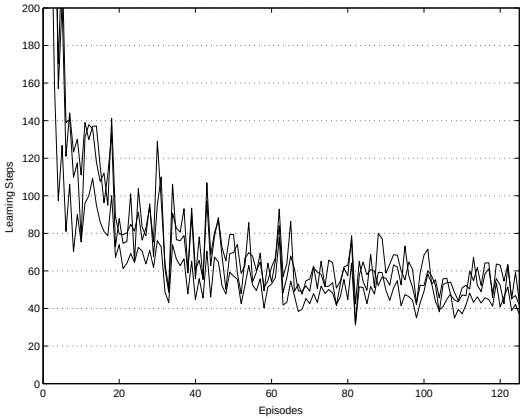

Fig. 5.25: No Trickle - One Has Knowledge, Learning Curve Start

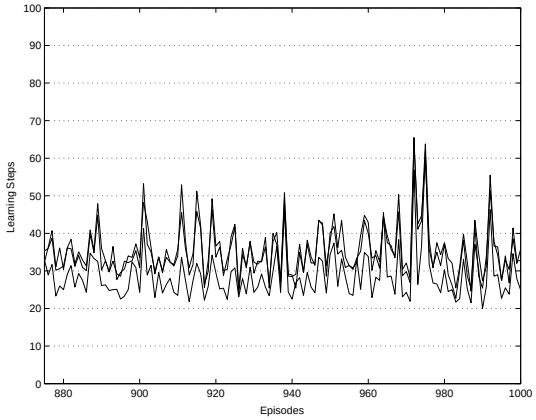

Fig. 5.26: No Trickle - One Has Knowledge, Learning Curve End

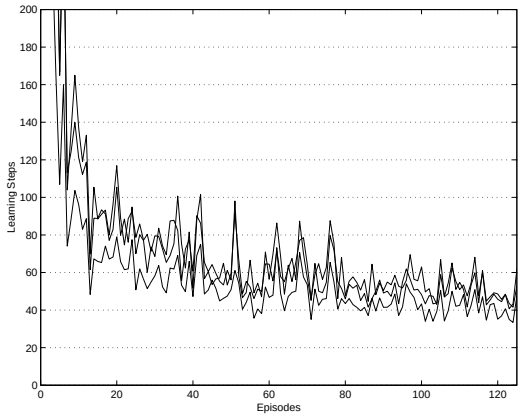

Fig. 5.27: Half Trickle - One Has Knowledge, Learning Curve Start

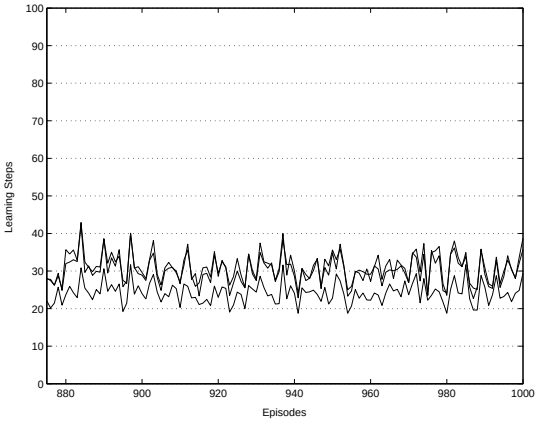

Fig. 5.28: Half Trickle - One Has Knowledge, Learning Curve End

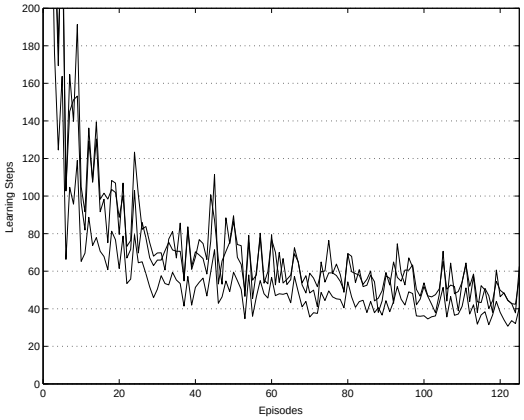

Fig. 5.29: Full Trickle - One Has Knowledge, Learning Curve Start

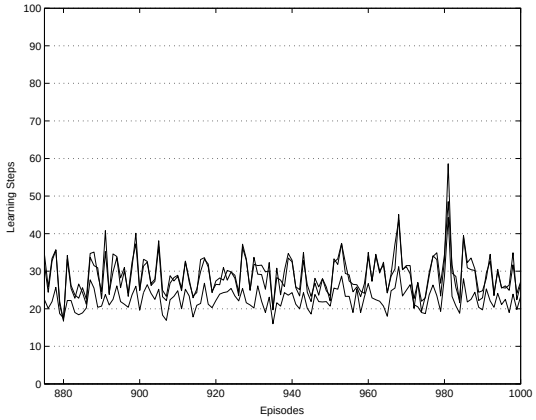

Fig. 5.30: Full Trickle - One Has Knowledge, Learning Curve End

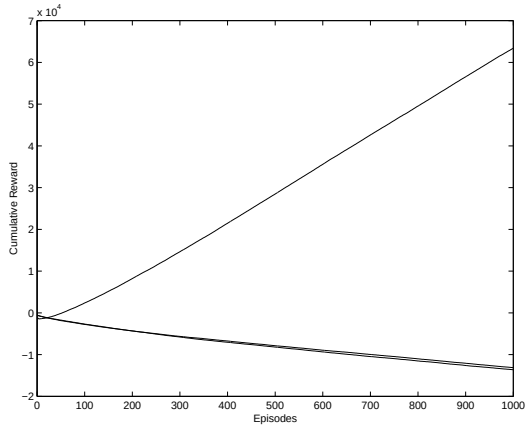

Fig. 5.31: No Trickle - One Has Knowledge, Reward Curve

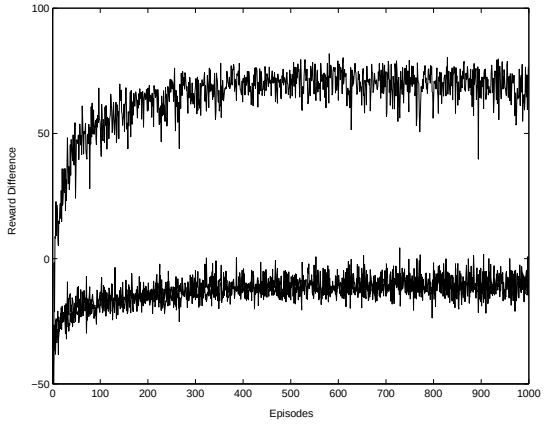

Fig. 5.32: No Trickle - One Has Knowledge, Reward Difference Curve

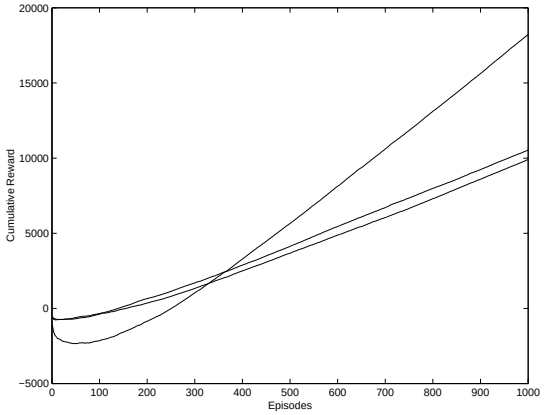

Fig. 5.33: Half Trickle - One Has Knowledge, Reward Curve

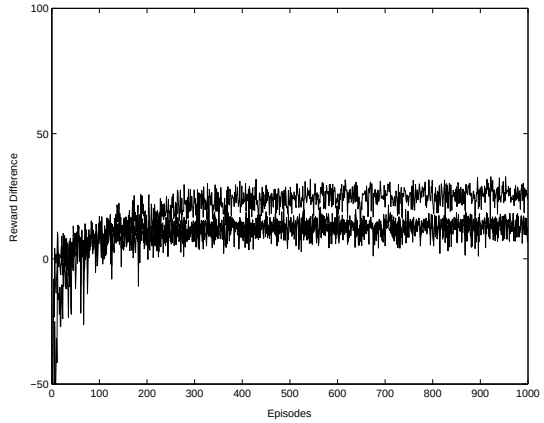

Fig. 5.34: Half Trickle - One Has Knowledge, Reward Difference Curve

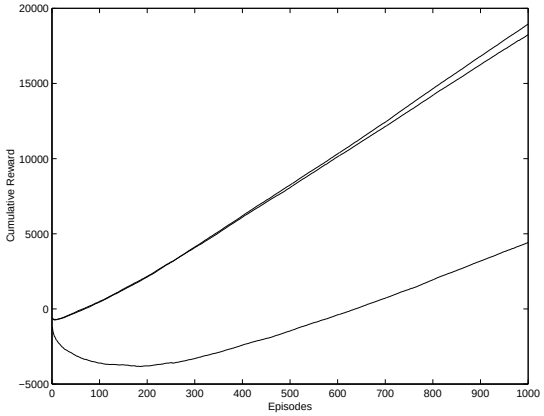

Fig. 5.35: Full Trickle - One Has Knowledge, Reward Curve

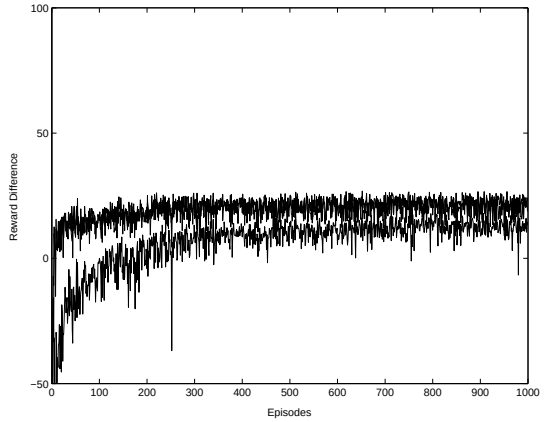

Fig. 5.36: Full Trickle - One Has Knowledge, Reward Difference Curve

additional knowledge receives the episodic reward about 95% of the time. This is an entirely expected result. In the earlier episodes, it takes this agent much longer to explore the state space, so he will not begin following a recognizable policy that will take him to the goal until after the other two. In the later episodes, since this agent is receiving this reward regularly, his policy is dominated by this reward, meaning that while the other two agents are optimizing their policy to move to the goal to minimize negative reward, this agent is following a policy that will lead him to this episodic reward. Thus, it is more beneficial for the agent to stay further away from the goal until the other two agents are near the goal, because he will experience less learning steps, and thus less negative reward. This new evidence helps to confirm the idea brought forth in the last section, explaining why this agent will take more action steps, while taking fewer learning steps.

Also of particular note is that as the benevolence increases, the cumulative group reward of the three agents also increases, which confirms that as the agents become less competitive, the group will perform better as they do not encourage suboptimal policies. It would be interesting to show that as the benevolence increases, the knowledgable agent's policy is less dependent on the episodic reward and more dependent on getting to the goal as quickly as possible.

This suggests that as benevolence between agents increases, additional knowledge becomes less relevant and simply complicates the learning process.

5.2.3 All Agents have knowledge of others

5.2.3.1 Learning Curves for different Static Trickle levels with all agents having knowledge of other agents

The learning curves for the three different benevolence levels, in the case where all three agents have knowledge of the other agents can be seen in Figures 5.37 and 5.38 for No Trickle, Figures 5.39 and 5.40 for Half Trickle, and Figures 5.41 and 5.42 for Full Trickle.

Again, we find that as the benevolence increases, the performance of the group improves as well. Of particular note, is that the performance is not as good as in the

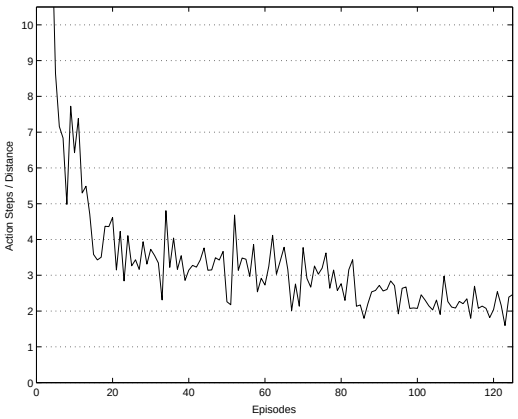

Fig. 5.37: No Trickle - All Have Knowledge, Learning Curve Start

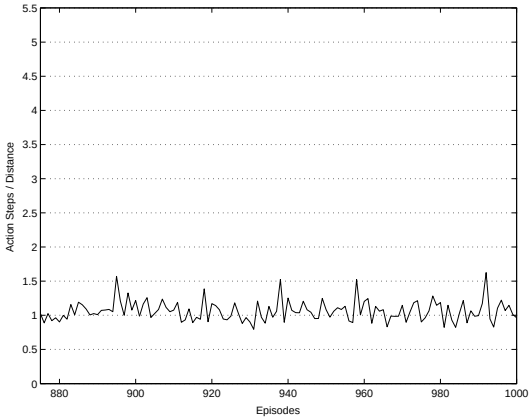

Fig. 5.38: No Trickle - All Have Knowledge, Learning Curve End

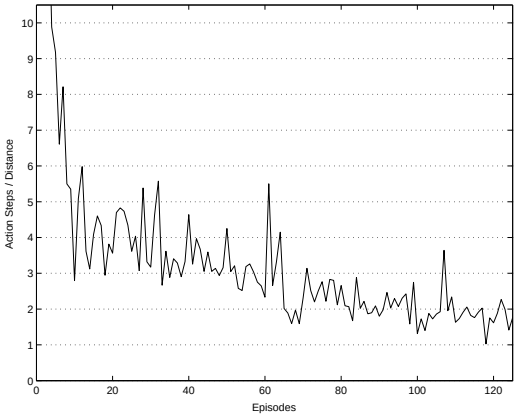

Fig. 5.39: Half Trickle - All Have Knowledge, Learning Curve Start

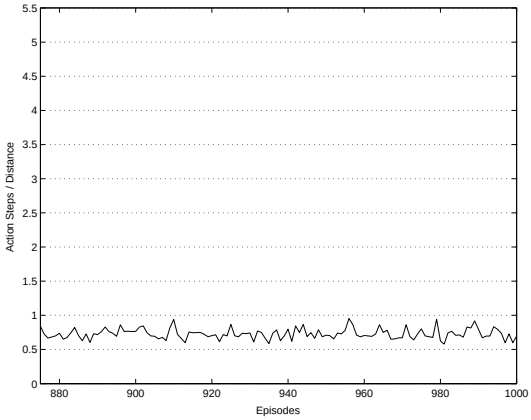

Fig. 5.40: Half Trickle - All Have Knowledge, Learning Curve End

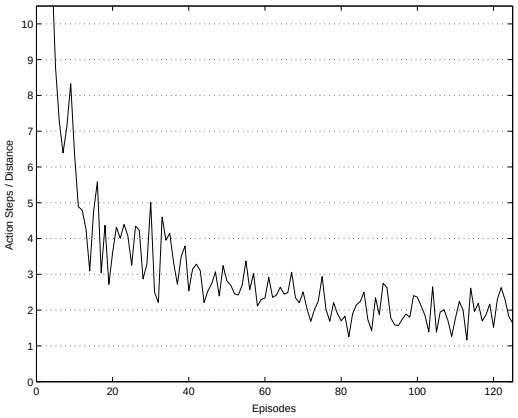

Fig. 5.41: Full Trickle - All Have Knowledge, Learning Curve Start

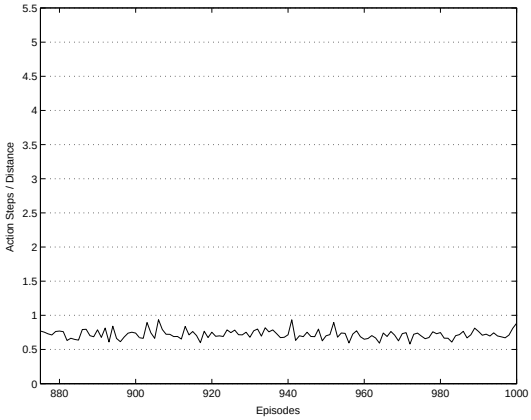

Fig. 5.42: Full Trickle - All Have Knowledge, Learning Curve End

cases where all agents have no knowledge or where only one agent has knowledge. By increasing the agents' knowledge, their state spaces are increased and learning becomes more complex and slower. This brings up the issue of when is it desirable to have more knowledge of the system, and when is it desirable to have limited knowledge. Similar to the discussion about local or global perspective, in certain situations, following the cliche, sometimes "ignorance is bliss."

5.2.3.2 Reward Curves for different Static Trickle levels with all agents having knowledge of other agents

The reward curves and the reward difference curves for the three different benevolence levels in the case where all three agents have knowledge of the other agents can be seen in Figures 5.43 and 5.44 for No Trickle, Figures 5.45 and 5.46 for Half Trickle, and Figures 5.47 and 5.48 for Full Trickle. Here are the reward curves for the three different benevolence levels in the case where all agents have knowledge of the other agents.

As the benevolence increases, the cumulative rewards of the three agents also increases. The curves also become less varied as the competitiveness of the group is reduced.

Another thing to note is the shape of the reward curves. In the Full Trickle, and Half Trickle cases, halfway through the series of episodes, the agents gradually switch from receiving negative reward to receiving positive reward in each episode. This can also be seen in the reward difference curves as they cross zero near 500 episodes. This is a result of fully exploring the state space and settling into the most beneficial policy. It is more obvious in this case than in the cases of less knowledge entirely because of the size of the state space. It simply takes longer to explore, and for the learning parameters to decay to a stable level, allowing the policy to converge.

5.3 Variable Trickle

For this experiment, the episodic reward is only given to the last agent who reaches the goal, and he is able to redistribute the reward amongst the other agents as he chooses.

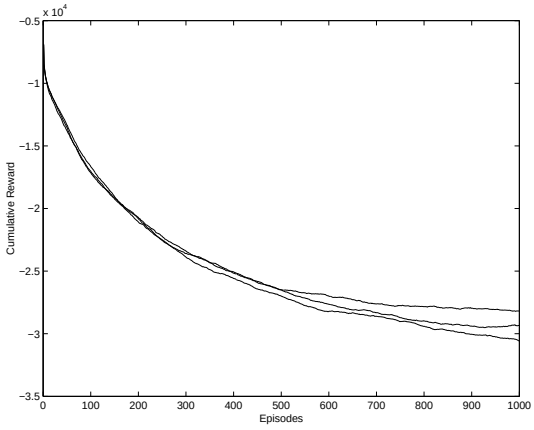

Fig. 5.43: No Trickle - All Have Knowledge, Reward Curve

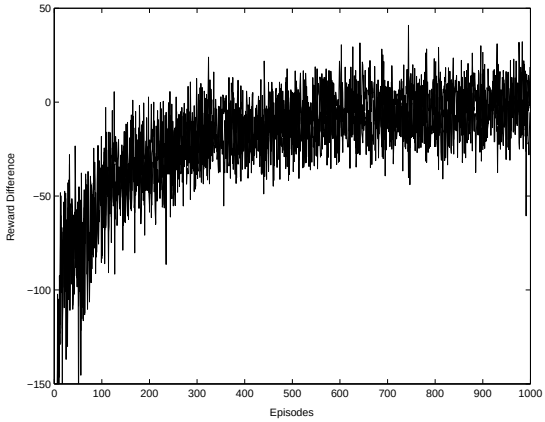

Fig. 5.44: No Trickle - All Have Knowledge, Reward Difference Curve

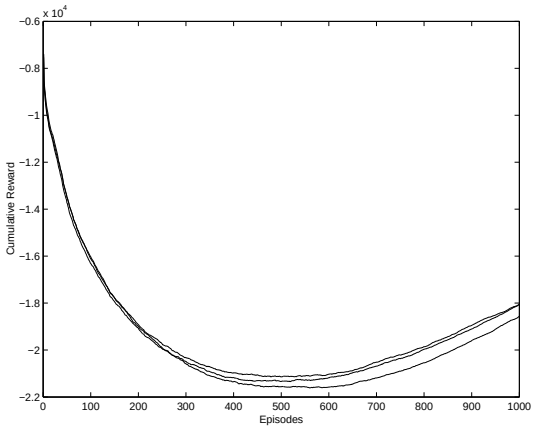

Fig. 5.45: Half Trickle - All Have Knowledge, Reward Curve

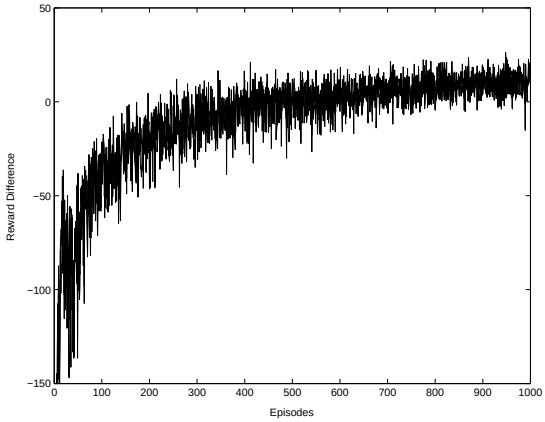

Fig. 5.46: Half Trickle - All Have Knowledge, Reward Difference Curve

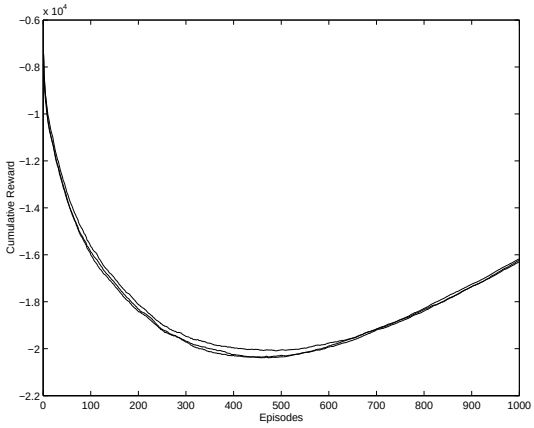

Fig. 5.47: Full Trickle - All Have Knowledge, Reward Curve

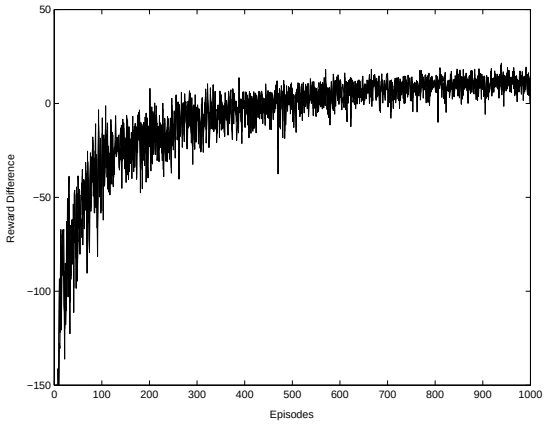

Fig. 5.48: Full Trickle - All Have Knowledge, Reward Difference Curve

This is done by allowing the agent to fluctuate the benevolence levels dynamically, so as to optimize the amount of reward he receives over time.

The original idea was to use reinforcement comparison as a method for choosing the appropriate benevolence levels of an agent. The levels to choose from correspond to preset levels in factors of 10%.

$$level \varepsilon [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$$

The reward received in each episode is counted and an average reward is updated by the equation:

$$\bar{r}_{t+1} = \bar{r} + \alpha * (r_t - \bar{r})$$

The reward is compared against the average reward as a basis for knowing how well the chosen action, or rather, the chosen benevolence level performed. Each benevolence level maintains a preference value which represents the desirability of choosing that level again in the future, which is updated by the equation:

$$pref(level)_{t+1} = pref(level)_t + \beta * (r_t - \bar{r})$$

These preferences are then compared against one another to find the likelihood of choosing each of the respective benevolence levels using a softmax action selection.

$$Percentage(level) = \frac{pref(level)}{\sum_{X=0.0}^{1.0} pref(X)}$$

5.3.1 No knowledge of others

5.3.1.1 Learning Curves for Variable Trickle levels with no knowledge of other agents

The learning curve for using variable trickle levels in the case where all agents have no knowledge of the other agents can be seen in Figures 5.49 and 5.50.

What was hoped to be found, was that the agents would gradually learn the optimal policy and optimal benevolence level to encourage the optimal policy in all of the other agents. Although, the learning curve does approach 0.5, it does not prove convincingly that the system is performing as well as the Half Trickle or Full Trickle cases, but it is performing better than the No Trickle case.

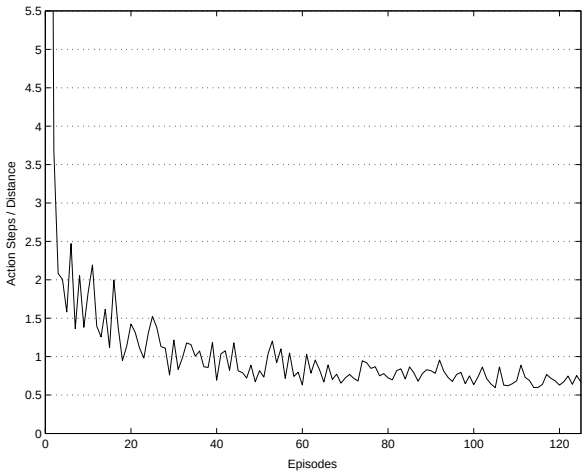

Fig. 5.49: Variable Trickle - No Knowledge, Learning Curve Start

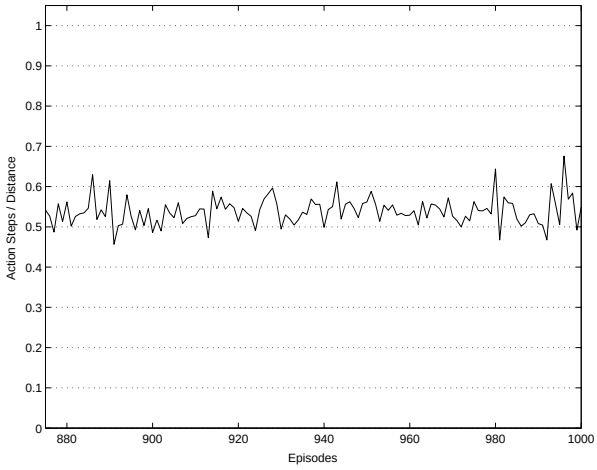

Fig. 5.50: Variable Trickle - No Knowledge, Learning Curve End

5.3.1.2 Reward Curves for Variable Trickle levels with no knowledge of other agents

The reward curve and the reward difference curve for using variable trickle levels in the case where all agents have no knowledge of the other agents can be seen in Figures 5.51 and 5.52.

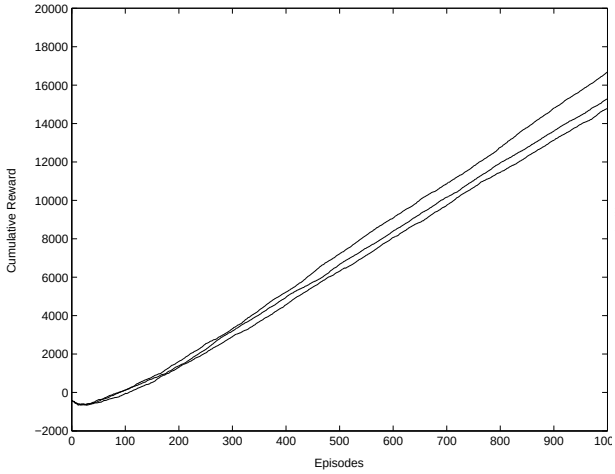

Fig. 5.51: Variable Trickle - No Knowledge, Reward Curve

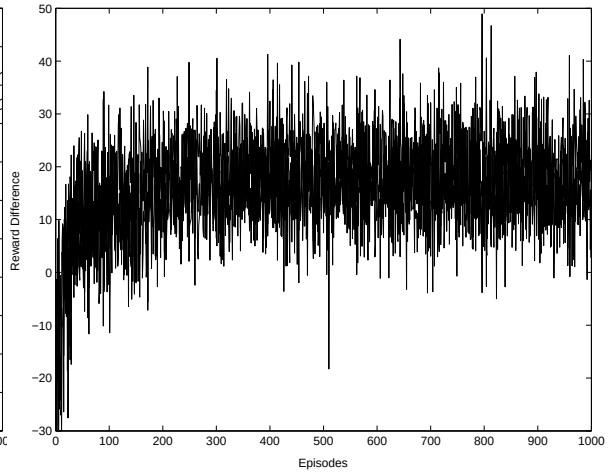

Fig. 5.52: Variable Trickle - No Knowledge, Reward Difference Curve

Looking at the reward curves, the results are most similar to the Half Trickle case, both in the cumulative reward received after 1000 episodes and in the variance in the reward difference curve.

What was hoped to be found, was that there would be a gradual increase in the reward received in each episode as the agents learned the optimal benevolence levels. But this can't clearly be seen, as the reward curve appears fairly linear after the initial learning in the beginning.

5.3.1.3 Preferences for Variable Trickle levels with no knowledge of other agents

Next, we look at the preferences for the various benevolence levels, to see if we can find some relationship between the chosen level and the expected reward relative to the other levels. T The preferences all start out at 100 and change in a relative amount to each other. If a particular benevolence level is more beneficial than the others, then its preference level will increase and will be more likely to be chosen when the benevolence level is chosen using a softmax selection. The preferences across all of the benevolence levels can be seen in Figure 5.53.

When plotting the preferences after 1000 episodes, we find that there is no direct relationship between the benevolence chosen and the expected reward. We were hoping to find a curve that was proportional between the benevolence level and the preference to choose it. Instead it appears that the preferences of the benevolence levels is being determined pretty randomly, which confirms why the learning and reward curves show comparable results to the Half Trickle case. Some reward is better than no reward, so agents that will distribute an arbitrary amount of reward to the other agents will be more cooperative than agents that distribute no reward at all.

5.3.2 Why this implementation of Variable Trickle is ineffective

Using reinforcement comparison as it is implemented here to determine the variable benevolence is completely ineffective, which in retrospect seems pretty obvious. There is simply not enough time to learn the best level to choose. By changing the benevolence levels at every step, the other agents do not have enough time to react to the amount of reward given in order to influence their future actions. One possible

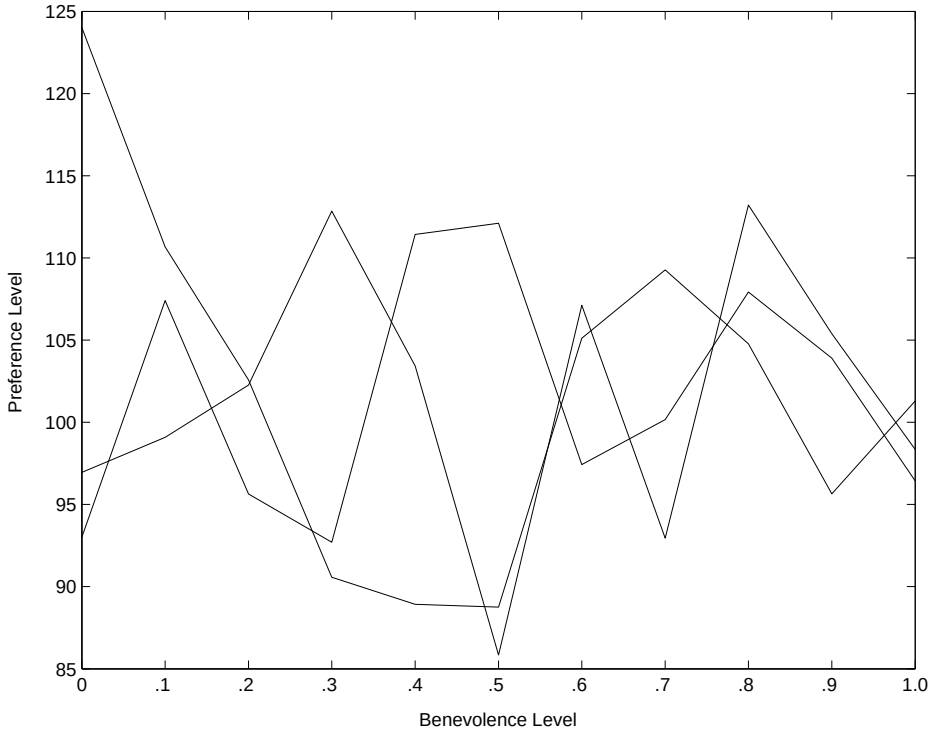

Fig. 5.53: Variable Trickle - No Knowledge, Preferences

solution is to keep this strategy but wait 10 or 100 episodes to try to notice any difference. Also, it may be necessary to look at the rate of change as opposed to the change in reward, because the agents are learning simultaneously and constantly improving their behavior due to trying to get to the goal in order to minimize negative reward.

5.4 Why this task is ineffective

Upon deeper inspection into the current setup, it becomes evident that perhaps this particular task is flawed. Selfish agents always want to maximize reward, or minimize negative reward, and by giving them zero reward when in the pond, the task loses its cooperative/competitive aspect. It is always in the agents' best interests to move to the pond as fast as possible. The episodic reward simply complicates the learning by enticing them to sometimes choose suboptimal behavior in order to possibly receive

more reward. The cooperative aspect is that agents are meant to learn to share reward in order to undermine choosing this suboptimal behavior. This seems to be an ineffective task to test the learning. A new task might provide more beneficial research, which is what we do in Chapter 6.

Chapter 6

Revised Experiments & Results

What if we could instead create a competitive task where agents must learn to cooperate in order to improve their behavior. If the agents become fully benevolent, optimal behavior becomes possible. So the new task removes the zero reward for being at the goal, and instead agents receive a negative reward for every action step. In addition, once an agent reaches the goal, he remains in the goal and waits for the other agents to complete the episode, acquiring negative reward as well, so they want the other agents to finish as quickly as possible too. In this case, the team reward is proportional to how fast the last agent reaches the goal.

Thinking in the role of the last agent that enters the goal, he must "thank" or payoff the other agents for going in before him. Especially, in the case where all three agents will have knowledge of the others, none of them would be willing to go into the goal before the other agents, unless there was some incentive, such as reward given to them from the other agents. A real world example of this could be an endurance test between two competitors. The competitors, or agents, will try and hold out as long as possible, receiving negative reward, in hopes of winning the prize, the episodic reward. If the agents were able to share reward, such as one agent promising to give the other agent half of his winnings if he bows out of the competition to let the first agent win, then they wouldn't be stuck holding out forever until both of the agents receive more negative reward than the prize of winning was worth.

6.1 Implementation Issues

The first step in implementing this new task was to implement a working doNothing action. When an agent became Near to the goal, a flag is set indicating that this agent has “Been” to the goal. With this flag raised, the only allowed action is the doNothing action, which is essentially an empty function that does not affect the agent position. In addition, this action is only allowed to occur when this flag is raised, so agents are always moving when not in the goal. To solve the issues discussed in Section 4.2.2.2, upon completion of the action, which is instantaneous, instead of immediately calling the next learning step, the agent waits until the other agents finish their next step, so as to emulate the time delay between steps. This makes the agents partially synchronous when agents are in the goal, but is necessary to be able to evenly time the negative reward they are receiving at each step. The disadvantage of this approach, is that the agents become dependent on one another, and in the case where two agents are in the goal waiting for the third, if the third agent falls, as described in Section 4.4.3.2, the trial will crash because there is no way for the system to detect this failure as it is currently implemented.

6.2 Static Trickle

The same set of experiments using various benevolence levels and the different cases of agent knowledge are conducted using this new task to see what observations can be made. When looking at the reward curves, the expected utility landscape is mapped out in hopes of finding a relationship between the benevolence level and expected utility, which will be useful when implementing the Variable Trickle case.

6.2.1 Comparing the Learning Curves

6.2.1.1 Learning Curves for different Static Trickle levels with no knowledge of other agents

The learning curves for the three different benevolence levels in the case where all agents have no knowledge of the other agents can be seen in Figure 6.1 for No Trickle,

Figures 6.2 and 6.3 for Half Trickle, and Figures 6.4 and 6.5 for Full Trickle,

6.2.1.2 Learning Curves for different Static Trickle levels with one agent having knowledge of other agents

The learning curves for the three different benevolence levels in the case where one agent has knowledge of the other agents can be seen in Figures 6.6 and 6.7 for No Trickle, Figures 6.8 and 6.9 for Half Trickle, and Figures 6.10 and 6.11 for Full Trickle,

6.2.1.3 Learning Curves for different Static Trickle levels with all agents having knowledge of other agents

The learning curves for the three different benevolence levels in the case where all agents have knowledge of the other agents can be seen in Figure 6.12 for No Trickle, Figure 6.13 for Half Trickle, and Figure 6.14 for Full Trickle,

6.2.1.4 Observations about the Learning Curves for Static Trickle

As expected, the performance of the group improves as the benevolence increases in every case. The performance also closer approaches the optimal value, and learns faster, as the amount of knowledge agents possess is minimized.

There is a noticeably larger difference between the No Trickle and Half Trickle cases than the Half Trickle and Full Trickle cases. This is because there is a constant source of reward in the Half Trickle and Full Trickle cases, so agents always tend to move towards the goal. Without reward distribution among agents, the group favors suboptimal behavior which keeps them outside of the goal as much as possible. Towards the beginning of the No Trickle trials, the agents' behavior is random and they will stumble upon the goal and get stuck allowing for one of them to win by being the third agent to stumble into the goal. So, the agents learn to move towards the goal because of this occasional reward from winning episodes, but then as the random exploration decreases, the performance of the group also decreases, as seen best in Figure 6.1, as the agents learn that its better for them not to step into the goal. In fact,

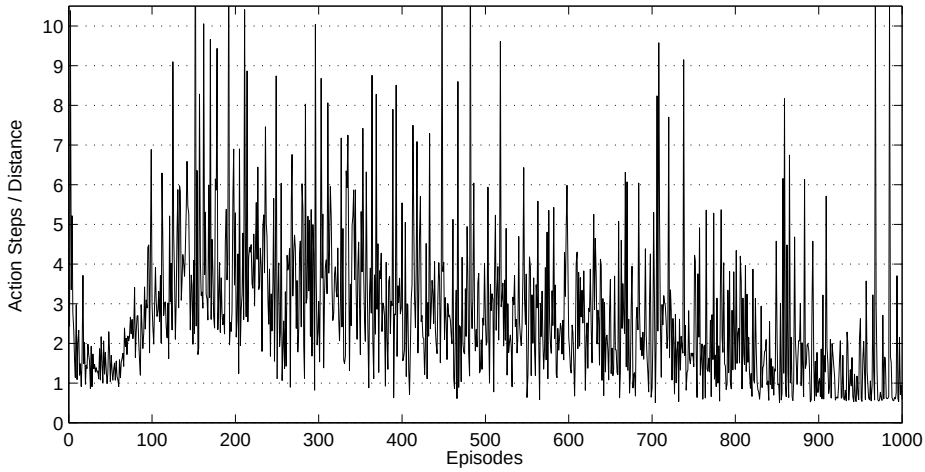

Fig. 6.1: No Trickle - No Knowledge, Learning Curve

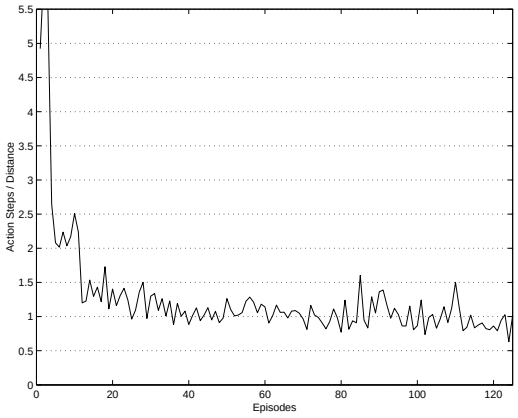

Fig. 6.2: Half Trickle - No Knowledge, Learning Curve Start

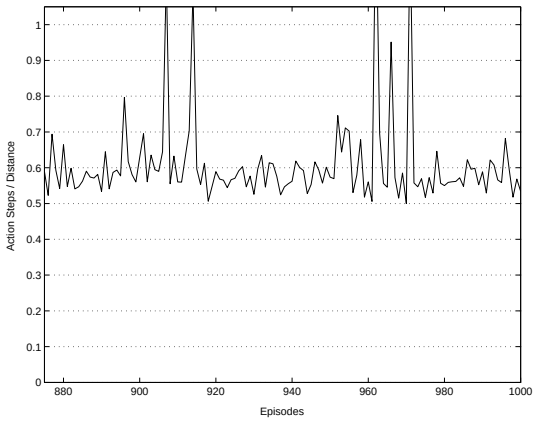

Fig. 6.3: Half Trickle - No Knowledge, Learning Curve End

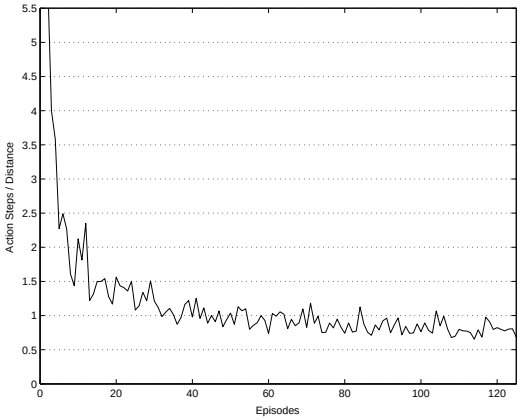

Fig. 6.4: Full Trickle - No Knowledge, Learning Curve Start

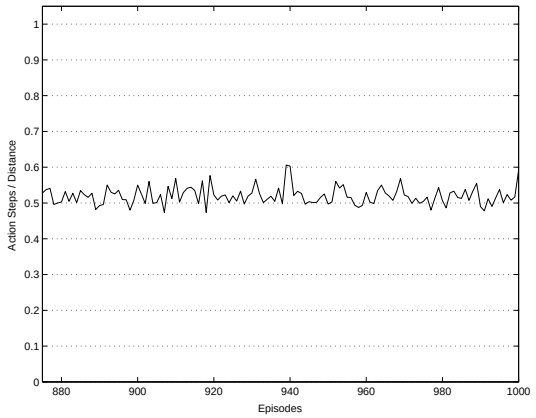

Fig. 6.5: Full Trickle - No Knowledge, Learning Curve End

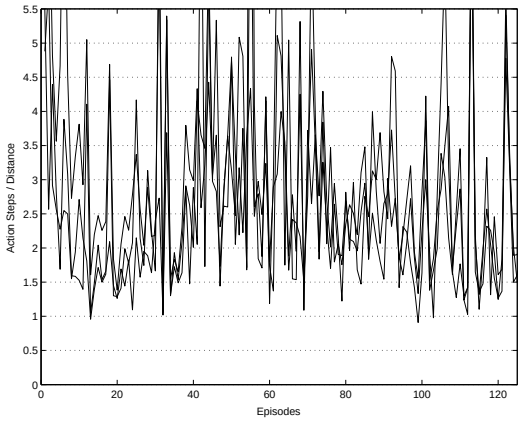

Fig. 6.6: No Trickle - One Has Knowledge, Learning Curve Start

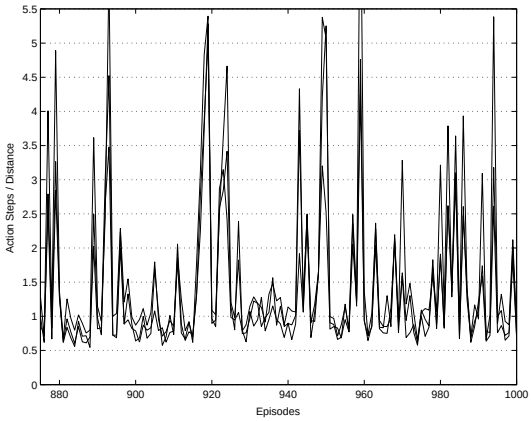

Fig. 6.7: No Trickle - One Has Knowledge, Learning Curve End

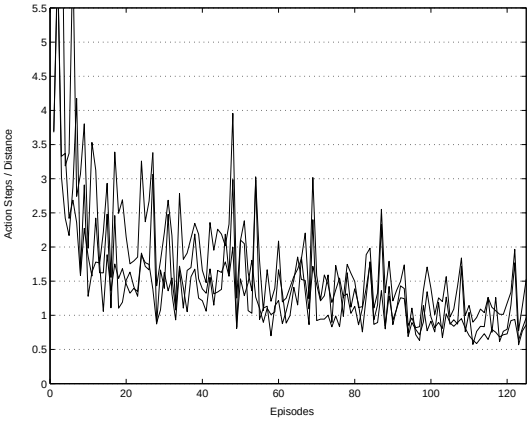

Fig. 6.8: Half Trickle - One Has Knowledge, Learning Curve Start

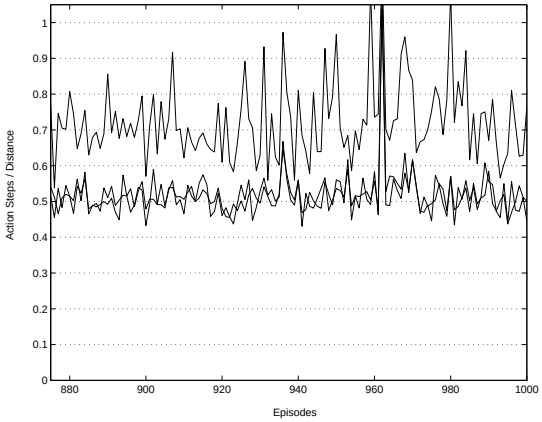

Fig. 6.9: Half Trickle - One Has Knowledge, Learning Curve End

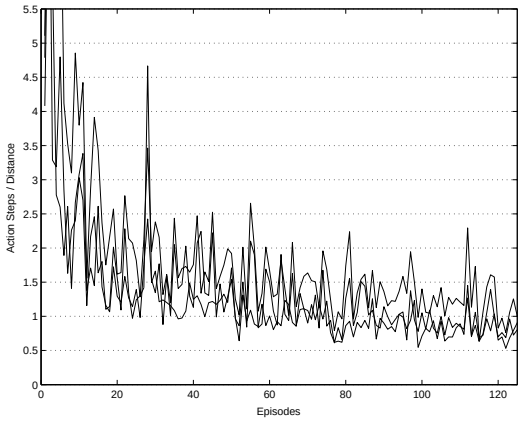

Fig. 6.10: Full Trickle - One Has Knowledge, Learning Curve Start

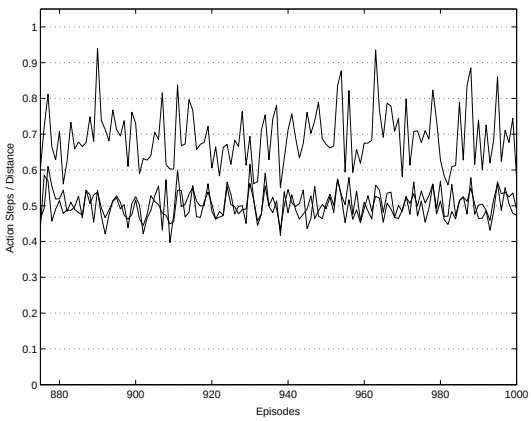

Fig. 6.11: Full Trickle - One Has Knowledge, Learning Curve End

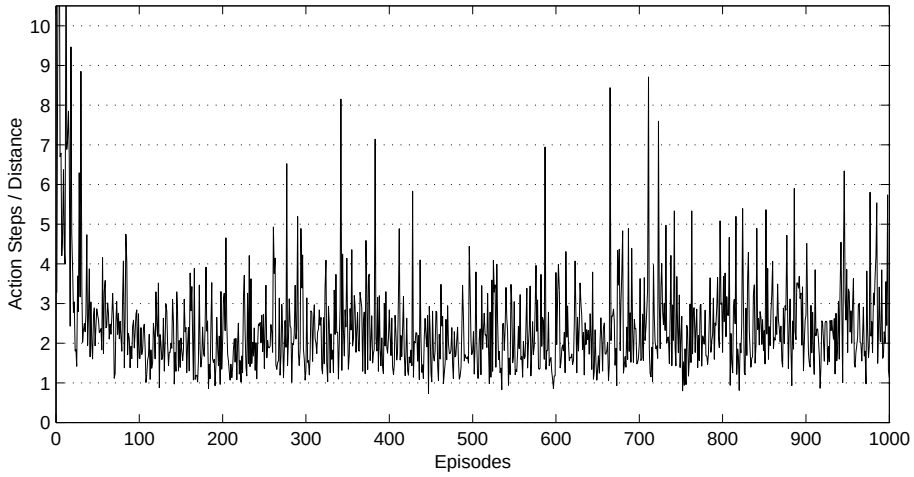

Fig. 6.12: No Trickle - All Have Knowledge, Learning Curve

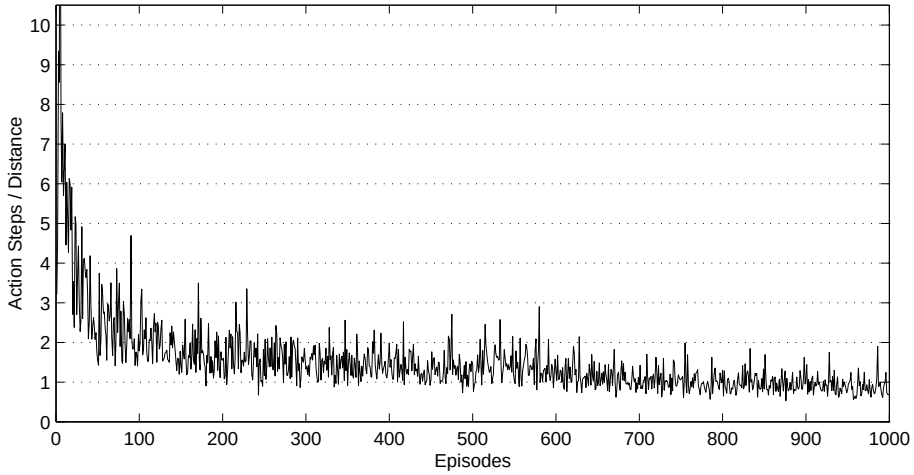

Fig. 6.13: Half Trickle - All Have Knowledge, Learning Curve

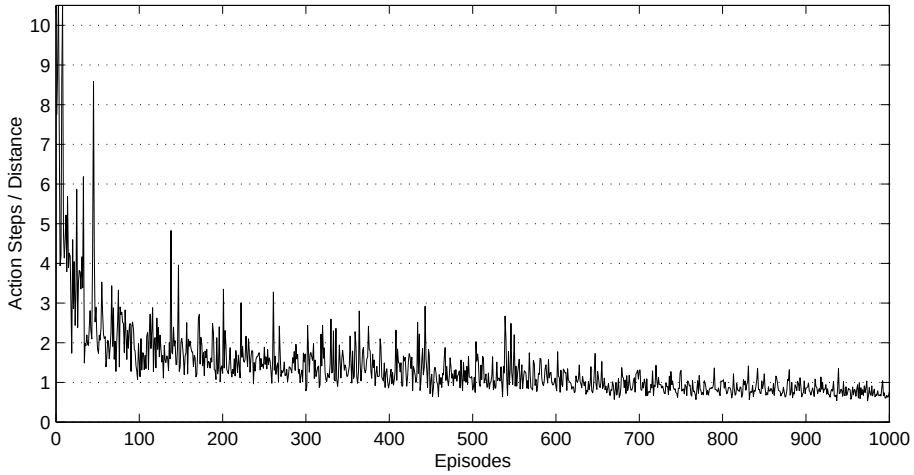

Fig. 6.14: Full Trickle - All Have Knowledge, Learning Curve

we find that the agents' policies favor moving away from the goal as much, if not more so, than moving towards it, because agents tend to receive less reward when in the "Away" state space due to the large area and time between each learning step. It would be interesting to investigate how much performance would decrease as different lower threshold limits are set on the ε-greedy action selection.

Performance does then improve gradually over the thousand episodes, as policies begin to favor moving into the goal again as much as any other action as the occasional positive reward is normalized by the relatively large constant negative reward. It would be an interesting benchmark comparison to conduct a test using agents choosing strictly random actions, to see if they performed even better than these competitive agents without knowledge of the other agents.

6.2.2 Comparing the Reward Curves

The ultimate goal of allowing agents to dynamically vary their benevolence levels is intended to maximize agent utility over time. To this end, the expected utility landscapes of these systems need to be modelled, where we define the expected utility landscape to be the relationship between benevolence levels and expected utility to be received after 1000 episodes when using this benevolence level. We split the analysis into two sections for homogeneous and heterogeneous agents. The landscape for homogeneous agents is expected to be identical between agents. The landscape for the heterogeneous agents will likely differ between agents, so a more in depth comparison will need to be done looking at the individual utilities and the group utility.

6.2.2.1 Reward Curves of Homogeneous Agents

The reward curves and the reward difference curves for the three different benevolence levels in the cases where all agents have no knowledge of the other agents can be seen in Figures 6.15 and 6.16 for No Trickle, Figures 6.17 and 6.18 for Half Trickle, and Figures 6.19 and 6.20 for Full Trickle.

The reward curves and the reward difference curves for the three different benevolence levels in the case where all agents do have knowledge of the other agents

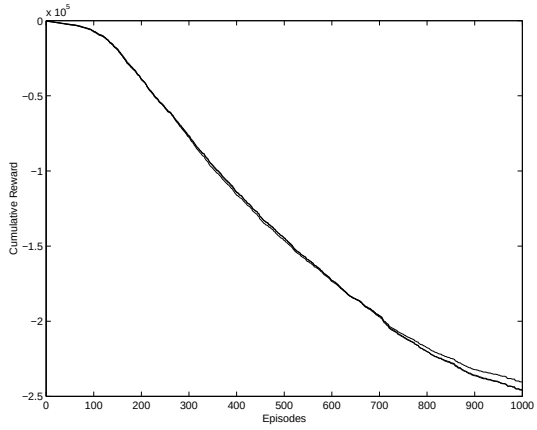

Fig. 6.15: No Trickle - No Knowledge, Reward Curve

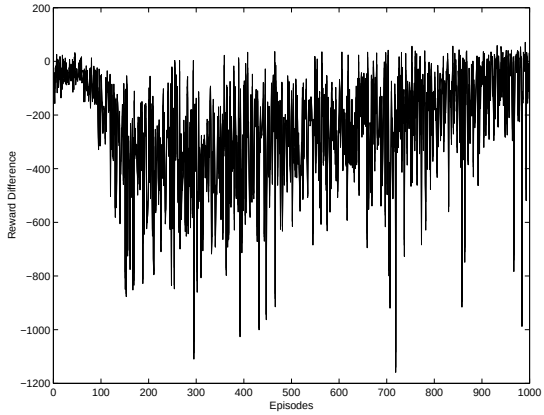

Fig. 6.16: No Trickle - No Knowledge, Reward Difference Curve

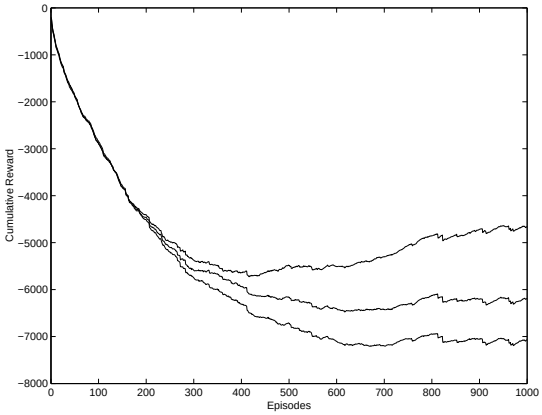

Fig. 6.17: Half Trickle - No Knowledge, Reward Curve

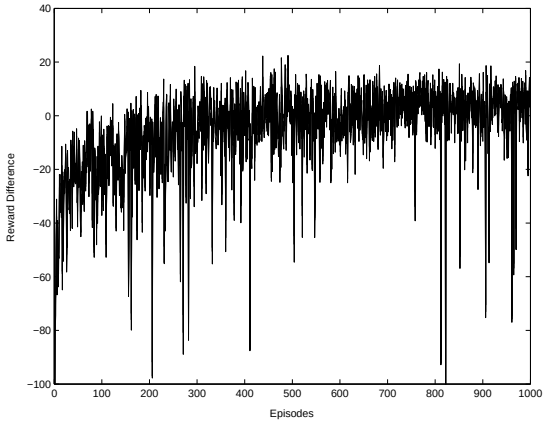

Fig. 6.18: Half Trickle - No Knowledge, Reward Difference Curve

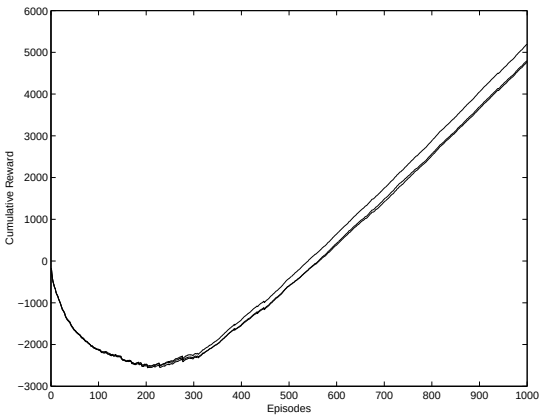

Fig. 6.19: Full Trickle - No Knowledge, Reward Curve

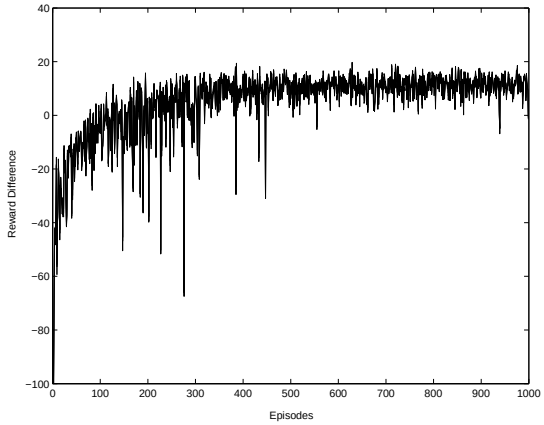

Fig. 6.20: Full Trickle - No Knowledge, Reward Difference Curve

can be seen in Figures 6.21 and 6.22 for No Trickle, Figures 6.23 and 6.24 for Half Trickle, and Figures 6.25 and 6.26 for Full Trickle.

6.2.2.2 Observations about the Reward Curves for Homogeneous Agents using Static Trickle

As expected, the cumulative rewards received by the agents increases as the benevolence increases.

In the No Trickle case where agents have no knowledge of each other, the same behavior changes discussed above are reflected in the reward curves. After the initial learning, the agents begin receiving less reward in each episode, then gradually over the thousand episodes begin receiving more reward as agents become less competitive.

In the case where agents have knowledge of others agents, the trend actually seems to be the opposite where agents continue to receive less reward in every episode, likely because they will strictly only receive reward when the other two agents are already in the goal.

However, it is interesting to note that the reward levels in the case of no knowledge are more than twice as low as the reward in the case of full knowledge. This is definitely a result of the state space being larger, because by having a much larger state space, agents are able to distinguish when the other agents have moved into the goal and will proceed to move into the goal to finish the episode. While the agents without knowledge are purposely staying out of the goal as long as possible and are blindly, or rather randomly, choosing to step into the goal. So, the explanation may lie in the fact that in the no knowledge case, three agents have to randomly step into the goal, whereas in the full knowledge case, only two agents have to randomly step into the goal and the third will learn to move into the goal.

The other interesting observation is in the case where agents have no knowledge of other agents, and use the Half Trickle reward distribution, as seen in Figure 6.17. The three agents' rewards diverge from one another. This can be explained by the fact that agents are receiving more reward if they are winning the episodic reward, so there is still some element of competition. In each separate trial of 1000 episodes, when looking at which agent wins each episode, there is an indication that in each trial

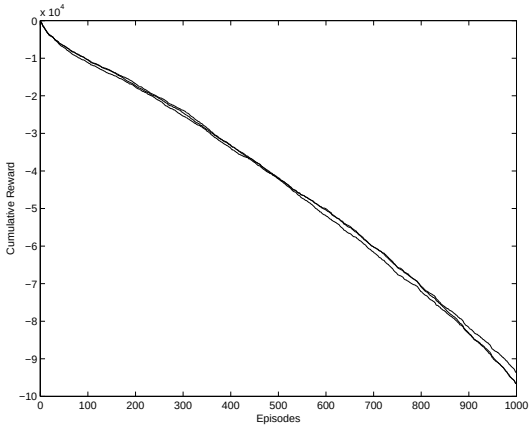

Fig. 6.21: No Trickle - All Have Knowledge, Reward Curve

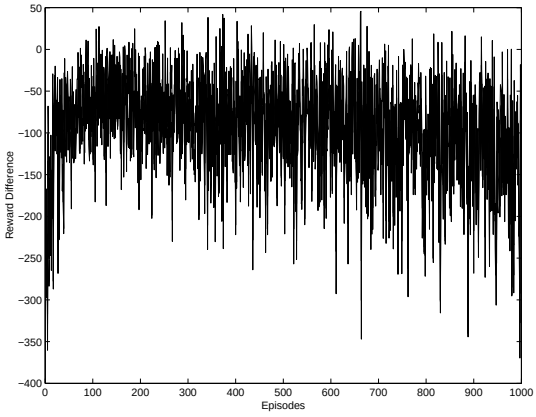

Fig. 6.22: No Trickle - All Have Knowledge, Reward Difference Curve

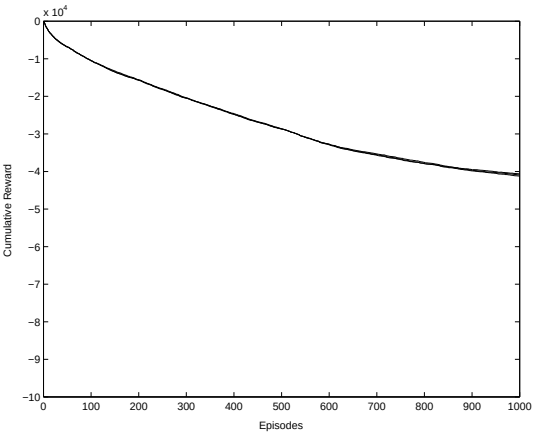

Fig. 6.23: Half Trickle - All Have Knowledge, Reward Curve

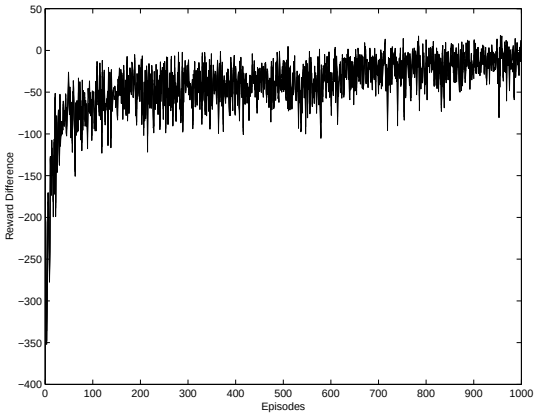

Fig. 6.24: Half Trickle - All Have Knowledge, Reward Difference Curve

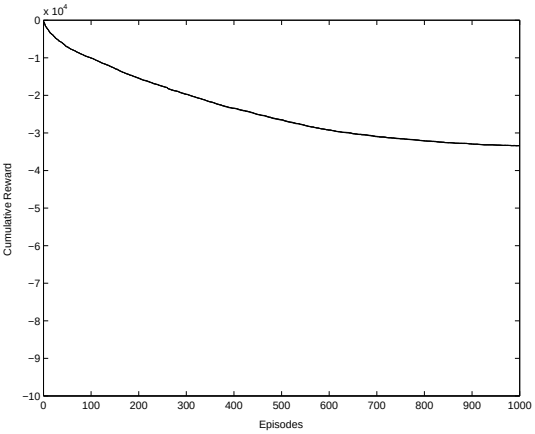

Fig. 6.25: Full Trickle - All Have Knowledge, Reward Curve

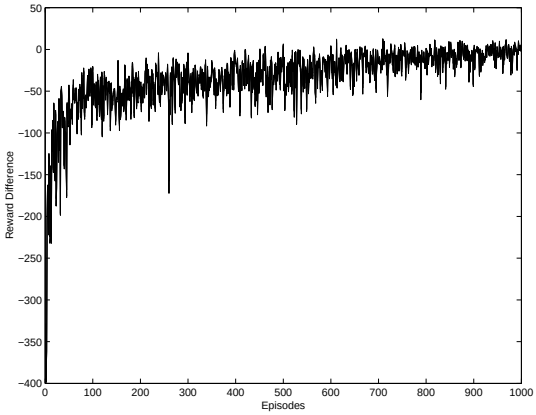

Fig. 6.26: Full Trickle - All Have Knowledge, Reward Difference Curve

certain agents can become more dominant than others by learning a policy which will get them into the goal slightly after the other two agents on a regular basis. This result comes from a set of 12 separate trials, but the three separate agents' reward curves would likely converge if more trials were done. The only reason this divergence is more obvious than other cases may be a result of the scale of the graphs.

Using just three trickle levels, the expected utility landscape in the case of homogeneous agents, as can be seen in Figure 6.27, seems to be a curve starting out very low at low benevolence levels and ending high at high benevolence levels. A more thorough cross-section of various trickle levels could prove useful, but the general curve can be inferred from this current data. Most likely there is a huge jump in the very low benevolence values and then a fairly linear relationship between the benevolence level and the expected utility.

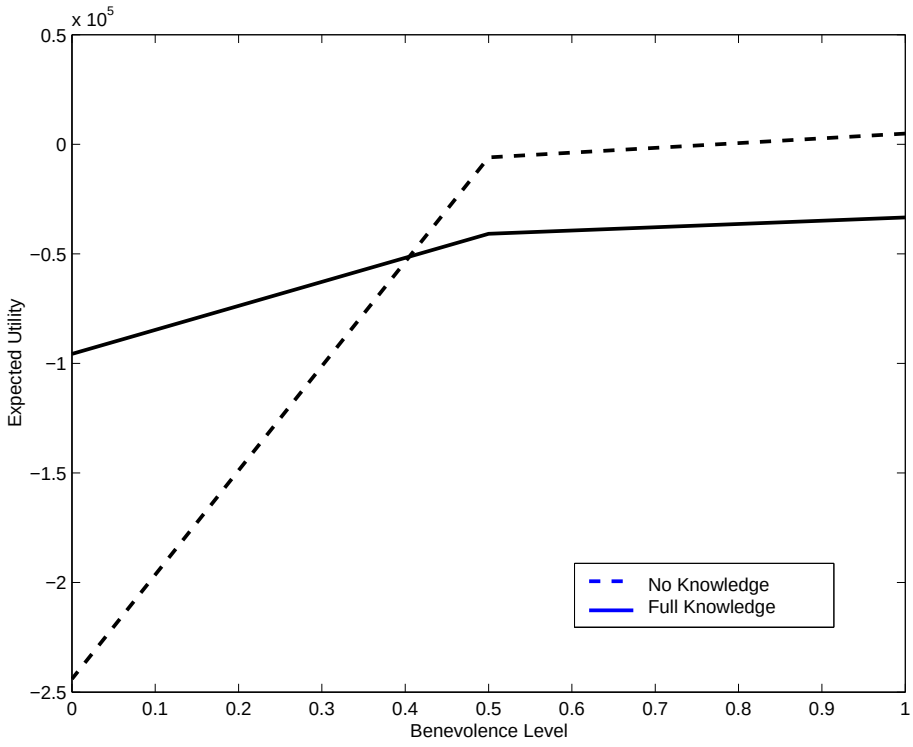

Fig. 6.27: Static Trickle - Homogeneous Agents, Expected Utility vs Benevolence

6.2.2.3 Reward Curves for Heterogeneous Agents

In the case of the heterogeneous agents, an interesting investigation is whether the more capable agent might benefit greatly over the other agents by only giving them enough reward to encourage them to enter the goal. If no reward gets distributed, agents will think no better of moving into the goal than any other action, so the episodes will end only when the agents randomly stumble into the goal and get stopped. Whereas, if the agent were to give the other agents a small portion of the reward, they might be able to learn to effectively move to the goal, because some reward is better than no reward, so the amount of reward given relative to the amount of reward kept by the agent might not matter as much in this case.

We hope to see that the more knowledgable agent will purposely resist stepping into the goal before the other two agents. The reward for choosing actions that keep him out of the goal should actually be higher unless the other two agents have entered the goal.

In hopes of more accurately mapping the expected utility landscape, additional benevolence levels were chosen to test how well the agents and the group performs and how much reward they receive. The six levels used are 0%, 10%, 25%, 50%, 75%, and 100%. The reward curves and reward difference curves for the six different benevolence levels in the case where one agent has knowledge of the other agents can be seen in Figures 6.28 and 6.29 for 0% Trickle, Figures 6.30 and 6.31 for 10% Trickle, Figures 6.32 and 6.33 for 25% Trickle, Figures 6.34 and 6.35 for 50% Trickle, Figures 6.36 and 6.37 for 75% Trickle, and Figures 6.38 and 6.39 for 100% Trickle.

The most interesting observation to be gained from this data confirms the theory put forth. An agent can expect to receive more individual reward in the long run if it gives away a small portion of its immediate reward. There is a dramatic improvement in the amount of expected reward for both individual agents and for the group when agents give just 10% of any reward received as opposed to giving no reward. The agent with more knowledge clearly has a distinct advantage over the other two agents and is able to win the vast majority of the episodes, so the other two agents have little motivation to move into the goal, unless they can expect to receive some reward from the winning agent.

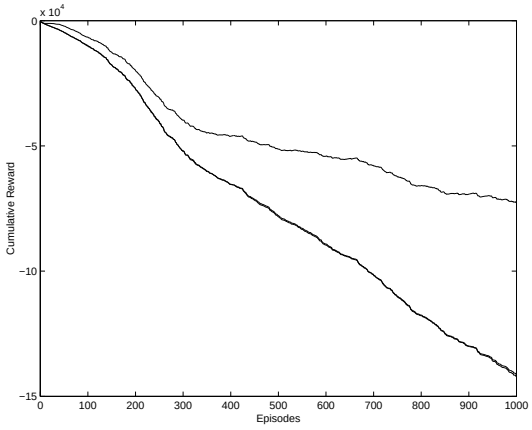

Fig. 6.28: 0% Trickle - One Has Knowledge, Reward Curve

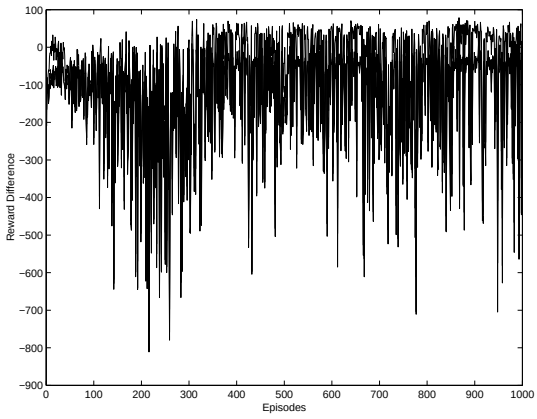

Fig. 6.29: 0% Trickle - One Has Knowledge, Reward Difference Curve

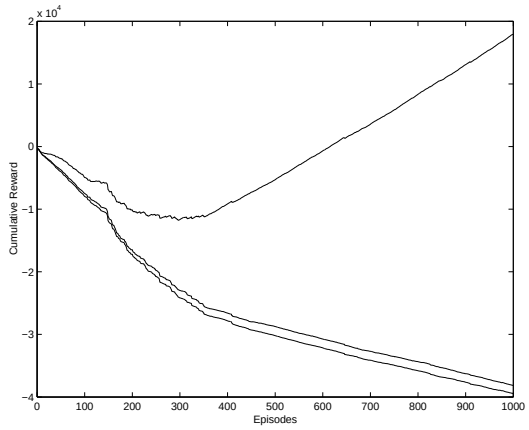

Fig. 6.30: 10% Trickle - One Has Knowledge, Reward Curve

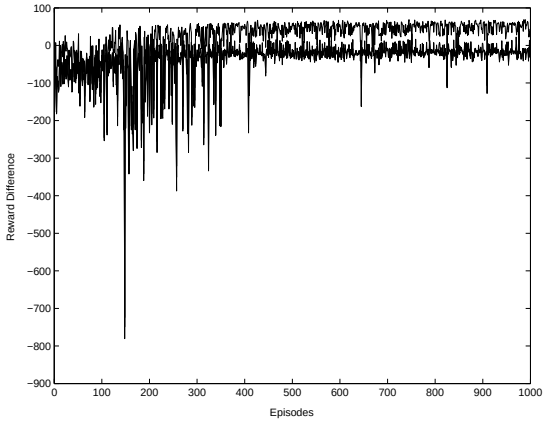

Fig. 6.31: 10% Trickle - One Has Knowledge, Reward Difference Curve

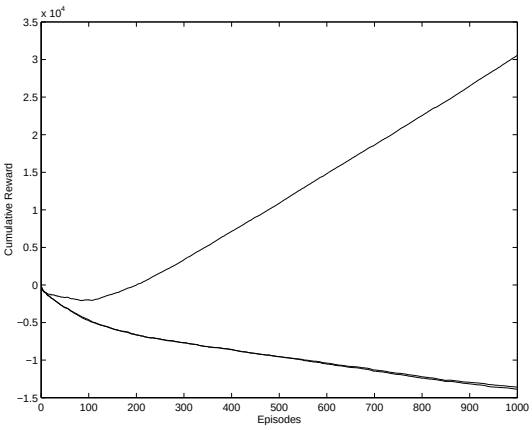

Fig. 6.32: 25% Trickle - One Has Knowledge, Reward Curve

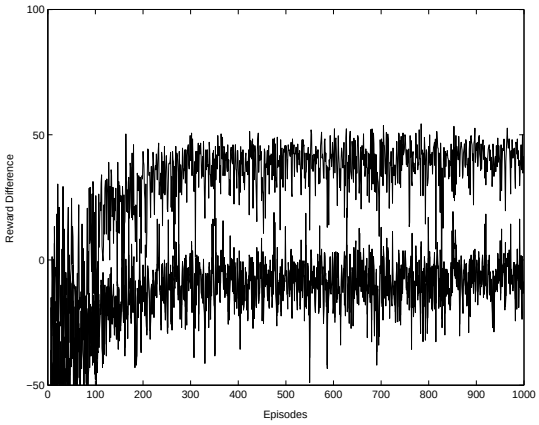

Fig. 6.33: 25% Trickle - One Has Knowledge, Reward Difference Curve

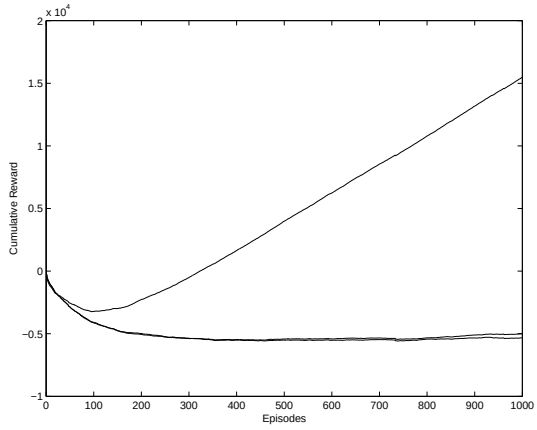

Fig. 6.34: 50% Trickle - One Has Knowledge, Reward Curve

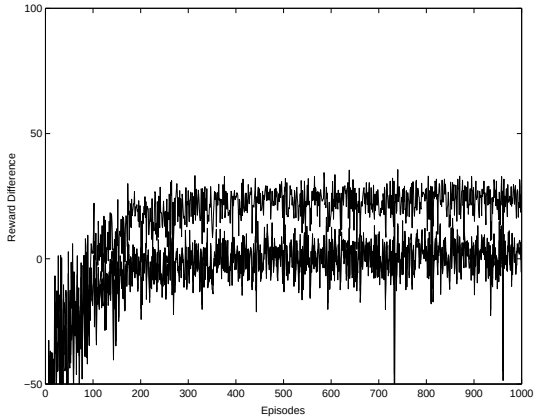

Fig. 6.35: 50% Trickle - One Has Knowledge, Reward Difference Curve

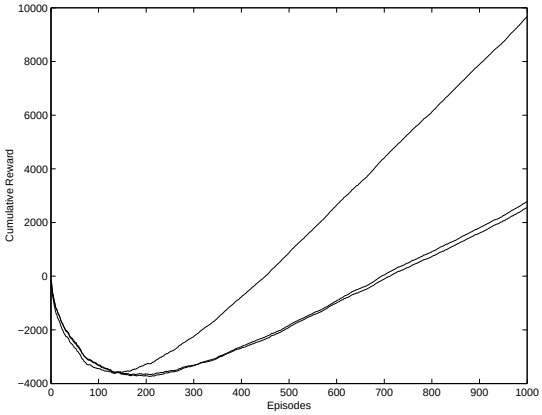

Fig. 6.36: 75% Trickle - One Has Knowledge, Reward Curve

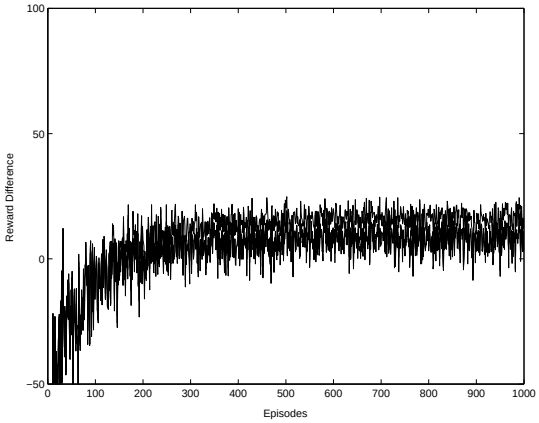

Fig. 6.37: 75% Trickle - One Has Knowledge, Reward Difference Curve

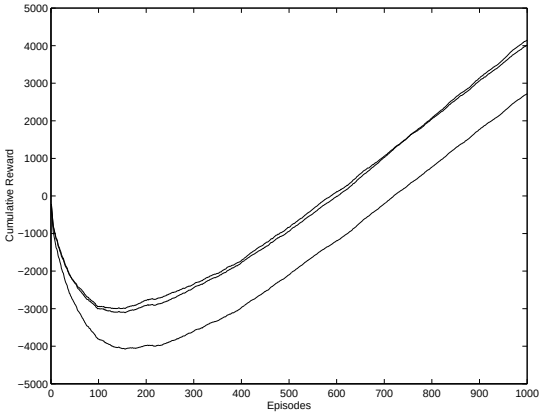

Fig. 6.38: 100% Trickle - One Has Knowledge, Reward Curve

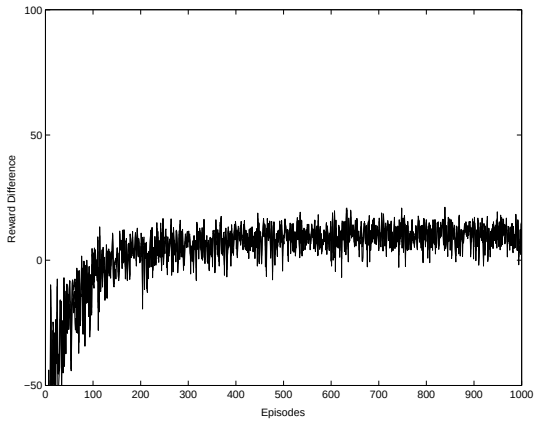

Fig. 6.39: 100% Trickle - One Has Knowledge, Reward Difference Curve

When plotting the expected utility of the individual agents, as can be seen in Figure 6.40, we find that the expected utility is different between the agents at the same benevolence levels. The two agents without knowledge have a similar shaped graph to that found in the case of homogeneous agents, whereas the knowledgable agent's graph favors the lower benevolence levels, but above zero.

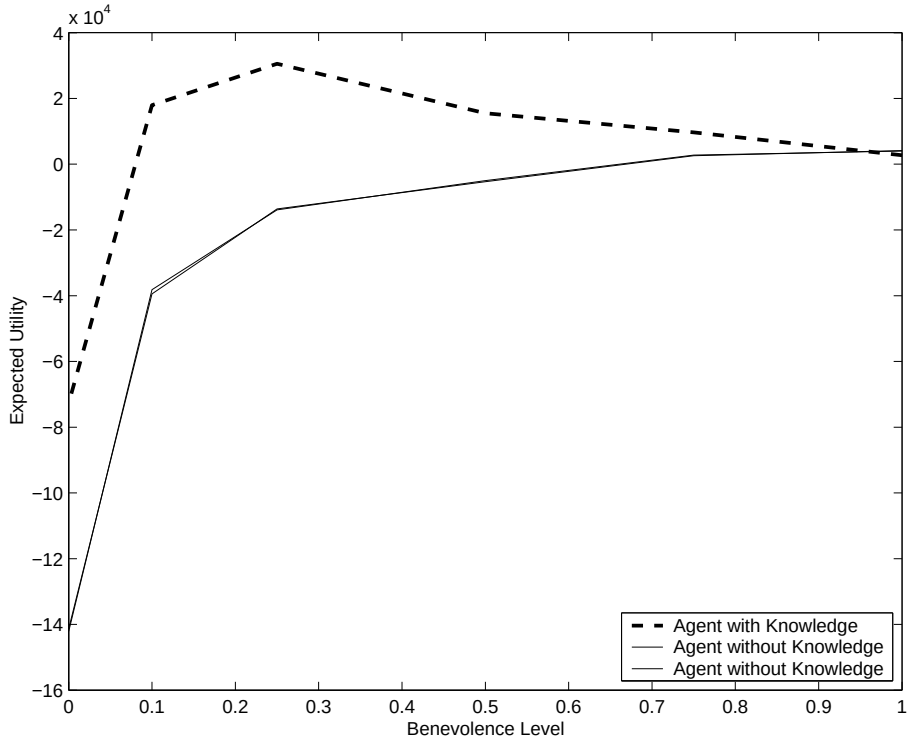

Fig. 6.40: Static Trickle - Heterogeneous Agents, Expected Utility vs Benevolence

When we look at the group cumulative expected reward, we see a curve similar to the homogeneous expected utility graph, although the expected utility for 75% is slightly higher than 100%, as seen in Figure 6.41. It is not clear if this is a significant result that there is higher group utility when heterogeneous agents use a benevolence level of 75% in this task. A more rigorous investigation involving many more trials could answer this question.

Many multiagent systems are designed with the intention that selfish agents should drive the system towards a global maximum, however this does not seem to be the

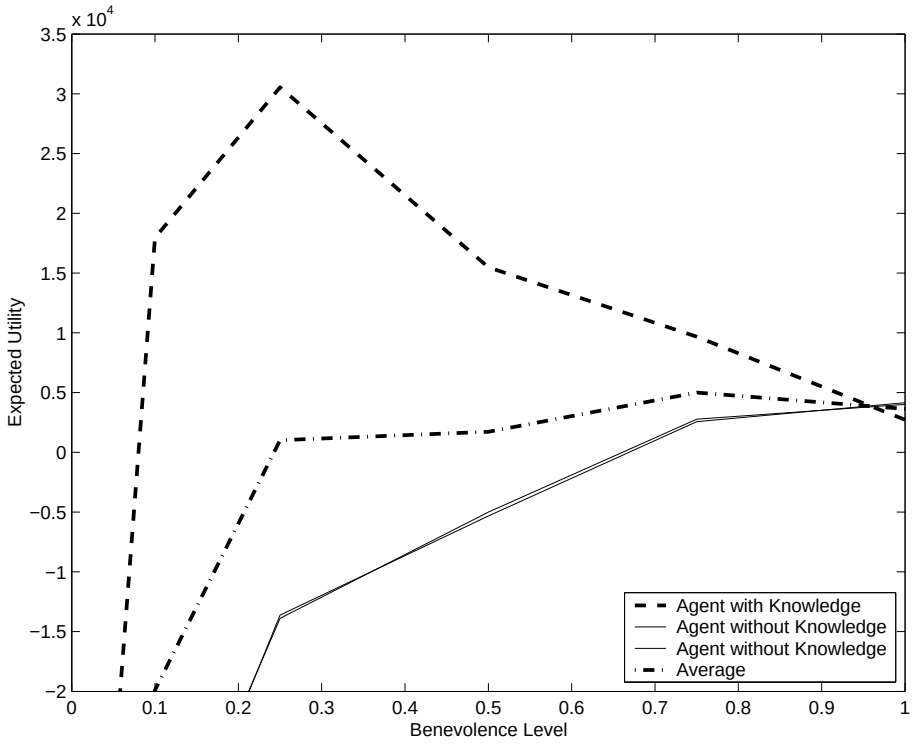

Fig. 6.41: Static Trickle - Heterogeneous Agents, Expected Utility vs Benevolence

case in this multiagent system that uses Trickle-Down Utility. If the knowledgable agent were able to dynamically pick its benevolence level, then it would favor a level of about 25% even though the group seems to do better with higher levels.

6.3 Variable Trickle

An investigation into Variable Trickle was started including some preliminary results, but lack of time prevented extensive effort towards getting the implementation to work.

The first implementation was altered slightly to choose the same benevolence level for ten episodes in hopes of allowing enough time for other agents to react to the reward given from one agent to another, but not waiting for too many episodes so that the preference levels have time to settle within 1000 episodes. This simple change resulted in the preference levels seen in Figure 6.42 in the case of agents with no

knowledge of others.

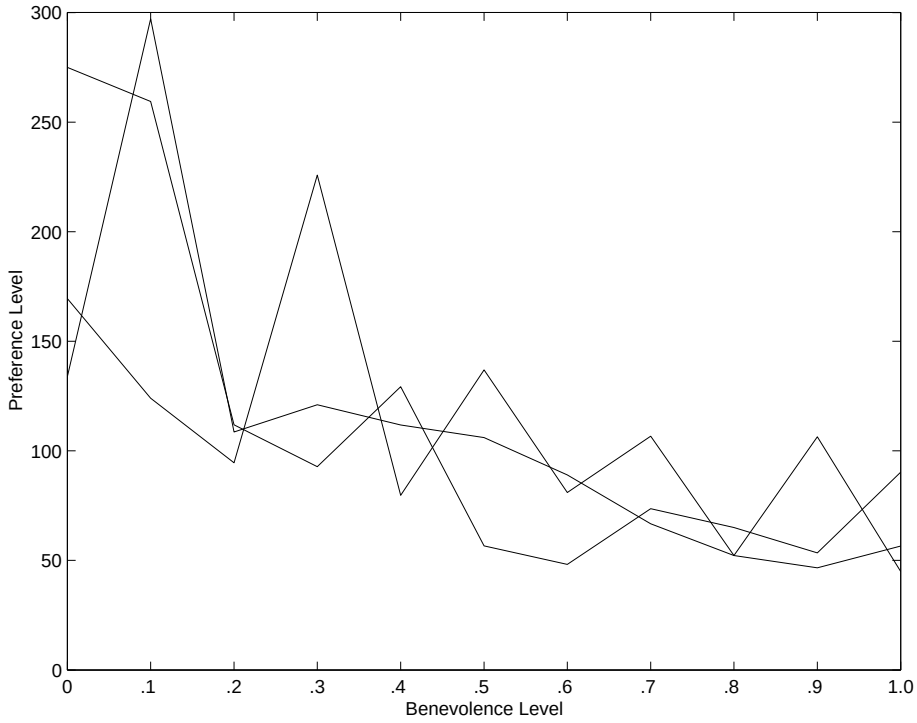

Fig. 6.42: Variable Trickle - No Knowledge, Preferences

Although, it does not look like the expected utility graph of homogeneous agents, it does show a general trend in the opposite direction, whereas the lower benevolence levels are generally more preferred than the higher benevolence levels. This makes sense when considering that this is still using the average reward and not an average reward difference as suggested in the previous chapter. If an agent gives away less reward, then, of course, his reward will be higher in those ten episodes than in the previous ten episodes or the average reward received in a given ten episodes.

A better implementation of choosing the appropriate level is necessary to show that the variable trickle case is beneficial for an agent to use. The next logical step is to try changing what gets updated in each step, such as using the reward difference instead of the actual reward received in that step. This is promising when looking at the reward difference curves, because the agents receive the most reward when the reward difference is at its highest. In the case of the knowledgable agent, the final value of the

reward difference curve is higher in the 10% case than the 25% case, even though the cumulative reward is higher in the 25% case, but this is a result of slower learning in the 10% case. Perhaps, this suggests that an agent could learn to give more reward in the beginning of a trial to encourage faster learning, then gradually reduce the amount given in order to optimize individual reward even more. Towards this end, it would be useful to map the expected utility landscape with a third dimension over the number of episodes that have been experienced.

Another idea was to change the actions to be: raise benevolence, lower benevolence, or keep benevolence the same. The agent could move the benevolence up or down by a specified amount, such as 5% or 10%, and wait for a set number of episodes to see if there is a positive or negative reaction to the change in benevolence levels. But this too will have to deal with the issue of receiving more reward just because you aren't giving away as much, as opposed to receiving more reward in the long run. And also, whether or not this is enough time to notice any difference in how the other agents react. An eligibility trace of some sort may be useful since the system is effected by all the previous benevolence levels, not just the current level.

There are a large number of other questions about the system dynamics that need consideration, but this is left for future work in this topic.

Chapter 7

Discussion

In this chapter, we go back to our original investigation questions, Section 3.2, and use the results found through this research, as described in Chapter 5 and 6, to see what kind of insight we can find. Then we will discuss the significance of this research. And finally, we will consider a number of future directions and further investigations to be done in this topic.

7.1 Investigation Questions

7.1.1 Question 1: Static Trickle

7.1.1.1 How does the performance of Full Trickle compare to Team-Game Utility?

In this research, it is evident that Full Trickle Utility performs just as well as Team-Game Utility in tasks where TGU is the best method to use. The only difference between these two methods is the source of the reward, whether an agent will receive reward from the system or from another agent. But if the amount is the same, then there is no difference as far as the learning is concerned, so the performance is identical in these two methods.

The Trickle Utility method has the advantage over TGU of being able to dynamically allocate which agents should be able to receive reward, so as to only reward those agents responsible for accomplishing the goal. Essentially proving that the Trickle

Utility is a more flexible system to use, while still producing comparable results to TGU.

7.1.1.2 In what kind of task is the optimal solution going to be for agents to be entirely benevolent, entirely competitive, and partially benevolent to some agents?

A task that requires agents to be entirely benevolent is a task where all agents are equally essential towards completing the task, such as the task used in all of my experiments. If each agent's individual utility is dependent on the actions of other agents, then each agent will perform optimally only by sharing all reward equally.

A task that benefits from agents being entirely competitive is a task where only one agent is necessary to complete the task. If the same sort of task were used, where agents needed to find the shortest path to the goal, then by rewarding only the first agent to reach the goal, the entire group will learn to move as quickly as possible to goal, because it will be the only source of reward so agents will actively compete against one another. If agents were to share this reward, then agents might not learn the optimal policy since there is not as much pressure to reach the goal. So, by having competitive agents, the entire system of agents might actually perform better than by using cooperative agents.

A task that benefits from agents being partially benevolent would be a more complex task that involves heterogeneous agents. An example might be designing a football team, which consists of fully customizable agents, some of which will be physically and/or mentally superior to the other agents. These 'superstars' will deserve more reward for their actions and their teams may actually benefit more by allowing them to keep their 'glory'. Those agents with the most cumulative reward across all the teams will get be recognized as the best football playing agents and could compete in an 'all-stars' competition.

7.1.1.3 How will the results vary between homogeneous agents and heterogeneous agents?

It seems evident, that when using homogeneous agents, it will only be beneficial to use either benevolent or competitive agents, depending upon the task. There may never be an appropriate use of a partially benevolent agent. As found in the case where one agent had more knowledge than the other two agents in the revised experiment, that agent benefitted greatly from being partially benevolent, giving 25% of his reward, because this provided the other two agents with enough reward to encourage them to move into the goal as quickly as possible, while still allowing the knowledgable agent to gain the most reward.

However, the system utility was found to be optimal when the knowledgable agent was mostly, if not fully, benevolent, so the question is raised whether this method is the best method to use in the case of heterogeneous agents. Perhaps a more complex method such as the Wonderful Life Utility [24] would produce better results, because the knowledgable agent would acknowledge that he is only worthy of one third of the reward and the system would act as if it were a team of benevolent agents.

7.1.2 Question 2: Variable Trickle

7.1.2.1 What is the best technique to use to find the optimal benevolence level?

It remains to be seen, what the best technique would be to find the optimal benevolence level. The technique I implemented using reinforcement comparison where the actions were various possible benevolence levels is rather ineffective. By constantly changing the benevolence levels randomly, the system gets confused as agents try to adjust to the amount of reward they are receiving for their actions.

A better method would involve a gradual ascent towards the optimal value, and need not even require any form of reinforcement learning. An agent could initially start out being partially benevolent, perhaps 50%, to any agent it hasn't encountered before, and then gradually increase or decrease the benevolence level and observe the effect on the system. Since the benevolence levels of the other agents are affected by all the previous benevolence levels, not just the current one, the system will respond

more predictably by gradually changing the benevolence levels.

There is still the issue of how accurately the system can be observed because it can be assumed that the other agents are also dynamically changing their benevolence levels. An agent could model what the other agent's benevolence levels are based upon how much reward they give out, and then this can be taken into account when determing how benevolent to be towards the other agents in the future.

7.1.2.2 How quickly can the benevolence level converge to the optimal value?

The benevolence levels cannot change too drastically too fast, so there will definitely be some large delay before the benevolence level can converge to the optimal value. It would likely take at least ten episodes for an accurate observance of any measurable change in the system, and it may take at least ten gradual changes to reach the optimal value, so it may take a few hundred episodes to be able to comfortably settle on a final value.

7.1.3 Question 3: Agent Interactions

7.1.3.1 How are the benevolence levels of one agent affected by the benevolence levels of the other agents?

An interesting investigation that would have been conducted if the variable benevolence levels were able to be found, is to fix one or two of the agents benevolence levels and allow just one of the agents to dynamically choose his benevolence levels towards the other two. In this, we might have shown the inter-dependence of one agent's benevolence levels to the other. But this can be left to future work.

7.1.3.2 In the case of heterogeneous agents, will more capable agents become more competitive than the others?

In finding the expected utility landscape of heterogeneous agents, it is clear that the more knowledgable agent would become far more competitive than the other agents if he were able to choose his own benevolence level, because at low levels his cumulative reward was far higher than at any other benevolence level. He is capable of winning

the vast majority of episodes, so the other agents would actually learn that they will receive more reward by allowing him to win the episodes and letting him distribute some amount of reward to them.

7.2 Research Significance

7.2.1 How effective is this method?

This method is shown to be better than Team-Game Utility, so it can be used in its place whenever TGU can be used effectively. But it can also be used in situations where TGU cannot be used effectively. Although, as intended, it does not take into account any game theory ideas, in order to try to minimize computation and knowledge of the system.

This may not be the best method to use when trying to optimize system utility, but it may be the best method to use when trying to optimize individual utility. This method suffers from what is described as the "tragedy of the commons." But in certain applications, this might be a desirable method to use. For example, when one is designing a computer role-playing game, with AI agents as many of the supporting characters, these characters are meant to emulate humans, who may in fact care more about their individual needs rather than society as a whole. So designing this MAS using Trickle-Down Utility, may in fact create the most realistic virtual society for the players to interact with. A society that follows game theoretical concepts for finding a solution where all members are satisfied is rarely found in the real world, so why should we try and create one in the virtual world that is meant to emulate the real world. Creating realistic interacting agents is the ultimate goal of AI in computer games, so Trickle-Down Utility could be a promising method to use.

7.2.2 How will this scale?

This method could scale well, if an agent were aware of all the agents in his local neighborhood, and was able to store the benevolence levels for these agents based on previous experiences. Enough knowledge of the local environment would be necessary,

but global knowledge is not necessary and is undesirable and realistic for creating human-level AI. The agent would only need to consider distributing reward to those agents locally accessible that may or may not have been partially responsible for accomplishing the desired goal. Benevolence levels of agents encountered can be stored so that future interactions can continue using the previously learned levels. This likely will scale better than Team-Game Utility and should scale as well as the Wonderful Life Utility, because each agent is always only taking into account information that is locally available and does not ever need to consider itself amongst the overall system [1].

7.3 Future Directions

There are numerous directions to be taken using this type of implementation. Whether to improve the techniques themselves, or to explore various aspects of agent interactions within multiagent systems. All of these ideas mentioned were part of the overall plan, but are left to be looked at in future work.

7.3.1 Possible Changes in Implementation

7.3.1.1 Improved Learning

Additional effort could be put towards encouraging agents to learn faster. Perhaps, the state space could be changed slightly to reduce computation even further. The learning parameters could be made to decay in a more effective manner, so as not to need a lower threshold limit. In addition, it might be interesting to investigate the benefits of adding an eligibility trace to the Q-learning, because the current reward is an effect of the previous few state-action pairs and may encourage faster learning.

7.3.1.2 Independent Agents

One element of the implementation that I wasn't terribly satisfied with was the source of information for the agents. It would be more idealistic if the agents were able to observe the other agents as if situated in this virtual world. Instead of receiving

their coordinates from the system to compute their proximity to their goal, and instead of receiving the proximity of other agents to their goals from the other agents, the agent could ideally be able to localize itself and model the necessary state space information solely through its sensors into the world, emulating a robot explorer. So if another agent was on the other side of a hill, and the agent had no way of detecting him beyond visual contact, then that should affect the knowledge held about that agent. This approach is known as sensory honesty, and would bring computer game implementations more in league with using human-level AI [8].

7.3.1.3 Reward ID

Another issue needing consideration is how to identify a reward. If this method gets used in a more complex system, there may be a situation where an agent will receive reward and pass on part of it to other agents, and then from that reward he passed, some agent passes reward back to him. Do we want to allow this potentially infinite cycle of reward passing to continue?

One idea is to only pass reward to others if its is above some specified value, such as a reward of 10. Only if it is above this does an agent consider splitting it and passing it on. Although, this feels more like an implementation hack than a legitimate criterion.

Another idea is to contain within every reward an identity, like a packet number in a network routing problem, with the intention that an agent will never pass the same reward to the same agent. Or, perhaps, give the reward an age that decays over time.

In this implementation, the problem is bypassed by only redistributing reward when the reward is received directly from the goal, which means it is only ever split and given to other agents once, by the last agent who enters the goal. Although, an identity field has been built in that can later be utilized for a more effective method of deciding whether or not to redistribute a given reward.

7.3.2 Future Experiments

7.3.2.1 Different Actions

A slight variation on the current implementation is to compare the performance of heterogeneous agents that have different actions available to them. The easiest implementation would give one agent the eight action as they were used in these experiments, and the other two agents could have only the four primary cardinal directions. Or, perhaps, allow one agent to move twice as fast as the other two agents. The idea is that if one agent is physically more capable, would he become more competitive because he would be able to win more episodes than the others? On top of that, we could experiment with giving these agents both different amounts of knowledge and different actions.

7.3.2.2 Extra Agents

The next major experiment I was planning to try after getting the variable trickle implementation to work is to add additional agents into the trial, but keep the task the same, where only three agents are needed to complete an episode. This creates a situation where every agent is not essential to the task, and so the Trickle method would prove much more useful than the Team-Game Utility method, because with Trickle, agents can learn to give less reward to agents that are performing worse. Agents may become extra competitive to reach the goal in order to receive any reward and in doing so the performance of the entire system might improve as episodes finish much faster. With this new data it would be interesting to see how the expected utility landscapes differ from the previous experiment for both homogeneous and for heterogeneous agents.

7.3.2.3 More Complex Task

Another major experiment that was envisioned is to create a more complex task that involves a sequence of subtasks to be completed before the group receives episodic reward. One such task would be a relay race, where agents need to move a baton from one end of the track to the other and pass it on to other agents for them to complete

their leg of the task. Once the last agent reaches the goal he must pass reward back to the other agents, and the reward would trickle back to all of the agents. This would be a linear redistribution of reward, so if it is the only source of reward then the actual amount isn't as critical since it is only relative to that agent.

But the next task could be one where agents could choose to cooperate with one of a group of agents in each episode and he will learn which of these agents is more likely to give him more reward. Or perhaps it could take two agents to help one agent, and that agent must pass reward to both agents. These two situations would create more networked redistributions of reward that are more effective for the complex systems that this technique would be useful for.

7.3.2.4 Conflicting Goals

A task could be developed where agents could have conflicting goals and the group might end up working at cross-purposes. One such task might have a large number of goals, and agents need to go to as many goals as possible and place a marker or raise a flag of some sort to indicate that they have been there. But only one marker can be at each goal, so if another agent comes to the same goal, he will replace the marker with his own. Agents are then rewarded for the number of markers they have. Cooperative agents might spread out to cover all of the unmarked goals in the time allowed, emulating a foraging task. Whereas, a very competitive agent might follow other agents and mark a goal right after an agent has marked it to ensure that he always has more goals marked than the other agent.

Chapter 8

Conclusion

This investigation set out to examine the effectiveness of Trickle-Down Utility, a novel multiagent reinforcement learning utility function. To test this, three agents needed to learn to reach a goal as quickly as possible, where the third agent to reach the goal receives the reward and must distribute some of the reward to the other agents to encourage them to keep trying to accomplish their goals.

In the first set of experiments, the agents receive negative reward when outside of the goal, and zero reward when inside of the goal, which makes agents move into the goal as quickly as possible and then they might jump in and out in hopes of receiving the positive reward for being the last one to enter the goal. The second set of experiments forces agents to stop moving once inside the goal and they continue receiving negative reward until all of the agents enter the goal. This creates a task where agents' individual utility is much more affected by the actions of the other agents, which is more conducive towards agents learning to cooperate. In both experiments, agents are programmed with various static benevolence levels so that the performance of the different systems can be compared.

An investigation into the feasibility of establishing variable benevolence levels did not produce a working implementation, due to time constraints. Several factors were suggested towards the cause of failure, including a few suggestions towards what could be done to produce a working method.

The results from these investigations show that an agent does perform better when being fully benevolent in the case of homogeneous agents, and when being

partially benevolent in the case of heterogeneous agents. It should be possible for agents to dynamically learn this level once an working method for finding the optimal benevolence level is implemented.

It has been shown that Trickle-Down Utility performs equally to Team-Game Utility, and it should have the flexibility to be effective in situations where TGU is ineffective, once the variable benevolence levels are implemented. In addition, Trickle-Down Utility does not incorporate game theory concepts such as the Wonderful Life Utility, in a purposeful attempt to see how well a system can perform with agents that have limited knowledge.

This research could prove useful towards designing multiagent systems, by establishing benevolence levels as a numerical parameter. Previous multiagent systems utilized benevolent or competitive agents, which are the two extremes when using benevolence levels. For many tasks, fully benevolent or fully competitive agents are likely to be the optimal solution, but for complex tasks or tasks involving heterogeneous agents, partially benevolent agents might prove useful. By allowing agents to dynamically select their own levels, we would be able to show that fully benevolent or competitive agents are in fact the optimal configuration for certain tasks.

Designing a MAS of selfish agents, who are not driven to maximize world utility, but only individual utility, is more reflective of a human society, and so would be a more beneficial method to use in designing a society of AI game agents. Agents will only work towards the benefit of others if it is also towards its own benefit.

An example game that could utilize this method could be an adventure game similar to Tomb Raider [4], where a group of treasure-seeking adventurers, a human player and several AI agents, enter an ancient tomb and compete against one another to collect as much treasure for themselves. This tomb may include complex puzzles or doors that could involve multiple agents to unlock, so by learning to cooperate in certain situations, these agents might receive more reward by working with other agents than by working alone. An element of the game may be for the player to be able to give treasure to these other agents in order to influence their actions, whether by sharing any treasure they find together to encourage future assistance, or by deceiving them by placing jewels to lure competitors towards empty or perhaps dangerous rooms.

Bibliography

[1] Adrian K. Agogino and Kagan Tumer. "Team Formation and Communication Restrictions in Collectives". *Proceedings of the Second Interntional Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS03),* (916-917). Melbourne, Australia. July, 2003.

[2] Blizzard Entertainment. *Starcraft*. http://www.blizzard.com/starcraft.

[3] Rodney A. Brooks. "Intelligence without representation". Artificial Intelligence 47, (139-159). 1991.

[4] Eidos Interactive. *Tomb Raider*. http://www.eidosinteractive.com.

[5] Owen Holland. "Multiagent systems: Lessons from social insects and collective robots". AAAI Spring Symposium on Adaptation, Coevolution and Learning in Multiagent Systems, Stanford, 1996.

[6] Junling Hu and Michael P. Wellman. "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm". *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98),* (242-250). Madison, WI. July, 1998.

[7] ID Software. *Quake 3 Arena*. http://www.idsoftware.com.

[8] Damian Isla and Bruce Blumberg. "New Challenges for Character-Based AI for Games." *AAAI Spring Symposium on AI and Interactive Entertainment*, 2002.

[9] John E. Laird and Michael van Lent. "Human-level AI's Killer Application: Interactive Computer Games". AI Magazine, 22(2): (15-25), 2001.

[10] Lionhead Studios, Electronic Arts. *Black & White*. http://blackandwhite.ea.com.

[11] Michael L. Littman. "Markov games as a framework for multi-agent reinforcement learning". *Proceedings of the Eleventh International Conference on Machine Learning*, (157-163). San Francisco, CA. 1994.

[12] Yishay Mor and Jeffrey S. Rosenschein. ”Time and the prisoner’s dilemma,” *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, (276-282). Menlo Park, CA. AAAI Press, June 1995.

[13] John F. Nash. “Non-cooperative games”. *Annals of Mathematics*, 54: (286-295). 1951.

[14] ORIGIN. *Ultima Online*. http://www.uo.com.

[15] Sandip Sen, Neeraj Arora, and Shounak Roychowdhury. “Using limited information to enhance group stability”. *International Journal of Human Computer Studies* 48(1): (69-82). 1998.

[16] Sony Online Entertainment. *Everquest*. http://everquest.station.sony.com.

[17] Peter Stone and Richard S. Sutton. “Scaling Reinforcement Learning toward RoboCup Soccer”. Proc. 18th International Conf. on Machine Learning, (537-544). 2001.

[18] Peter Stone and Manuela Veloso. “Multiagent Systems: A Survey from a Machine Learning Perspective”. *Autonomous Robotics*, Vol 8, Num 3, July, 2000.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998

[20] Torque SDK. http://www.garagegames.com.

[21] Kagan Tumer, Adrian K. Agogino, and David H. Wolpert. “Learning Sequences of Actions in Collectives of Autonomous Agents”. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, New York, NY, 1999.

[22] Valve Software. *Half-Life:Counter-Strike*. http://www.counter-strike.net.

[23] Chris Watkins and Peter Dayan. “Q-Learning”. *Machine Learning*, 8, (279-292). 1992.

[24] David H. Wolpert and Kagan Tumer. “An Introduction to Collective Intelligence”. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center, 1999.