Creating Complementary Cooperative Agents Using an Opinion System

by

Alan Taylor, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

MASTER OF SCIENCE (Interactive Entertainment Technology)

University of Dublin, Trinity College

September 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Alan Taylor

September 9, 2008

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Alan Taylor

September 9, 2008

Acknowledgments

I would like to thank my supervisors, Mads Haahr and Saturnino Luz, and all my lecturers who taught me so much during the last year. To Steve for creating a fantastic course. To all my classmates that spent the long hours working alongside me. To my parents for everything. Lastly to Chelle for all her support especially during all the stressful times.

Alan Taylor

University of Dublin, Trinity College
September 2008

Creating Complementary Cooperative Agents Using an Opinion System

Publication No. _____

Alan Taylor, M.Sc.
University of Dublin, Trinity College, 2008

Supervisor: Mads Haahr

Video games that utilise companion AI to enhance their core game play are becoming more common. Specifically these games give the player a second, AI controlled, cooperative agent to play the game with. Even though developers are overcoming some of the problems that plagued development of such games, the companion's AI is still a source of frustration for most players. Their behaviour can often seem unbelievable, which can leave players wishing the companion was not a feature of the game. This dissertation presents the idea of an opinion system that can be incorporated into certain video games to improve the simulated cooperative intelligence from the companion in how it plays in order to improve the enjoyment a player gets from having an AI companion in their game. This opinion system allows an artificial agent to form an

opinion, internally, of the player in order to improve it's decision making principles, i.e. *make decision X when health is below Y*. The companion will also respond to, and help, weaker gamers to make them feel more comfortable while playing the game. Through the completion of small scenarios and observed player behaviour, the player will be given a companion that is unique to them. The companion's final playing style will reflect and be complementary to the player's own playing style.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The aim of this dissertation is to investigate a method of creating Artificially Intelligent (AI) agents that coordinate their action based on the decisions of the player during a video game. The intended results are that from a small amount of user input and constant monitoring of the player, an opinion of the player will be formed in the agent. Using this opinion to extend their AI, the agent will produce a much more believable decision making process. This extension to their AI will also result in a companion that is far more complementary to each player's playing style. It is also hoped that this complementary AI will create cooperative companions that are more suited to weaker players of games to increase their confidence in playing games.

The main difference between a cooperative video game and single player video games is the addition of a second character. This second character can be controlled by either a human or AI. The addition of the second character is intended to improve the enjoyment received from a single player game. The creation of cooperative games is known to be a very difficult task in the video game industry and has not had the best track record [8]. The scripting of events and the overall mechanics of co-operative games are much more complicated than single player games. A game scripted and engineered for the enjoyment of a single player becomes much more difficult to script when a second player is introduced [8]. Applying enough focus on the cooperative agent is essential to making them feel integral to the overall game. This can include

making them a part of the story line so players grow attached to them as they would their own player or even giving them a purpose for being there, e.g. introduce a game play mechanic that requires two people to work together to complete. This reduces the burden players feel from having to carry these companions for the duration of the game and increases overall enjoyment. Developers have often found it difficult to introduce a co-operative element to their game and in some cases [9] have had to drop the feature. "We started off being so arrogant that we said "oh co-op will be easy, but then at every turned is has f****d us" [8]. But recent and upcoming games, e.g. Gears of War [10], Mass Effect [11], Left 4 Dead [12] and Resident Evil 5 [13] seemed to have solved the problem. Making a companion feel integral to the game, giving them a purpose and a function, ultimately reducing the player's desire to just play the game solo.

Often co-op games are played online or with two players on the same machine. But it has been shown shows that while some players prefer to the social interaction and intelligent behaviour that comes with playing with human players there are another set of players who find it more convenient to play with computer controlled characters [14]. So for players that do not have an extra player to play with, a lack of an online connection or who just have a preference to play with a computer controlled player means that developers need to write AI behaviour for the second player to fill in for the absence of real human intelligence. What players often actually get are co-op companions that are following a set of pre-defined behaviours that always perform the same action in the same situation [2]. While the companion that players are given usually has very high intelligence its action decision principle is constrained to how the developer has programmed it [15]. A general action decision principle is used to determine which action an agent should make next, usually based on what they know about their environment and their current state. While these elements used to make decisions are perfectly acceptable in predictable decisions, at times when faced with unanticipated scenarios the agent can act in ways that players would find irrational [15]. A proposed idea is that the companion should incorporate the player as a main element in its decision making principle and learn to adapt to each player's playing styles.

1.2 Objectives

Developers cannot anticipate how agents should behave in all situations in order to satisfy the expected reaction from a player. Developers have used player testing to cover all rules in agent behaviour to achieve success in agents producing rational actions in all given situations. I look at some of the situations that are most often encountered in action games like Gears of War [10] and use them as the basis for asking the user to help teach the agent to stay rational in its behaviour. To try and reproduce some of the team work elements involved in co-op games I have proposed to add teaching scenarios to the beginning of a game where a player can train their own AI companion as well as implementing observation techniques to adjust other elements of the companion's behaviour.

The scenarios are scripted and prompt players to select an input from a GUI to determine how they want to co-ordinate with their companion. The teaching portion is short and akin to the training scenarios used in games like Splinter Cell: Double Agent [16]. It is felt that adding this scenario to the beginning of the game is of no extra annoyance to the player. [17] has shown that players would prefer to have some form of training added to the start of a game. Using the scenario it can show the player both how to play the game as well as gathering the required information that is needed from them. I allow the player to use varying game play styles to easily visualise the immediate change in the companion's playing style. The intended outcome is that the player is given an AI companion that is complementary to their playing style. I will show that by teaching the companion early on and letting it observe the player, the need for any long term teaching can be removed from the rest of the game.

Although the opinion system will not be relatively complex in the data it captures, it will be shown that, used properly, we will see several different complementary behaviours in the companion for each different player. This small addition to the game will be able to take advantage of an agent's normal, pre defined behaviour and with the combination of the opinion system, produce smarter and more believable AI.

1.3 Opinion System in Video Games

This dissertation hopes to show that the opinion system will be an acceptable approach to importing video game AI across multiple genres of games. To show its broad application in the world of video game AI we will focus of a single genre in the current video game market and apply an opinion system on top of similar AI used in that genre. In Section 3.1 we will discuss further on the types of genre the opinion system is suited for and a host of example applications.

1.4 Dissertation Layout

Chapter 2 reviews the state of the art in multi-agent learning. Chapter 3 focuses on the design of this project and what needed to be created. In Chapter 4 I talk about the implementation of the project. Chapter 5 gives an evaluation of the success of this project. Finally I will give my conclusions in Chapter 6 as well as any future work that could be done in this project.

Chapter 2

Literature Review

2.1 Introduction

The purpose of this chapter is to complete a review of the state of the art in learning techniques for creating more believable artificial intelligence, including cooperative agent learning. Some techniques are used in the domain of video games but it is believed that all techniques could be applied to learning in video games. This chapter is broken up into four sections. Each section is introduced with a quick review of a different learning technique and is followed by a review of a number of papers that have been written using the same learning technique.

2.2 Neural Networks & Genetic Algorithms

A method that is often used to create more believable artificial intelligence is neural networks (NN). A NN is a basic artificial representation of how a human brain functions. A basic description of the brain is a highly interconnected arrangement of billions of neurons. Each neuron can communicate with other connected neurons through the passing of electronic signals. The function of each neuron is to evaluate the input of multiple signals from connected neurons and to fire off a single signal if the sum of the received signals is above a certain threshold [1]. Neurons are very efficient at generalizing from given examples and as such, when confronted with a new scenario they can produce good responses [18].

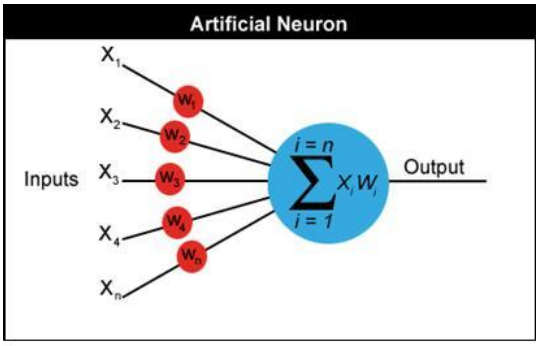

Figure 2.1: Neural Net [1]

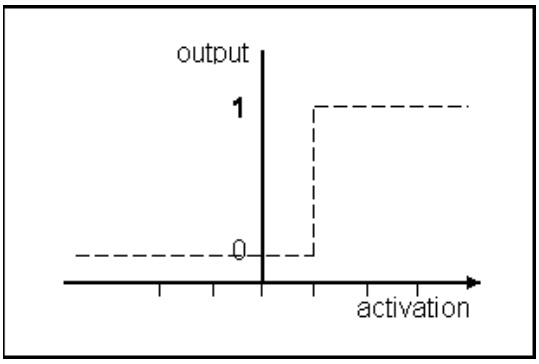

Figure 2.2: Neural Net Step Function [1]

The number of NN needed in any given project is determined by complexity. It can be anywhere from a few to a few million for the more ambitious projects. The most common NN used is called a feedforward network. The construction of multiple neurons is used to make a feedforward network. The main function of a neuron is called a step function. It takes each input and sums up their associated weights and returns a signal if the sum is above a threshold, or a 0 otherwise. Figure 2.1 shows the construction of a feedforward network. A number of neurons are positioned in a hidden layer and each input is feed into each neuron. The number of hidden layers is something that needs to be figured out during testing.

Genetic Algorithms(GA) are the most commonly used training algorithms used for reinforcement learning (2.3) for their ease of use and efficiency in complex situations [18]. GA are used with NN to train and evolve their data to produce optimal solutions.

They are designed to mimic natural selection where a combination of two strong species can combine to create a stronger species with traits from both of the original species [19]. Every NN requires a training algorithm.

A GA collects the weights that are assigned between NN and forms a map of them into a chromosome. Usually the values are represented as binary, 1s and 0s. The number of chromosomes represents the number of NN. Each chromosome represents a different way of solving the given task. In the case of foraging agents, where agents move around an environment and try to find certain items like rocks, one chromosome could represent an agent who moves fast while another represents an agent who can see very far. A fitness score is awarded to each chromosome. The score reflects the ability of a chromosome to complete the given task. Two of the chromosomes are selected for mutation. The selection process can be based upon the strongest members [2] or using a roulette method [18]. The roulette methods randomly chooses which member to select for the mutation with members with higher fitness having a higher chance of being selected. A crossover rate is used to determine which portion of each chromosome from each member will be used in the combined chromosome. Finally a mutation is applied to the combined chromosome which may flip random bits in the chromosome from 1 to 0 or viceversa [19]. One cycle of this selection, breeding, mutation process is known as a generation [20]. The process repeats over many generations until an optimal solution is found or the allowed number of generations is completed [21].

2.2.1 Machine learning techniques for FPS in Q3

Neural networks and genetic algorithms have been used to try and improve the AI in certain video games. In [18] the author looks at using NN and player observation in order to produce more human like behaviour and strategies in the popular first person shooter (FPS) game Quake 3 [22]. It is noted in this paper that AI, especially in FPS games has reached its limit and fails to produce adaptive behaviours and complex strategies, and that developers tend to resort to creating AI that is superior only because its accuracy and health is better than the player's, not its behaviour. Although it has also been shown that many players perceive a stronger enemy as a smarter enemy [23] it has also been shown that players consider this type of agent behaviour to be cheating.

The learning data recorded for these agents was taken directly from analysing the complex behaviour of human players. The behaviour is broken down in sub-behaviours with each sub-behaviour being assigned it's own NN. The main areas of agent improvement that were observed while playing Quake 3 [22] were:

1. A player's movement during combat
2. A player's movement outside of combat
3. A player's actions with weapons

During 1, observations are made of how the player moves and the NN is assigned the task of learning which movement is best to perform during a combat situation. During 2, observations are made of how the player moves when escaping from enemies, movement when approaching enemies and movement when gathering items on the map. As item spawn points are static Quake 3 [22], agents are able to create check points for each item on the map. This allows for each sub-behaviour to be treated as the movement to a checkpoint. Best paths and map layout are also learnt from these observations. During 3, observations are made of which general weapon information of the player. Observations include, which gun is being held by the player, how much ammo does the player have left, and general information about the players shooting skills etc.

The final results were interesting. After 2000 generations the author started to see human like behaviours in each of the 3 categories. Although lot of redundant data was recorded where the observed player behaviours were too complex to understand and as such the final agent was not at a high enough level, competitively, to use as an actual Quake 3 bot, the results were promising for future projects.

The paper does mention that NN do produce the best results when compared to using decision trees or Nave Bayes. It also mentions that typically player actions that are recorded are quite small and are taken from short training scenarios. The reason is that the computational overhead for a full game is too much. Therefore to record more data the author has done all computation off-line after the data has been recorded. The problem with this is that the evolution is not being done in real-time and there needs to be over 2000 generations of training the NN with the genetic algorithm before human like behaviours emerged. This would not be fast enough for most games but

could be used to help train agents during development to create agents for the final game. This method doesn't seem to be level specific as it works on top of a regular Quake 3 agent and as such it should have the necessary information of how to traverse all levels.

2.2.2 Real-Time Evolution Of Neural Networks In The NERO Video Game

Stanley improves on the computation problem of the previous paper by evolving NN in real time. It explores the improvement of agent adaption to its environment [2]. Their final application allowed users with no prior knowledge of machine learning or NN the ability to train agents. This is an important application especially in video games. It allows players to train their own agent without having to have any knowledge of the learning system running in the background.

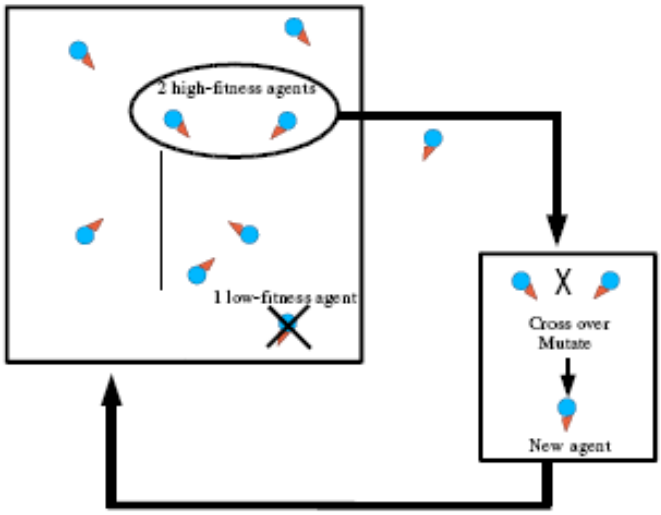

Figure 2.3: Replacement cycle in rtNEAT [2]

The agents in this paper are evolved in a specifically created game called NeuroEvolving Robotic Operatives (NERO). The game was built using the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) application, which evolves NN in real time. The game houses multiple learning agents each controlled by a NN that makes

decisions based on sensory information. The NN are relatively simple to begin with and evolve to become more complex over time.

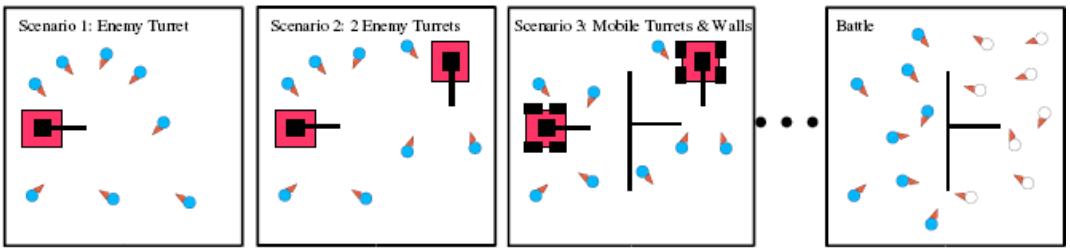

Figure 2.4: NERO Training Scenarios [2]

To improve the speeds of evolution in the game, agents are separated into different species and compete with their own groups of species primarily. Agents interact with their own groups first. This gives them more time to optimise their own structure before they need to compete with agents from other groups. information is stored within their genes to show which species each agent originates. In competing, two agents with the highest fitness level mutate to produce a new fitter agent (the assumption being that generally that it will be fitter but due to randomness this may not always be the case). An additional optimization is that each NN does not pass through any hidden nodes and only the fittest nodes survive. This reduces the number of nodes that need to be checked. While this approach is not optimal for searching for solutions in high-dimensional space, it works well for lower dimensional spaces where most solutions are found.

Although NN are very effective in producing excellent results, their computational overhead and time needed to create complex NN is too slow for the modern day action game. They also offer no guarantee of consistency in the learning that is being done. There needs to be a certain amount of random exploration before learning is achieved [24]. They do offer the ability to create competitive characters in games to produce unpredictable AI that offers more of a challenge [15].

2.3 Reinforcement Learning in Cooperative Multi-Agent Learning

Reinforcement learning (RL) is a reward-based learning method that is used to teach agents the optimal action to choose in a dynamic environment [25]. Agents are not given any commands of which actions to choose but learn from exploring their environment and trying out various actions. This leads to rewards being given to the agent. These rewards in turn lead to the agent favouring certain actions and weakens the favouring of other poor actions. There are two main characteristics that govern RL: trial and error search and delayed reward. Actions affect immediate rewards and as such the possibility of all subsequent rewards [26]. Repeated execution of this process leads to an agent learning optimum actions strategy, i.e. the strategy that leads to the best rewards, [3]. To prevent an agent becoming complacent with the actions it likes to perform, there is an added function that forces the agent to explore its set of actions it has not used previously. The function is called the ε-greedy approach. A greedy approach is one that will always choose its perceived to be the best. If the less preferred action results in a higher reward, the ε-greedy approach gives a random chance of an agent using a less preferred action in a given situation with the possible result being the action produces a better reward. The probability of this action being used in future increases [26]. One of the major outcomes of the use of RL is the adaption of agents to uncertain environments, e.g. an agent coming in contact with a new enemy of which it has no knowledge of the enemy's weaknesses.

Another reinforcement learning algorithm is Q-Learning. It can be used to learn optimal actions given past knowledge of actions without requiring a model of the environment [26]. Instead, an agent quantifies the expected outcome from performing a certain action in a given state. A state-action Q value for each action that can be performed given a state is added to a look-up table. After an action is performed, a reward is given, the new state is observed and the Q value is updated in the look-up table [27].

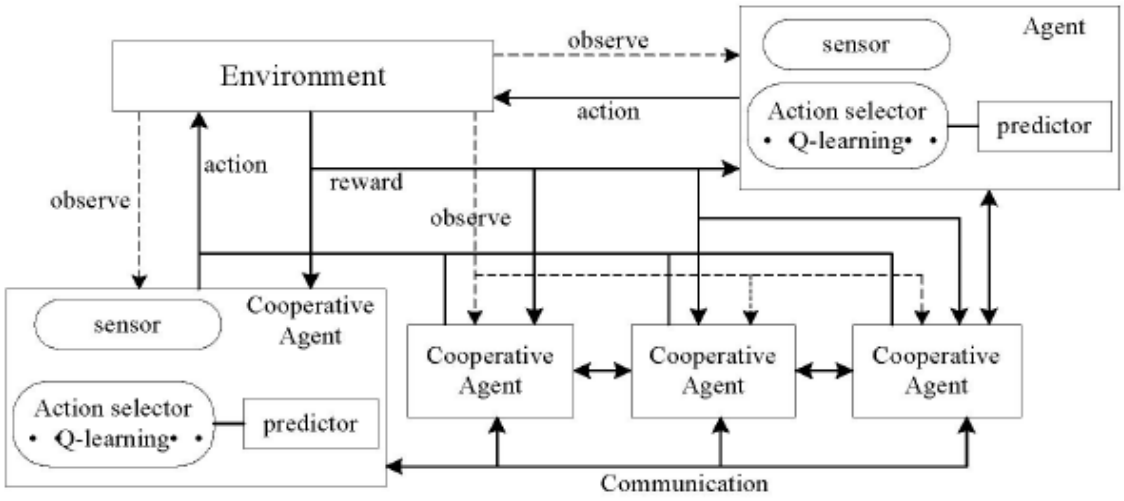

Figure 2.5: Multi-agent Cooperative Learning Model [3]

2.3.1 Multi-agent Cooperative Learning Research Based on Reinforcement Learning

Liu [3]presents a Multi-agent Cooperative Learning Model and a Multi-agent Cooperative Learning Algorithm to improve both coordination of actions during learning and computational overheads from RL which affect convergence speeds (the speed at which the actions of all agents actions are similar in each scenario). Their Multi-agent Cooperative Learning Model, Figure 2.5, is an advancement of the Bowling model [28]. The model uses a predictor to calculate other agents future actions through observations. The RL model uses Q-Learning combined with an action selector module to select optimal actions based on Q values. The execution module performs selected agent actions which in turn change the agent's state and their environment, e.g. an agent performs a pick up action resulting in the agent changing its current state to be, "picking up", and result in the picked up object being moved in the environment. The Multi-agent Cooperative Learning Algorithm combines the possible action an agent may take with the prediction of an action that another agent may take (this prediction is based on the history of the other agents' actions), to produce a long term reward. The best reward leads to a certain action being performed in the future;

Both model and algorithm were tested in a hunter-prey, team cooperative simula-

tion. Three cooperative agents move around a two-dimensional grid trying to surround another agent and force it to move into a certain node to be captured. Four tests were run with different memory lengths and sampling values. The memory size describes the size of previous action knowledge. No matter the choice of what values to use for the memory lengths and sampling rates, the tests always moved towards convergence. It was shown that with the right memory value, the speed of convergence increased and the right sampling value caused the algorithm to be more stable. Agents form an accurate prediction of the actions other agents will perform in the future over a gradual period. Overall, results are positive with an increase in speed of convergence and a much smaller search space.

2.3.2 A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination

[29] solves the problem of slow speed of a very large increase in state-action space that occurs from multi-agent learning with a distributed Q-Learning algorithm. Their solution is to allow agents to share strategies with other agents. E.g. an agent approaching an uncertain environment can use the strategy of an agent who has already encountered this environment. They improve the ε-greedy strategy of Q-Learning and add a sharing-strategy to reduce the amount of searching needed to be performed in the state-action space. The algorithm also allows the sub-division of goals. The experiment used is the same hunter-prey model looked at in the last paper [3]. The goal is a little different though as the hunters task is to surround the prey on all four sides. The goal is that among the hunter agents, each side is sub-divided to a specific agent.

The ε-greedy strategy is in this algorithm to select the best action given a certain situation. As explained before this may not lead to the satisfaction of an expected reward. Therefore the ε-greedy strategy randomly explores different actions in certain situations. As the game progresses the knowledge learnt by the agent increases which further increases the state-action space the agent has to search. The improved ε-greedy strategy in this paper aims to reduce the search space over time by reducing the ε value. When searching for an action, two are chose. A random action and an action that will give the optimal reward. Based on a random ε value, one of the two actions is selected. The ε value is gradually reduced over time and reduces the amount of exploration with

respect to the learning process.

An observation given in the paper is the problem that Q-Learning requires a trial and error approach to learning. In their experiment, the hunter may not be able to see the prey given both of their current positions. The hunter must then make a random movement in the hopes that in its next position it can see the prey. To improve on this problem, hunter agents can get the position of the prey from any other hunter agent who can currently see the prey.

The method presented in this paper shows significant improvements over regular Q-Learning for multi-agent cooperative learning. Although this method does reduce the action-state space that needs to be searched, it still enforces a low number of actions each agent can perform to keep the space as low as possible. In a regular video game the number of agent actions would be very large and we would not see a method like this being used. Also the paper mentions that the introduction of errors into the system is possible and thus requires a large number of runs, 500 in this experiment, to reduce this error. It is unlikely a situation that could be learned from would be encountered this much in most video games and would seem to be a long time to wait to produce optimal actions.

2.4 Team Mate Modelling & Cooperative Multi Agent Learning

Multi-agent environments are environments where there are more than one agent. Co-operative agents are agents who work together to solve a given task especially when faced with uncertainties and incomplete information [30]. The advantage of multi-agent cooperation is that eventually the two agents will start to parallelize their efforts. This leads to agents that can suggest which actions to perform to each other or agents that prioritize their actions based on the actions of other agents [20]. Compared to cooperative agents, agents act independently to meet their own criteria in unknown scenarios, usually results in undesirable situations [5].

One of the simpler methods of multi-agent learning is Team Learning. In team learning, only one agent is actually learning. The behaviours learnt by this agent are then used by all the other agents in the environment. The main advantage of this type

of learning is that it uses single agent learning techniques to avoid and the co-adaption of multiple agents who all learning together. The two main types of team learning are homogeneous and heterogenous team learning. The learning agent in homogenous team learning supplies a single behaviour for all agents in the environment. The learning agent in heterogenous team learning can supply a different behaviour for each agent in the environment. This agent is trying to improve the overall performance of the whole team. Homogeneous team learning is best suited in situations where the behaviour of each agent is not required to be optimal. One situation is in a foraging environment where each agent does not need to have optimal foraging skills. This reduces the search space of possible actions. Heterogeneous team learning is best suited to environments where certain agents need to specialize in certain tasks, e.g. one agent specializes in scanning object while another specializes in lifting objects. The search space is drastically increased in these cases, a task not suitable for homogeneous team learning. A task suitable for this type of learning could be football where specific agents require different skills [20].

2.4.1 Rational Coordination in Multi-Agent Environments

Gmytrasiewicz [4] coordinates the actions selected between multiple agents based on both their environment and also the actions of other agents. They use a recursive nesting of models with dynamic programming to create rational actions for agents to perform. The design of the project improves coordinated actions in unpredictable scenarios without affecting agents' already pre-defined policies. The proposition is that in such cases where agents face unpredictability, they should just use all their own knowledge of the environment, and their knowledge of other agents, in order to take the best action. The coordination that occurs between agents helps each agent hypothesize about what actions other agents may take in the future. Doing so allows an agent to anticipate changes in its environment and helps it choose an action that will put it in a best possible new state. This decision model is said to create "precise" coordination between agents. This project requires a model to handle recursive thinking in order to make rational action decisions. The recursive nature of the method used in this paper arises when an agent believes that another agent could be considering to choose an action based on what actions other agents might make. An example of where situation

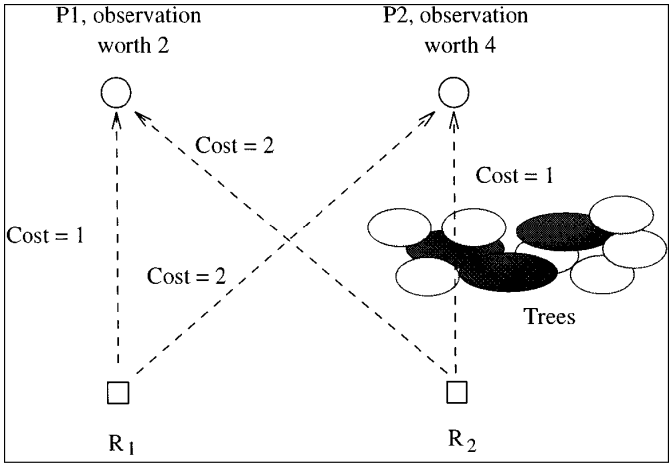

Figure 2.6: Agent Learning Environment [4]

occurs is given:

In the situation shown in Figure 2.6, two agents, R1 and R2 can perceive their environment and are tasked with gathering all information about it. Each agent is only allowed make one move but the gathered information will benefit both of them. From R1s position he can see and move to position P1 and P2. The cost of moving to P2 is greater than that of P1 but the information gathered would be double. Given the choice of not moving or moving to P1 or P2 the agent creates a payoff matrix (a matrix representation of the current situation). To make a decision of which action to take, R1 needs to predict the actions of R2 with leads to three different predictions. The first is that R2 will move to P2 to make the best observation and therefore R1 should move to P1 and maximise the combination. R2 may not be aware of P2 as a position to move to and therefore R2 will only consider staying put or moving to P1. The last prediction is that R2 may behave irrationally. The probability that the agent can see through the trees to P2 is given a score of 0.1, and 0.8 for the probability of not being able to see through the trees and a 0.1 score for action irrationally. The best decision for R1 to make is to predict how it thinks R2 would predict R1s actions, assuming R2 is thinking rationally. This causes a recursive nesting of information. To terminate the nesting in what is called a *sub-intentional model*, R2 would need to have some subjective probabilities of R1's previous actions based on past encounters of R1's actions. This is the belief that each of the models is accurate. If no such information

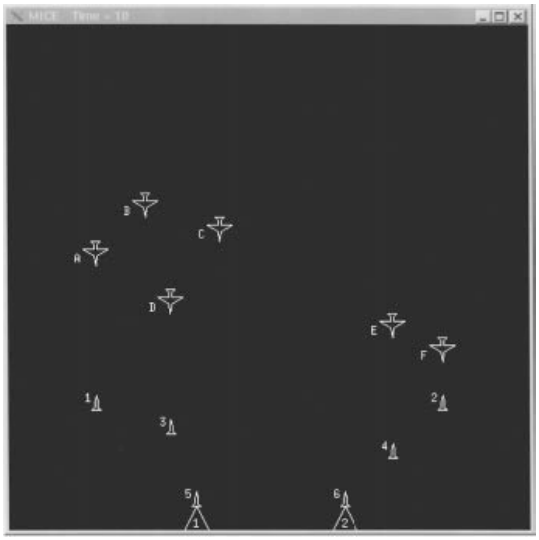

Figure 2.7: RMM Learning Demo [4]

was available the nesting terminates and produces no new information. The recursive model created to solve the problem is called a Recursive Modelling Method (RMM). The model uses dynamic programming to unravel nested agent information to make rational decisions. An in depth, mathematical review of this model and how it can be used to solve the above problem is given in the paper.

The main demo used to test their model is a simple air defense game where two agents cooperate to protect their base from incoming missiles. There are a total of six missiles, each with a different weight and each agent can only fire 3 intercepting shots. Interception of each missile is dependent on the threat level (how big each missile is and how far away each is) as well as the perception each agent has of the other. Each agent was able to observe which missile the other agent fired at but there was no communication between them.

The experiment consisted of different agents performing the task. The first experiment used the RMM model described in the paper. The second experiment used a worst case scenario including random decision agents. The third used independent agents who tried to maximize their own payoff matric but did not try to predict the action of the other agents. The last test used a human player with knowledge of how to play the game. One test had two human players and a second had one human and

one RMM player. The RMM model achieved the best scores but suffered from the unpredictability of other agents as well as not having enough information about other agents. The human and independent agents performed similarly as both used intuitive reasoning rather than any deep consideration of the other agent/player. As predicted the random agent performed poorly.

The recursive nature of the method used in this paper arises when an agent believes that another agent could be considering to choose an action based on other what actions other agents might make. It would be a good addition to creating more rational companion agents but not human like agents. Anticipating how the environment would change given a certain action would be very desirable to implement. Before an agent runs into a situation it could calculate the chances of survival given that a player is likely to support or not support them.

2.4.2 Multi-Agent Learning Model With Bargaining

[5] uses cooperative game theory in the form of an algorithm based on the Nash bargaining solution. The bargaining solution is often used in the prisoners' dilemma puzzle where the cooperation of two people playing the best strategy leads the best solution for both of them (see [31]). The experiment uses two grid based games, Figure 2.8 where the objective is to reach a designated end node on the grid. Agents perform actions simultaneously and do not know the location of the end node or the reward for reaching it. Each agent has its own objective (reach the end node in the fewest steps) with a payoff that it wants to maximize. The payoff of the objective is determined on the joint actions of each agent. To improve coordination between the two agents, each agent is able to perform actions to improve their own objective without preventing the other agent from improving their objectives. An agent is rewarded with 100 points if it reaches the end node, 0 points if neither agent reaches the end node and -1 points if both agents move into the same node resulting in them moving back into their previous node. The first grid permits the agent to move within the confines of the grid but the second grid places a wall above the agents forcing them to share the first node they can move into.

The algorithm for each agent chooses an action and then observes the reward based on the outcome of the action and the new state. The reward is then based on the joint

Figure 2.8: Grid Game 1 & Grid Game 2 [5]

strategy policy taken by each agent. The agent then selects a new random state to explore the grid. The algorithm results in mutual cooperation between agents.

2.4.3 Multi-Agent Reinforcement Learning Algorithm To Handle Beliefs Of Other Agents' Policies And Embedded Beliefs

Makino [32] picks the best policies for an agent based on the agent's policies of other agents. Each agent does not have knowledge of the other agent's policies but can observe the scene fully and can therefore make its own beliefs of the other agent's policy. The paper talks through the development of their algorithm while creating more intelligent agents. The experiment uses the prisoner's dilemma cooperation puzzle [31].

Level-0 Agent: The most simple agent ignores all other agents in its environment. Its level of learning is quite low. Level-1 Agent: To improve on the level-0 agent, the level-1 agent needs to make an estimate of the other agent's policy using some form of statistical estimation. The number of observations an agent needs to make of other agent's actions is quite large, though, meaning that the process is quite slow. By obtaining accurate beliefs about another agent the agent will perform optimal actions. Level-2 Agent: The next agent has a belief about other agents metapolicies. This is a map which is formed about the policies beliefs of policies. When used by an agent it can estimate the policy of another agent (assuming the first agent has an embedded policy within the second agent). A more accurate belief of another agent's policy is obtained. Level-n Agent: The agents continue like this, creating more embedded beliefs of other agents policies using metapolicies to obtain better beliefs.

The experiment used is the same prisoners' dilemma ([31]). Each of the two agents can cooperate or defect at each turn in order to receive the lowest prison sentence. The results of each agent is observed by the other agents and a pay-off is given. Each agent's desire is to maximise their own pay-off. The final results showed the higher level agents, Level-n agents, converged to maximum pay-offs after around 2000 iterations but agents fail to cooperate 30% of the time.

Creating a belief about other agents seems like a strong addition to an agent's decision algorithm. In the application of video games, an agent who bases its decisions on those of the player is going to be more complementary if they are support AI and more cunning if they are enemy AI.

2.4.4 Learning conventions in multi-agent stochastic domain using likelihood estimates

Boutilier [33] compares a Bayesian best response method and likelihood estimates to try and converge coordinated actions between cooperative agents who are trying to achieve the same goal. The cooperative agents used in this experiment cannot observe the actions of other agents. Agents can only observe the states performed by another agent and will never have access to the "intentions" that the agent had in choosing their action. In the game, agents make observations of other agent's various states which allows them to create a probability of the agent's intentions of their choice of action.

The game used is a very simple stochastic model. Agents can change state to either being on the left (l) of a board or on the right (r) of a board with the intention of moving to a certain goal. The optimal joint action of the agents it to end up in the same position, (l ,l) or (r, r). The paper first gives an example of how agents are modelled in the game where actions are observable, using both a Fictitious Play learning model then a Bayesian Best-Response Model. In the fictitious play learning model, each agent keeps a count of how often the other agents chose a certain action before. This allows the agent to calculate the probability of the other agent choosing a certain action given a certain situation, i.e. the probability of moving left or right. There is no guarantee that using this model will see both agents converging to the same strategy although given the limited number of actions, it is given that once agents converge to the same

strategy they will remain using it. The Bayesian best-response model assumes that all agents hold previous knowledge of other agents. The agent holds a degree of belief that the other agent will choose an action/policy given a certain state. The starting belief of each agent is (1, 1), i.e. each agent holds the belief that the other agent will move left or right with equal weight. Both agents choosing to move left will update this belief to (2, 1), a deterministic evaluation that a chosen action will give a chosen result. In a repeated situation we would see both agents converging to a similar optimal joint action and moving left again. There is no guarantee of how quickly this will occur if agents randomly choose opposite directions to move. The beliefs of each agent will become (1, 2) and (2, 1) respectively. It is shown that the larger the number of agents in this type of game, given the same actions will see a very rapid convergence. Adversely, trying to get agents to converge to opposing actions (one left and one right) sees a very slow rate of convergence. The second example models agents that do not have the ability to observe other agent's actions, but only the states that occur after each action. Further, all agents have knowledge of the game structure and can therefore find evidence as to the choices of other agent's actions.

Unlike the previous Bayesian model, an agent can not use updated beliefs of other agents to predict future actions. By applying Bayes' theorem [34] to the previous model a belief of another agents strategy can be created. Due to the stochastic nature of this model the probability of an agent choosing to move left and moving left is 0.9 and a 0.1 chance of moving right. Two occurrences of uncoordinated decisions, e.g. (l, r) + (r, l), will result in a agents having a belief of (1.938, 2.061). This prevents any future chances of convergence unless the agents chose the same optimal joint action to begin with. The unlikely chance of an action failing is enough to break out of this sub-optimal joint action and it has been shown interestingly that actions that are more predictable and less error prone are slower to converge. To improve convergence, random selection of best responses is added. The theories in this paper are quite promising but it does not give too much detail in how it works given large numbers of actions each agent can take. Also, each model does have the chance that agents who are converging might break out of this cycle due to unexpected scenarios. The evidence that was gathered for my project was to allow all action to be observed to ensure rapidly increase convergence of actions between player and companion.

2.4.5 Learning models of other agents using influence diagrams

In [6], Suryadi created a learning model for agents to learn the models of other agents based on observations of the other agent capabilities, preferences and beliefs. A number of models of each agent are created and given probability ratings based on the likelihood of them being accurate. If none of the models are accurate, a modification of one of the other models is made. An Influence Diagram displays an agent's knowledge of a decision problem and can be evaluated to learn an agent's beliefs. It consists of three types of nodes: a nature, a decision and a utility node. The nodes represent an agent's uncertain beliefs about its environment, all of the actions an agent can take and an agent's preferences respectively.

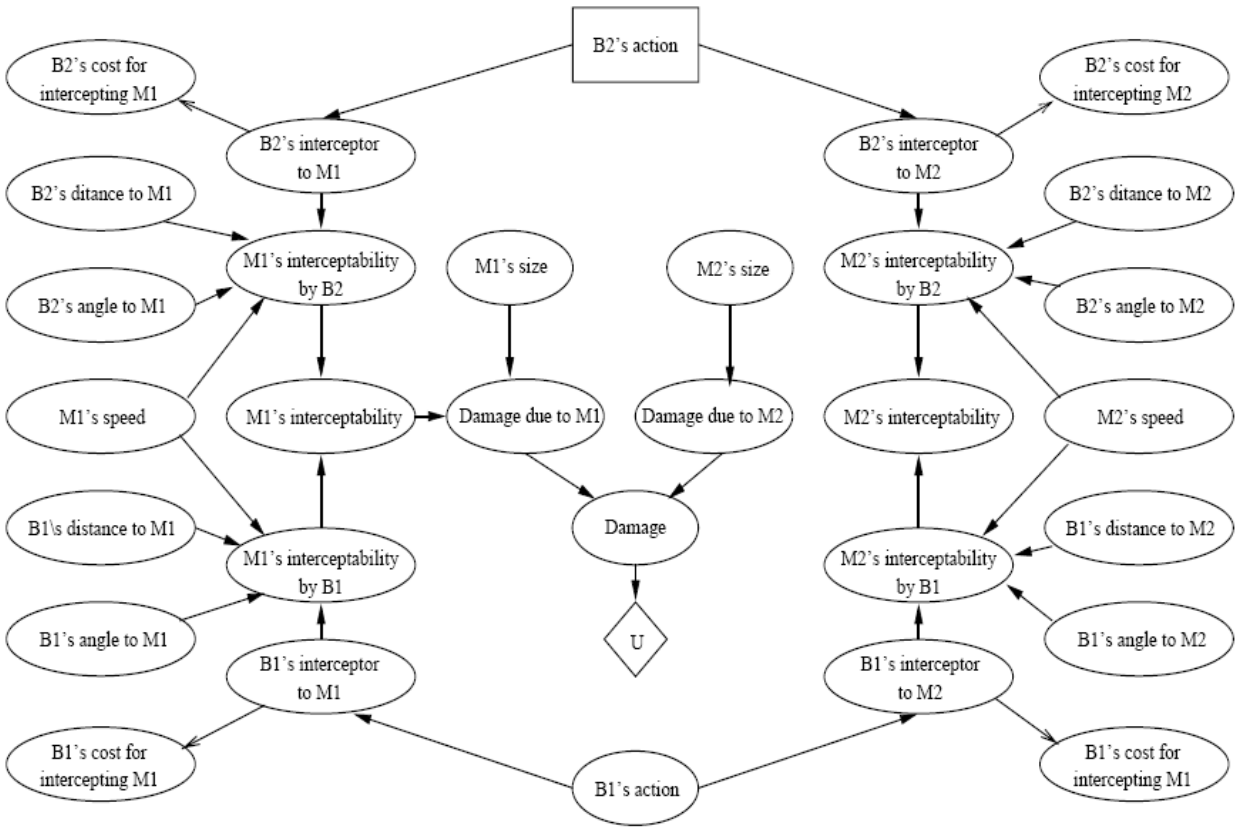

Figure 2.9: Decision Model [6]

The capabilities of an agent are the set of actions it can perform and are represented in the influence diagram on the decision nodes. All capabilities need to be accounted for no matter how unlikely they are to be performed (Figure 2.9). An unlikely action is given an "Other" value. An observation of such an action requires an update of the decision model to incorporate the action as being possible. This may occur in a game where an agent models a second agent as not possessing the ability to heal itself and later the when the second agent learns this healing ability, the first agent will need to update its model to represent the second agents new action. Conversely if agent that loses the ability to perform an action, an agent modelling it will need to collapse a portion of the decision model and give it an "Other" value.

Preferences help model previous agent behaviour. An inaccurate representation of preferences in a model can lead to the inability to explain an agent's previous behaviour. An adjustment to an agent's utility function (given a state, the features that influence the preference) is required to create an accurate preference. Each of the features that influence the utility function is weighted depending on the value of the feature in a given state, given a non-deterministic action. Neural networks are used to adjust weights when an agent has no knowledge of another agent's preferences. The training set used for the NN is a history of an agent's behaviours.

Beliefs are the probabilistic relationship between nature nodes in the influence diagram. If behaviour cannot be depicted from the preference and capabilities knowledge that an agent possesses, the model needs to be refined. The strategies given all increase the number of models to a large number and there was no method proposed to reduce this number. The ideas in this paper are very strong but results for how well the resulting model performed are inconclusive.

2.5 Opinion Systems

The main area of research in this project takes the idea of using an opinion system as a way of improving the AI of the companion. Adam Russell talks about how he used an opinion system [35] to increase the dimensions of a non playable characters AI in Fable [36] and the lessons learned during the development. The article uses the ever increasing interest in customizable characters in action-Role Playing Games (RPGs) as the basis for the need to create deeper, interactive non playable characters (NPCs). I

see this as very relevant to this project as it is in effect allowing the player to customize their own character with its own beliefs about how a game should be played.

The design of Fable is the ability for the player to not just increase their overall wealth and fighting experience in the game, similar to regular RPGs, but to grow as a character in terms of their appearance, personality and overall renown. To reflect all these changes within the game the developers wanted to create a much deeper conversation system where NPCs talking would respond topically to the overall identity of the player. Each NPC could therefore form their own opinion of the player. Typically NPCs in games that are used as support or enemies have good AI, whereas NPCs like villagers in games require very low AI, not much more than wandering. Also support or enemy AI tends to be centered around their environment as a basis for making decisions of which actions to choose. The interesting part of the villager AI in Fable [36] is that they use the player as a constant basis for evaluating their choice of action.

Opinions are easy to summarize. A person has an opinion about most things. At a very basic level a person either likes something or dislikes something. Obviously this is a very simple representation as the scale of human emotions is broader but it holds well for this example. An opinion state within Fable is defined as the evaluation an NPC has about the player at any given point. There is no historical knowledge of the players previous moves kept. It seems to be that who the player was before doesn't need to matter within the game. Again at a very basic level an NPC can like or dislike a player.

Each player within Fable can be quite complex in its attributes. Players can be good or evil, look handsome or look silly, a loving husband or a cheating husband, etc. To accurately represent each NPC's opinion of the player they wanted them to react more complexly than having a singular like/dislike opinion, the designers of Fable wanted each NPC to have a more complex, multi-dimensional opinion.

Some of the advantages given in the paper of choosing multi-dimensional opinions over single-track opinions are:

Orthogonality

A single-track opinion is like our basic example of an opinion. It doesn't offer more in the way of information than a single preference. On the other hand, have multiple opinions increases the variety of information that can separated about a person, like-

ability, respect, trust etc.

Greater Variety of Effects
Again a single opinion will not give more than a simple description of a player for each NPC, e.g. a nice person. The addition of multiple opinions gives a much more complex description, e.g. a nice person but not too brave.

More information
Instead of having to keep information about all the player's previous actions, the opinions act as a representation of previous actions. If an opinion for likeability is high, it can represent that the player must have previously done several nice deeds and therefore is a good person. This eliminates the need to store information about every good deed the player completed for each NPC.

The paper also discusses about how multi-dimensional opinions bring about complexity:
More Brittle
Changes to the categorization of opinions were liable to affect other areas of code. When an opinion is updated it needs to be check that it does not conflict with other opinions

More Confusing
When a player performs a task it needs to be figured out which opinions need to be adjusted. A single opinion would only need to be incremented or decremented but using multiple opinions requires the developers to decide how many of the opinions need to be adjusted.

More Difficult to Quantize
Every action made by NPCs is a combination of all the opinions in order to capture the volume of their overall opinion. Quantizing multiple opinions is much harder than a single opinion.

The advantages of opinions are that they are instantaneous. Unlike reinforcement learning, the player does not have to wait for the agent to come to the decision which is the best action to take. It may take a learning agent 2000 iterations to learn that a player like to perform action X in situation Y, but using an opinion the agent can learn this from one observation. Opinions can also be mixed with predefined agent behaviour to create a tailored AI for each player.

Chapter 3

Design

3.1 Introduction

The opinion system has potential for adaptive companion AI in a lot of different genres of games. This introduction to this chapter looks at several video game genres and the potential for the opinion system to be integrated for the use of improving companion AI. We will then discuss the genre that was eventually picked for this project and the reasons for doing so. We will then discuss a proposed architecture for the companion agent and how the opinion system could be applied to its AI.

Real Time Strategy Games: For Real Time Strategy (RTS) games, a combat feature that is typically available to the player is to send an army into battle and issue commands for that army to make. At this time an agent, that represents a General in player's army, could be observing the players successful attacks, ignoring unsuccessful attacks, and learn the opinions from these attack. Each opinion could correspond to the player's preference in what combat forces they use, e.g. the player might like using tanks with the addition of ground troops and the generals can observe this. As the game progresses and the player must fight lots of separate battles, they can send out their Generals, coupled with the Generals' opinion of the player to take command of the battles.

Fighting Games: In fighting games, like Tekken [37] and Soul Calibur [38], the

enemy AI could use the opinion system to improve their AI. Like most games, more challenging, unpredictable enemies are a requirement as facing the same challenge repeatedly can get old quick. The game's challenge can be kept fresh by varying the enemies AI frequently. Through observations of the player's fighting style, an enemy could make observations that a player likes to use high punching attacks and blocks infrequently. The enemy could form this opinion of the player in order to decide which actions to take. In this example, the enemy could use a leg sweep attack to focus on the week area of the player's body. The player is out-smarted and learns to change their fighting style

Role Playing Games: RPG's often overload the player with large amounts of character and companion customization. Players are a large number of options, be it the ability to improve physical attributes like speed and strength, the number of weapons that can be used or even the number of magic spells that they can cast. We could see, as the player starts choosing weapons and spells for themselves, that the companion could start to customize themselves in order to complete the player's choices. Focusing on the player creates an opinion for each of the player's strengths. This could lead to the companion selecting strategic choices in their own customization, e.g. the player does not like casting fire spells so I should in case we meet an enemy who is weak against fire attacks.

For this dissertation, the genre focused on is the First Person/Third Person Shooter genre, e.g. Halo [39] and Gears of War [10]. Both genres are very similar with the main game play style focused on action and shooting. These genres are highly suitable to cooperative gaming. One reason this genre is focused on is because of the low number of cooperative agents that the player usually has, typically one or two. A larger number of cooperative agents will need to form opinions of not just the player but possibly all other cooperative agents. Further more they made need to make opinions of the other agents opinions of the player causing far more complexity than would be feasible in this dissertation. These genres also score well when focusing on the objectives of this dissertation. The AI decision can be poor at times and usually do not focus on the player. Also they are a good genre that could integrate the confidence building techniques that should come from the opinion system. Lastly these genres will integrate

well with the teaching system and benefit from the removal of making the player give commands during the game.

3.2 Companion Architecture

The companion design is focused on the player first before anything else. This means that the companion will not choose an action to take based on its environment first, e.g. if an enemy is close, then attack. It will first look at the opinion it has formed of the player and decide, "How does the player want me to play and how does that affect the decision I make on what I see in my environment"? All companion actions are the same as the player. As presented in [23], support AI should be able to:

1. See things the player can see
2. Know things the player can know
3. Do things the player can do

Players want AI to behave as they would in the same situation[40]. We have taken this idea but instead of making the AI react exactly the same as the player would in the situation, it can react in such a way that is not mimicking the player but is choosing an action that is complementary the players chosen action. In this way the player can choose how the Agent should react but is also still left with a companion that behaves how they if they were in the agents situation. Most of the general AI architecture for the companion uses Matt Buckland's state driven architecture [41]. A more detailed description of State Driven Architecture and its implementation for this game is discussed in Section 4.7. A brief explanation of State Driven Architecture is to construct an artificial agent using finite state machines. Finite sate machines allow a developer to break down and artificial agent's behaviour into smaller pieces, or states. A quick example is that an agent's behaviour allows it to perform two actions, stand and move. Each action can be broken into a state that the agent can be in. An illustration can be seen in Figure 3.1. When the agent is in the standing state it can check it sees an enemy. Upon seeing an enemy the agent switches state to the move state and moves away from the enemy. After the agent has moved it can return to the stand state once it no longer sees any more enemies.

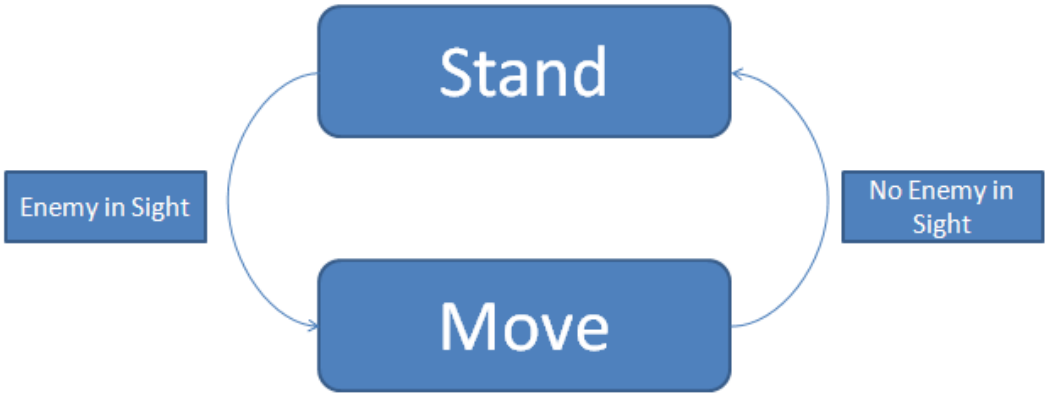

Figure 3.1: State Machine Example

When looking at what states the agent can be in there are only a few candidates: move, crouch, shoot and flee, i.e. combat behaviour. It was decided not to keep *crouch* as a state. The reason is that we wanted the companion to be able to move and be in cover at the same time. For this reason crouch was kept as an attribute of the agent that is moving in cover. The final state transition diagram for the companion is shown in Figure 3.2.

Unlike Gmytrasiewicz [4] or Suryadi [6], there are no constraints on the companion in terms of not knowing everything about their environment. It has full knowledge of how to move from any given point to another, assuming there is an unobstructed path. It knows the location of the player and enemies at all times. The reduces the number of things that needed to be coded for the agent to observe. While this system works for this project, in a larger game we would reduce the scope of what knowledge the companion is given to make it similar always to that of the player.

In Figure 3.3 I show the proposed companion architecture. Through teaching and player observations the companion forms an opinion of the player. This opinion is passed to the companion's state machine classes and is used in conjunction with normal AI decision making in order to make a decision on which action to choose. The chosen action is one that will best simulate the type of companion the player wants.

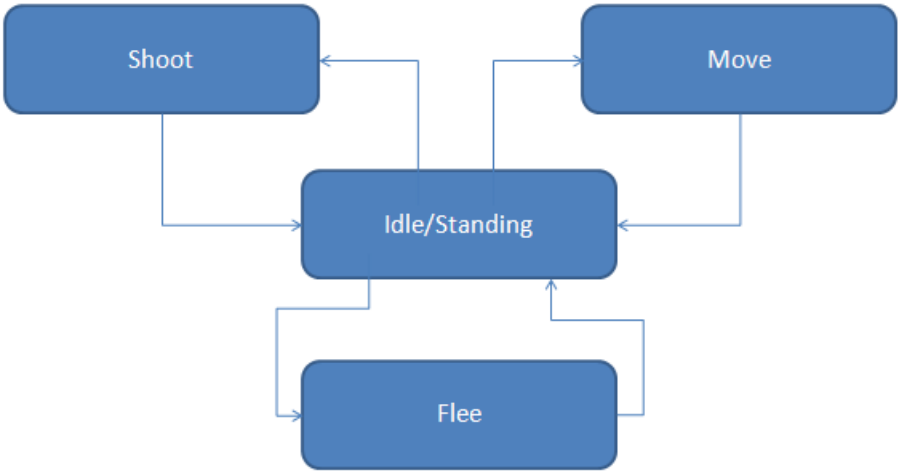

Figure 3.2: Companion State Transition Diagram

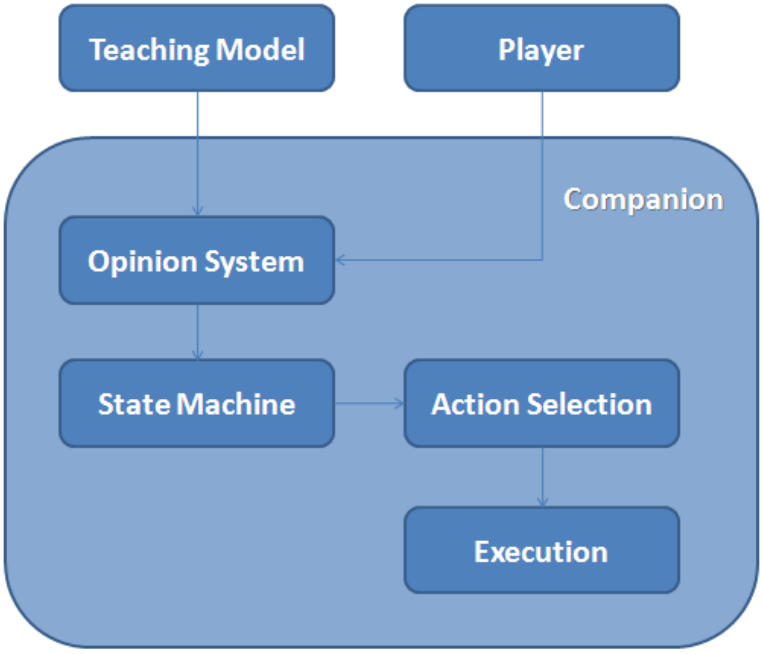

Figure 3.3: Companion Architecture

3.3 Choosing the Opinion System

This section will discuss the reasons for choosing an opinion system as opposed to the other methods reviewed in Chapter 2. Stanley [2] and Zanetti [18] both used neural networks (NN) with the intention for agent intelligence to become proficient over time. Though the use of NN could guarantee that the companion would be vastly more intelligent at the end of the game, the reality is that the NN are quite simple to begin with. They require a large amount of learning before we would see the emergence of human like behaviour. Further more the learning time for NN can be very long, with thousands of generation cycles needed and they require repeated repetition of the same tests.

The main reason for not using RL was the speed at which it takes the agent to learn. In RL there is an emphasis on trial and error techniques being used as a way to make the agent learn. In a action game, although the pace can be quite quick, the companion will not be completing enough actions with enough frequency for them to learn quickly. Also, although the technique helps agents perform well in uncertain environments, it is hoped that the opinions system would be used to teach which actions to perform in most of these uncertain situations through teaching. Also, unlike the ε-greedy strategy used in Q-Learning, [29], there is no random exploration of actions in the state space. Is is assumed that each enough situation types would be encountered in the training scenarios at the start of the game and as such the companion has knowledge of how of the proper action to perform in the right situation and does not need to try out other actions. Also any random selection of actions would lead to non-believable behaviour by the companion, something trying to be avoided in this dissertation.

When giving a talk in 2007, Peter Molyneux, the head of Lionhead studio's, who made Fable & Fable 2, used Asimov's three law's of robotics as the basis for creating better AI [42]

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey orders given to it by human beings, except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

For video game AI, Peter Molyneux translated them as:

1. Do not annoy the player - Don't get in their way or make irrational decisions
2. Focus on the player - What is it that the player would do
3. Look after yourself - Do not rely on the player to heal you

These are the rules that I feel the companion should follow I use these laws as a basis for how I want the creation of how to use the opinion system to create better companion AI. The opinion system seems to follow these guide lines when trying to improve agent AI. A new opinion can be created for each action the companion can make in their action set. By observing the player at all times, the companion will modify these opinions in order to make decisions the player would think are believable as it is the same decisions they would make in the same situation. Finally, the companion can use these opinions to form several roles for itself, complementary to the player, which helps the player and reduces frustration as the player will find the idea of the agent completing important roles to be beneficial.

3.4 Opinions

Selecting the opinions the companion would learn and observe, from the player, all focused on what are the attributes that should be given to the companion in order to first, make the actions rational in all situation and second, make them helpful to the player. This section gives an in depth view of each opinion and why each was selected. The design of each opinion is dual functioning. The scale of the opinions are between 0.0f at the lower scale and 2.0f at the higher scale. Initially all values for a default companion are set at 1.0f. This default companion has no preference in its decision making and as such is relatively useless until it forms its own opinion of the player. The effects of opinions above and below the default value produce varied outcomes and are discussed in more detail below.

Leader Often casual gamers will feel intimidated by certain games. Playing with a real human as the second player allows them to play the game without having to worry about taking the leader role. In this way they can enjoy the game and increase their overall confidence in playing the game[8]. But remove the second human controlled character and the player may avoid the game without receiving this helping hand. Allowing them to choose a companion that will take the lead is going to make them want to play the game more often as they feel less pressure to make all the decisions.

One of the main criteria when it came to designing the companion was to make sure it helped the player. For this reason I can use the *Leader* opinion as a way for players to create a companion that will take the lead during the game. If the player decides that the companion should take the leader role, the leader opinion is incremented by 0.1f. On the other hand the player may be quite a good gamer and only requires the companion to act as a support character. In this case the opinion is decremented by 0.1f and the companions role is now called a Follower.

As a follower the agent will not act on any decisions until the player has first made their decision. In this way, the companion will not run into a situation that the player would not themselves run into. This avoids players feeling bemused by the companion's choice of action. When the companion is in the leader role the player does not have to make the decisions on where to move to or which situations seems safe. They are entrusting the companion to guide them through the game and this increases the player's exposure to more of the game and thus increases their confidence in their own playing style. It is hoped that more players would be willing to pick up a game or return to a game if this feature was added.

Aggressiveness

Some single player games offer the player a chance to tackle a scenario in multiple ways, e.g. Dark Sector [43]. The player can sneak around a level and avoid all enemies or they can run into the level and shoot all enemies. Allowing the player to choose their approach to the game means that an agent won't run up to shoot an enemy and give away their position if the player is in cover and trying to use stealth to either sneak up on the enemy from behind or sneak past them altogether. Also, certain players do not like to use stealth too often, they prefer a more action orientated game where shooting is the main game play element and as such if the companion is the leader it will avoid

using stealth tactics in order to complement the player's preferences of playing style.

Tactics

A fun element of cooperative video games is the ability to perform cooperative tasks, e.g. the player and companion reach a high wall, the player tells the companion to give them a lift up so they can scale the wall, or vice versa. Generally the player presses a button to tell the agent to perform this shared strategy. The player is able to teach their companion which task to perform when given a choice. The cooperative task chosen to demonstrate this opinion was a simple flanking manoeuver. Here the player can request that the companion sticks close to him at all times. The other end of the opinion tells the agent will flank to the opposite side of a level from the player.

Accuracy

The *Accuracy* opinion is similar to the leader opinion. It is used to help improve a weak players confidence in a game. A player who is getting killed too often will not enjoy a game. The design of the companion should be that when a player has low accuracy and a low number of enemy kills the companion should help them out. Using the accuracy opinion, its assigned value can be increased each time a player misses their target and every time the companion has to kill an enemy. The higher the value of this opinion, the greater the chance that the companion will shoot at an enemy an also the greater the accuracy of each shot.

The other side to this opinion is that it can be used to not steal too much glory from the player. What this means is that the player is the hero in the game. We want players who are good at the game to be to kill as many enemies as possible. We therefore decrease the value associated with the accuracy opinion every time a player hits a target and every time a player kills an enemy. It is hoped as the player plays more their accuracy will improve and they can kill more enemies and usurp the hero role.

This opinion can become much more complex. If there are a large number of enemies on screen then the player cannot be asked to kill them all. The companion is there to help. We have to make sure that this opinion doesn't affect how the companion takes care of his share of enemies.

Self Preservation

An difference found in the *Self Preservation* opinion is that it is not taught by the player. It is an observation of when the player is healing by the companion. It was decided that this opinion could not be taught as asking a player, "At what percentage should the companion heal itself", is illogical. There is no one right answer. This opinion needs to be observed so that the player can play and heal themselves when they deem appropriate. As the game progresses and the game play changes, so will the player's decision on when it needs to heal. Looking at this observation, it was given a default value, the same as the other opinions of 1.0f, meaning that the companion heals when its current health is below fifty percent. Through observation of when the player chooses to heal themselves the companion will adjust this opinion. A cautious player that has a tendency to heal after minor injuries will be reflected in a companion that chooses to do the same. At the other end, a more reckless player who likes to push their energy to the limit will be reflected in their companion who will only heal when very necessary.

The number of opinions in this project is relatively low. As only a single scenario is going to be created that demonstrates one of the many cooperative actions that could be created for a cooperative action game, only one tactical opinion was created. I believe that in a larger game, a tactical opinion for each type of strategy should be used and would require its own short training scenario.

3.5 Teaching Combat Behaviour & Scenarios

The teaching component of the game that will appear during training scenarios. A player will be given a selection of different options of how they would like the companion to play the game. It was in the interest of the player that this section of the game should be quick and be a component of the game that is only required of the player to complete once during the game. The best part of the game to introduce it is obviously the start of the game. Because of this I decided that it could be incorporated and disguised as a training scenario. In [17] shows that surveyed players do want a training scenario at the start of every game to introduce them to the controls and feel of the game. Looking at various different training scenarios in other games, COD4, Splinter

Cell: Double Agent, the game is often paused and relevant information is displayed on screen. This information can be about the game environment or can instruct players what buttons to press to move the player or make them shoot etc. This makes this type of action seem acceptable and is used and as such I use a similar design for the layout of the teaching portion of our game.

A scenario will be presented to the player. In this scenario the player and the companion will need to work together to perform a given task. The game will pause every so often to ask the player to select from a list of choices on how they want the companion to play. An example would be to ask the player, "Who do you want to be the leader of the group?" to which the player can choose, "Player" or "Companion". This decision will update an internal opinion of the companion and the scenario will continue. Depending on certain outcomes of what the player selects from these teaching sections, the continuation of the scenario will give the player orders of where to move within the map. This is similar to training scenarios found in other games.

The decision was made of whether to allow the player to bring up a UI during the game in order to train the companion as the game progressed. This idea of long term teaching works well in strategy games like Black & White but I felt that, in an action game, game play should be smooth and free from interruptions. [23] presented that players want all agents to focus on them, and so should the companion. Players also like to be the hero of the game and by tasking them with bringing up a UI to issue commands to other agents deprives them of this feeling.

3.6 Adaptive Opinions

All players of any game is partaking in a lifetime quest for the length of the game, maybe this is reflected in the way their character in the game grows but this is also in they way they play the game. A players skills develop over the lifetime of a game. This means that the companion that is given the player at the start of the game is only complementary to them at that stage of the game. The companion also needs to change during the lifetime of the game in order to constantly be complementary to the player. We will discuss in the next chapter how the adjustment of some opinions were implemented and Section 5.6 looks at the problems face with the constant updating of other opinions and some potential solutions.

Chapter 4

Implementation

The chapter reviews the implementation of the software created for this dissertation and the incorporation of opinion system. We discuss in brief how the game was made in order to give a understanding of the type of game trying to be emulated to be show how the companion will interact with its environment and how it will use the opinion system to select different actions.

4.1 Class Diagram

This section takes a look at the high level class diagram. As explained in the last chapter, the teaching system used to adjust the companion's opinions, would be incorporated in with a training scenario level at the start of the game. For this reason a *Script* class was created that could be used to build a scenario which could be scripted while the player was teaching their companion. The reason for using the scripting system was to remove any intelligence from the companion until certain parts of its opinion had been given relevant values. Once the companion's opinions have been partially formed, its full intelligence is switched on and it is free to start performing actions.

The Script class is created first. It is responsible for the creation of a majority of the game objects including the game map, GUI/Teaching system, player, enemy and companion objects and controller input. This way, it could create a training scenario by loading a map, add the player and companion on the map and finally add the

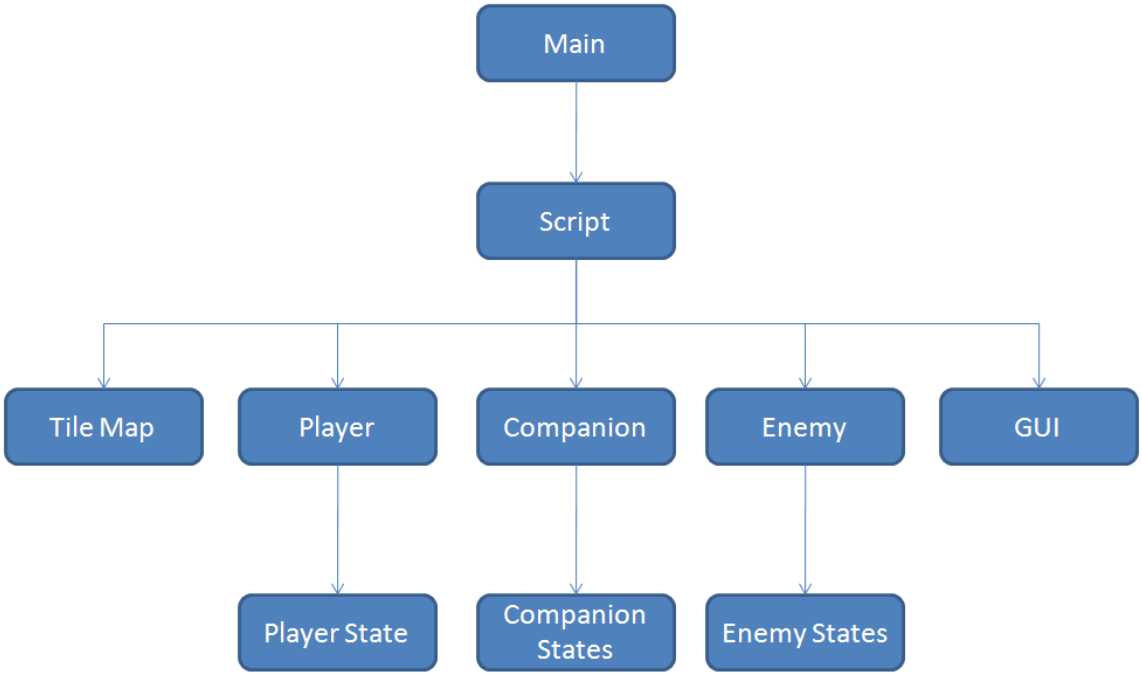

Figure 4.1: High Level Class Diagram

GUI/Teaching system on top of this to query the player about how to adjust the opinion system. For a future game a *Game* class could be used and which would have the same responsibilities as the Script class for creating a working game but removes the need for scripting.

4.2 Game World

To test the opinion system for a companion agent a game environment needed to be designed in which the player can participate and train the opinion. A game arena was created and which required a game play element to it in order to quickly display the companions game play style as its opinions change, e.g. once the tactical opinion is formed we would want to see the companion either sticking with the player or flanking straight away. As discussed before the opinion system would work best in a game with a cooperative element of one player and one companion, rather than a player controlling a full platoon of cooperative agents. The main game I try to emulate is Gears of War [10] as it contains a single cooperative agent for the most part and fits into the third person shooter/action genre of game. One core game play mechanic of the game is the ability for both player and cooperative agents to avail of using objects in the world as cover. This created somewhere to hide from enemies, take cover from incoming fire and somewhere safe to regenerate health. I wanted to emulate this type of system and it produced the idea that by allowing a player to remain in cover for the majority of the game or would produce one type of game play for the companion to use as apposed to just being out of cover and shooting. By giving the player the choice of playing in either style it would be possible to display two very different styles from the player. This would allow me to use the opinion system to show the companion will choose a similar style of game play as the character very quickly. This is where the inclusion of the *Aggression* opinion fits in as it allows the player to play in stealth or in action modes.

The whole map is orientated around a simplistic design to keep the level of complexity low in the anticipation of future complications like pathfinding and level design changes. A two-dimensional world was created as it is quick to design and maintain. The concepts used in the two-dimensional world translate very well to a more detailed three-dimensional world. The two-dimensional world is split into a grid.

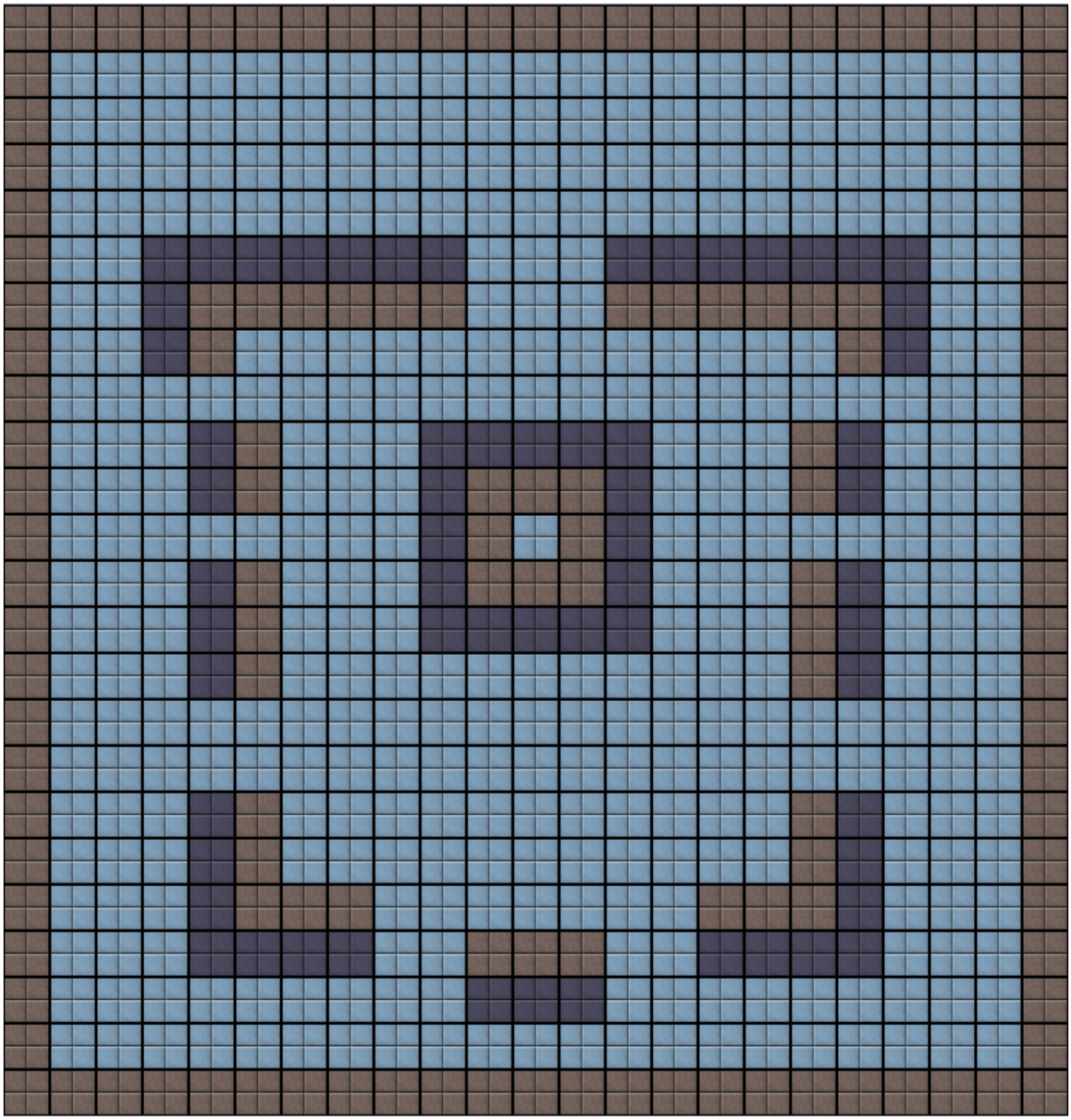

Figure 4.2: Game Map

This grid environment is built as a very simple two-dimensional tile map, Figure 4.2. While the tile map is not very impressive graphically, all methods discussed in the dissertation are easily applied to all First Person Shooter genre and Third Person Shooter genre video games in 3D. A tile map reads information in from a text file and parses it. Each letter in the file represents a tile that will be drawn to the screen. Each tile is a 32 x 32 pixel image and each can be given a separate attribute. Floor tiles can allow players to move on them while wall tiles block the player from moving any further. For the tile map in this game I created a 23 x 24 tile tile map. In my text file I encoded my map and used three different letters to represent different nodes: **f** = floor tile, **w** = wall tile and **c** = cover tiles (see Appendix B Pg.66). In Figure 4.2, **floor tiles** are the blue tiles, **wall tiles** are the brown tiles and **cover tiles** are the dark tiles. Players can walk on both cover and floor tile but not on wall tiles. If a player is standing on a cover tile they can press the right mouse button and their player will duck down thus allowing them to sneak around and hide from enemy fire. A draw function loops through the tile map information and draws it to screen as the game map. This method allows the level to be changed easily as only the text file needs to be updated. The information available in the tile map was enough to allow the creation of some simple collision detection for the player. As the player moves around the map, the four adjacent corner tiles from the players current tile are checked. As the player moves we can check the nodes that appear in front of them and predict whether the player is going to walk into a wall tile. If they are going to we can stop updating the players position. This method is quick and produces pixel perfect collision between the players sprite and all wall tiles. The flexibility of using the tile map is that any changes that need to be made to the layout of the level are easily changed by editing the text file. Subsequently, all logic for collision detection and crouch nodes is maintained. I created a couple of functions in the GUI class that allowed the highlighting of a given tile. These functions are used in the Script class during the scenario stage to tell the player which tile to move to. Also they are used by the companion, when it is the leader, for the same reason. I feel this game world is sufficient to the test the teaching system and the output of the opinion system.

4.3 Opinion System Implementation

1.0f	Opinion	2.0f
Follow	Leader	Lead
Stealth	Aggression	Action
Stick Together	Tactical	Flank
Low	Self-Preservation	High
Low	Accuracy	High

Opinion Table

In the companion class, a struct was created with each of the five opinions. The opinions are initialised in the constructor of the *Companion Class* and given a default value of 1.0f. An update function was created to change any of the values of each opinion (*UpdateOpinion(int opinionNum, float change)*). The function takes an integer and a float as parameters. *opinionNum* corresponds to the opinion that should be updated and *change* is the value that the opinion should be changed by. A simple check was added to make sure the adjusted opinion never got decremented below 0.0f or incremented above 2.0f. It is important that the *change* value at which opinions are adjusted are not too big. If a player has high self preservation and heals after receiving only minor injuries, this is reflected how the companion chooses to heal. If the player enters a situation where they are very low on health before recovering, this is possibly a once off and as such the companion should not start making this as its future decision when choosing when to heal. If this is the player's intention, then the companion should slowly starting adjusting its self preservation opinion to complement the player. A separate function was created which returned the opinion as a struct as the opinion was used in multiple classes.

4.4 Updating Opinions - Teaching System

For the teaching scenario I used a built in GUI API that is available in the game engine, ([44]). A function was created that took a char and a vector. The char contained a message to display the to screen pertaining to the question that the player needed to give a response to. The vector contained a number of options to display below

Figure 4.3: Example Teaching Questions

the message. When an option was selected, the function returned an integer which represented the chosen option. For each of the questions given to the player, only two options are ever passed. The returned integer was used to pick which option was chosen by the player. Each option would alter the same opinion but one represented a increment to the opinion value while the other represented a decrement to the opinion value. As mentioned in Section 3.4, it was decided that only three of the five opinions were capable of being adjusted through the teaching system. In a full game, there would be potentially a lot more opinions and the need for much more interaction with the teaching system. An example of the questions used in the game can be seen in Figure 4.3.

Each part of the scenario was scripted into 18 separate stages. The stages were not linked linearly as they changed according to which options are selected by the player, e.g. the choice in one stage could have 1, 2 or 4 possible outcomes with each moving to a different stage. This process didn't take too long to code as some stages were variations of other and the only functions the companion needed were to tell it were to move to and check when it reached its destination.

4.5 Updating Opinions - Observations

It had originally been intended that all opinions would be taught by the player. When it came to looking at what type of scenarios could be created to teach self-preservation and accuracy it became obvious that it was not possible for a player to teach these opinions. First we will look at the self-preservation opinion. One possible scenario is to give the player a scale from zero to one hundred and ask them to choose where in between these two values is the right value that the companion's health should be

equal to in order for them to run to recover their health. Thinking about this, no one player knows. They may be able to make an informed decision after playing through the game for a while, but even so, as the game progresses and the difficulty increase so does the players decision on how much health they can lose before they need to recover health. For this reason the decision was made to make the self-preservation opinion an observable opinion. Each time the player recovers health after taking damage the companion checks at what level was the players health before they recovered. Initially the companion will recover health after losing fifty percent of its overall health. If the player recovers health after taking only ten percent damage the companion will start to increase their self-preservation opinion to match this. The rate of increase is slow os if the player waits till they have twenty percent of health remaining before they recover health, and in one occasion recover health after only takin five percent damage, the companion will not suddenly start trying to recover health after receiving this small amount of damage.

For a similar reason, the accuracy opinion was taken out of the player's hands and it was made observable. The player is responsible for each bullet it creates and destroys in the game. So it knows when a bullet hits a wall and needs to be deleted or when it hits an enemy and needs to be deleted. When a bullet hits a wall a message is dispatched using the message dispatcher to the companion. It is basically telling the companion, "I missed that shot so you should be more accurate to make up for my poor performance". On the other hand if the player hits the enemy it dispatches a message informing the companion of its good performance and the companion will reduce the value of the accuracy opinion because the player doesn't need as much help shooting. This opinion could possibly be extended so if the player is good enough at shooting, the companion could perform a different action, e.g. throwing grenades, to maximize combined damage.

4.6 Player, Companion & Enemy Actions

Before designing the actions of the companion the details of player actions needed to be described first. As I was focusing on a game like Gears of War the selection of actions is kept quite similar, but more high level, i.e. players have on type of movement, one type of crouching action and one weapon selection. I kept the number of states quite

low in order to reduce the complexity of the project. The movement available to the player is eight directional (N, NE, E, SE, S, SW, W, NW) and collision detection is used to prevent them passing through wall tiles. Crouching is available to the player that is standing on or very close to a crouching tile. A transparent effect will be added to the players sprite to indicate they are in a crouching position. Player will be able to shoot in 360^o using a mouse to aim and shoot. Players who are hurt in battle can flee from the fight and move to a crouch node to regenerate their health. If the player goes four seconds without being hit then their health begins to start regenerating. Any further hits taken during this regeneration period will stop the process. Though not an action available to the player, communication can occur between the player character in the game and the companion and enemy. This will be used to pass messages to tell other agents that a certain event has occurred, e.g. I shot but missed, increase your accuracy. A more detailed version of this message dispatcher is discussed in Section 4.7

In contrast, to reduce complexity, the enemy agent used in the game is only given shooting and communication actions. It can shoot at both the player and the companion and inform both through communication if they die. The enemy is designed to be a static turret weapon that will attack when attacked.

HealthUpdate is a function common to the player as well as the companion. It emulates the health regeneration feature common in a lot of newer action games, e.g. Halo [39] and Gears of War [10]. When a player loses health in these games they can recover it to maximum levels but avoiding getting hit by enemy bullets for an extended amount of time. Each time the player or companion is hit, the current game time is recorded. It overwrites any previous recorded times they were hit. At each frame the system checks if the time since the companion was last hit is over four seconds long. If so, the health begins to recover at a steady rate until it recovers fully. During this recovery period, if the player or companion is hit again, their health stops recovering at the current level and another four seconds must pass, without receiving damage, before the recovery process begins again. The importance of this health system is to show that the self preservation opinion is working properly. During this health regeneration, the companion does not make any actions. The player is free to make actions if they choose. If the enemy was free to move the companion may need to flee from their

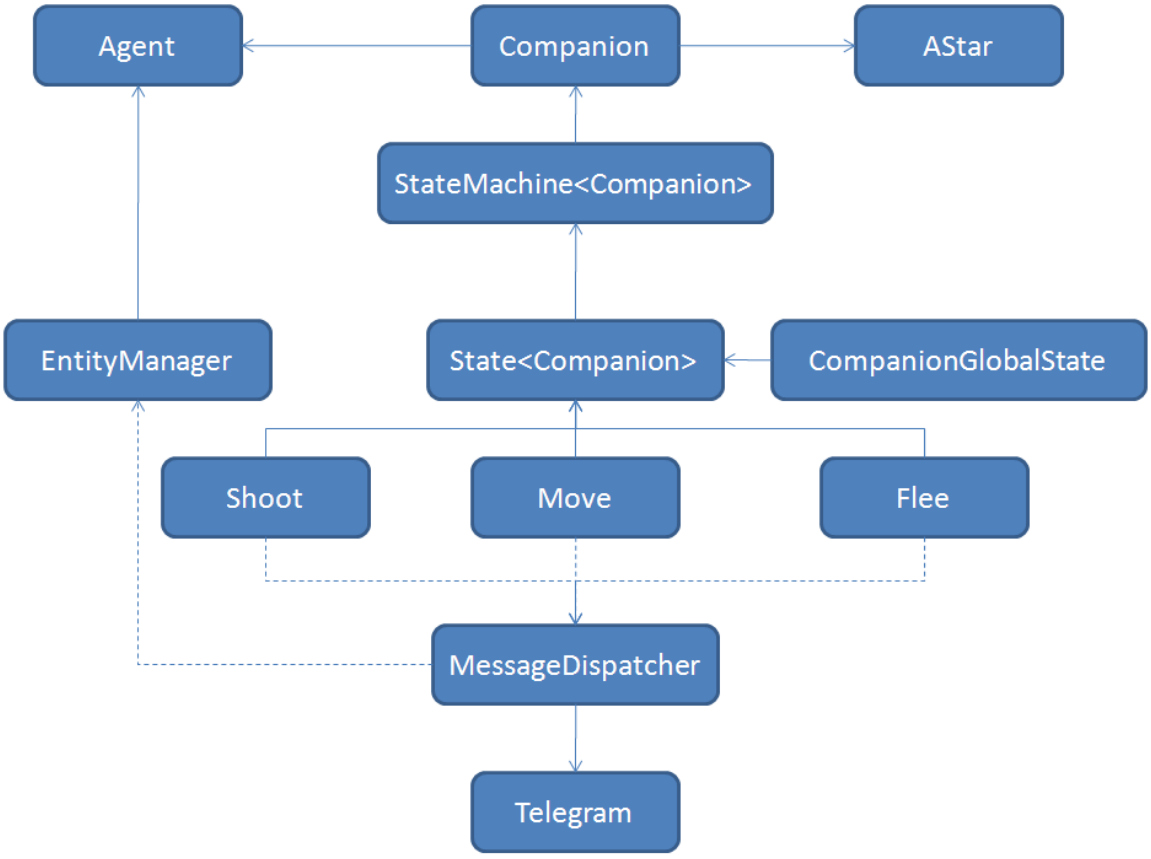

Figure 4.4: Companion Class Diagram

recovering position to fight the enemy or to find safer territory to recover.

4.7 Companion Architecture Implementation

The agent architecture shown in Figure 4.4 follows the design and implementation created by Matt Buckland [41]. This section discusses the functionality of the main classes in general and companion specific term:

Agent Class

The base *Agent Class* is created from which the player, companion and enemy are

all derived. Each derived subclass is given its own unique ID number (for use in the *Message Dispatcher* class below) and must implement an *Update* function that is used to manage the updating of states and a *HandleMessage* function (explained more in the *Message Dispatcher* class below).

Entity Manager Class

As mentioned above, when an agent is created it needs to get a unique ID. How this is handled is once the agent object is created it is passed to the *Entity Manager Class* to be registered with a unique ID. All IDs are stored in a std::map for quick access.

State Machine & State Classes

The *State Machine Class* is a reusable base class that defines the structure for each state. The companion has three state and each has its own unique state class. A *ChooseState* function was added to facilitate the companion's state changing decision (see Section 4.8). At all times a *Global State* is held by the companion. This is the companion's default state and is the state the companion will remain in if nothing satisfies its decision policy.

Message Dispatcher Class

While in any of the its states, the companion can send and receive messages. Received messages can be from the player or the enemy informing the companion to move or that they have been shot respectively. The *Message Dispatcher Class* handles the sending of all messages. It takes several parameters. The first gives an indication of when the message should be sent. Messages can be sent immediately or at a given time in the future. The second and third parameter indicate the sender and receiver of the message. The receiver tells the message dispatched class who to send the message to. This uses the unique ID created for each agent derived from the agent class. The sender indicates to the receiver of the message who sent it to them. The final parameter represents the actual message being sent. The type of message is taken from an enum in the *Telegram Class*. The receiving of messages is first handled in the *Companion Class HandleMessage* function. This function passes the message to the current active state. When a message is received it is compared using a switch statement in order to match the right action to take.

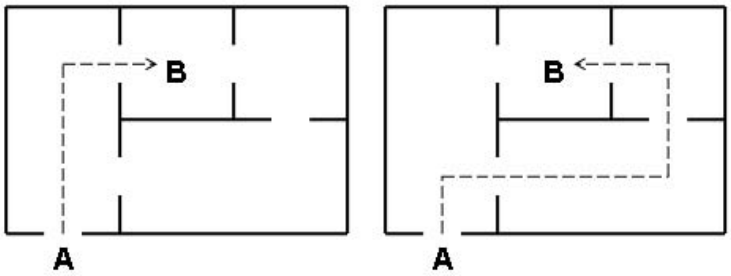

Figure 4.5: Path Finding [7]

A Star Class

Path finding is a part of the implementation concerned with getting the companion's movements working properly, Figure 4.5 . The companion will at many times in the game need to either follow the player or move around the map of its own accord. The need for the algorithm is so that when the companion is moving around the path it can find its way the start of the path to the end of the path while not walking across illegal tiles (walls are the only illegal tiles in this game, other tiles can include mountains or sea). The algorithm also needs to be able to find any path quickly and efficiently. For these reasons the A* path finding algorithm is used [7] because it fits all these criteria. The implementation for this algorithm is quite straight forward but very long to discuss. A full implementation can be found in the source code in Appendix A. If follows the same method described in [45].

4.8 Choosing Opinions - Behaviour Tree

I had originally wanted to try and create a single value from the list of opinions. This proved difficult and would require some extra time. Unfortunately this meant that the companions decision process was long and complex and consisted of a bunch of *if* statements. This meant some of the outcome where left unfinished but still showed enough difference in the outcome to show the addition the opinion system added. I will give a brief break down of the decision process using the opinion values.

There is a priority to the order in which opinions are checked:

1. Self Preservation
2. Leader
3. Aggression
4. Tactical
5. Accuracy

1: Check the companions health. If their health is below the threshold of the opinion then the agent either needs to flee and seek cover to regenerate health or if already in cover, it needs to remain there until the its health has regenerated fully.
2: Check if either the player or the companion is the leader. If the player is the leader, the companion needs to wait for the player to perform an action, e.g. move or shoot, before it can perform an action. If the companion is the leader it can perform an action without having to see if the player has performed an action.
3: This opinion only comes into play when the companion is the leader. It checks to see if the player prefers to play in stealth or action. The value of this opinion results to which path the companion should move along. As is seen in Section 4.9, when the player is the leader, the companion's AI is only smart enough to move along the same path as the player. This would require further improvement in a full game.
4: When the companion is ready to move it needs to know where the player wants it to move to. When the player is the leader the companion will stick to the same path the player is currently on. If the opinion value is set to follow, then the companion finds the node that is closet to the player and moves to it. If the opinion value is set to flank, then the companion finds the node that is closet to the player and then finds a node with the same y position but with an opposite x position. This results in the companion being at the same y position on the map but at the opposite side of the map from the player. When the companion is the leader, it uses the value of the opinion to tel the player which node it should move to. I managed to demonstrate a short example of this method of telling the player where to move. While it would need further improvement it does show the opinion working properly and its intended outcome.

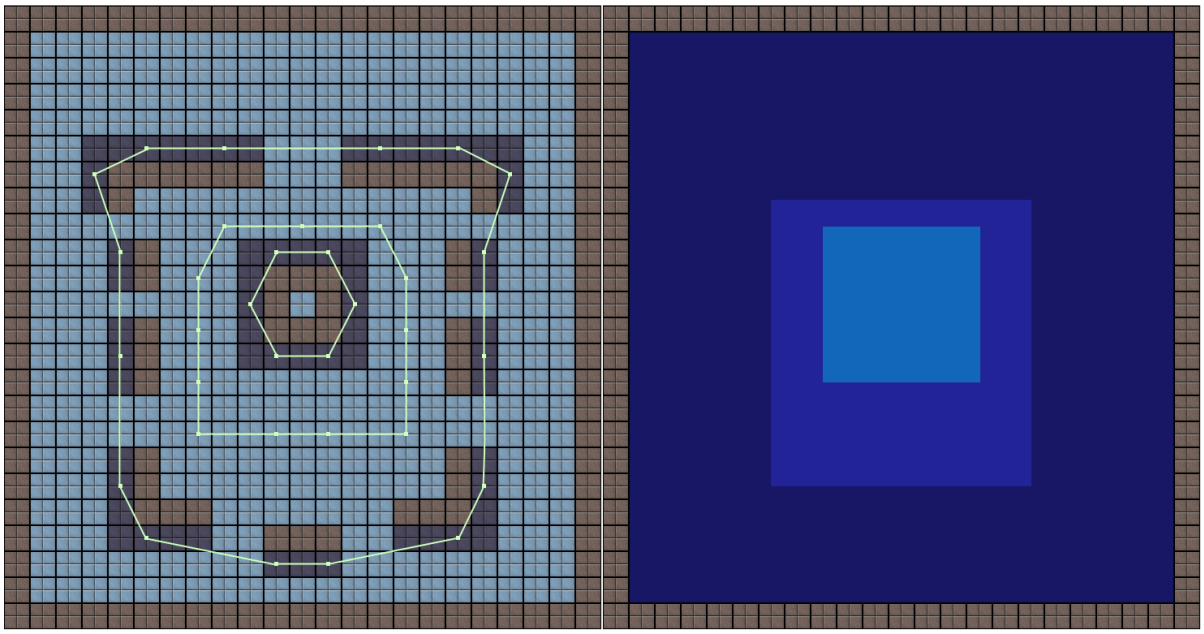

Figure 4.6: Companion Path Selection

5: Lastly the accuracy opinion is looked at when the agent decides to shoot its gun. It takes the value of the opinion and checks the percentage of the accuracy. 2.0f being one hundred percent accurate and 0.0f being zero percent accurate, with everything else being in between. The result was calculated for this project but was never fully integrated into the shooting code. Further changes could be made to this opinion. Its bottom value, currently 0.0f, could be updated so that it never falls below a certain level. This ensures that while the agent is not killing as much as the player, that they're not completely useless.

4.9 Choosing Paths

Because of the different ways the companion could play the game I wanted to demonstrate a selection of paths that the player and companion could follow. Each path is created per design of each level layout.

Stealth Paths

A stealthy player will take advantage of cover that surrounds the outer sides of the

level as a way of getting closer to the enemy without being seen. While in stealth mode the companion choose a path that allowed it to move in between cover and prevent it from being seen by any enemies.

Actions Paths

If the agent created was more aggressive it would opt not to use stealth nodes. In this case the companion can charge into the battlefield and follow a set of nodes that encircle the enemy.

Close Attack Paths

To get even closer to the enemy in order to improve accuracy there was a selection of close attack nodes that the player can crouch in.

Escape Nodes

If the agent is in danger and in the action path they needed to run to cover. A node was positioned between all the gaps in the walls. The agent choose the nearest node to it and move to it. From there the agent is able to move to the nearest node on the stealth path on which it can crouch down and recover health.

Selecting which path to move on depended on the type of companion each player creates. The outcomes that need to be covered are listed below. While there are quite a number of different outcomes a few functions will only need to be created to handle the right paths and nodes to move on. The only addition is how we can get the companion to move between paths given certain situations.

1. Player is the leader and uses stealth
2. Player is the follower and uses stealth
3. Companion is the leader and uses stealth
4. Companion is the leader and uses stealth
5. Player is the leader and uses action
6. Player is the follower and uses action
7. Companion is the leader and uses action
8. Companion is the leader and uses action

Each path is just a two-dimensional array of values corresponding a selection of nodes. The paths are created to fit the layout of the level, see left of Figure 4.6. For this reason they do break if the level layout is changed but the ease of updating the paths does not require much time. While the AI created for path selection is not very intelligent is used to display the basic idea that given a selection of opinions the companion will choose which nodes to move to and on which paths. Again this is more to show how quick the opinion system is in creating a complementary companion. Using the implemented A* Algorithm the Agent can move between any node easily.

When the companion is the follower, it is only tasked at moving to a node relative to the player. If the tactical opinion value indicates that the companion should stick close to the player the companion find out which path the player is on and moves the node closet to the player. On the other side, if the tactical opinion value indicates that the companion should flank as the player moves then it use the same logic as the sticking close opinion in choosing the node but move to the same node on the opposite side of the path.

When the companion is the leader, is has a more direct control over which paths it moves between as it does not require the player to lead it. When the companion is in stealth node it checks how close it is to the enemy agent. If it feels that it is not close enough it moves to the next node that is closet to its current node and closer in distance to the enemy. Once it reaches a node that it feels is close enough to the enemy, it moves to a close attack node in the center of the level which will allow the companion to remain in cover. If the companion is using aggression opinion is action then it moves between random nodes on the attack path. The companion uses the tactical opinion to indicate to the player which node they should move to next, either a node close to the companion or one that would result in the player being in a flanking position. For a complete game, the AI of the companion would need updating but it was still functional enough to show how different the actions it chose were, depending on the values of the opinion system.

When the companion is mortally wounded it will need to find cover in which to duck and regenerate its health. While on the stealth path the agent only needs to remain where it is and duck. While on the attack path the companion looks for the nearest escape node and moves to it. From here the companion can find and move to the nearest stealth node. Similar to the first case, while moving on a close attack

node, the agent can remain on the current node and just duck till it recovers health. Obviously this works in our game where the enemy is static. A better system would need to be improvised in a situation where enemies can move and the companion needs to flee from them even while in cover.

When the companion needed to find which path the player is currently moving along, it was originally seen as being a rather long task. But because of the design of the map and the paths it made the path location a much easier task. Each path is circular in design, left of Figure 4.6. They all circle the center of the map where the enemy is situated, and none of the paths overlap. Therefore I split the level up into three imaginary squares, as seen on the right of Figure 4.6. Each square represents the relative position of a path and by testing if the player is within the center or middle square it can be calculated which of the three square they are in. As such it can be determined which path the player is closest to. While not an ideal solution for a full retail game, this method worked to show the companion following the player along the right path.

Chapter 5

Evaluation

This chapter reviews the finished dissertation software and gives an evaluation of how successful it was. We will first look at overall performance of adding the opinion system and then how the opinion system fit into the genre being studied and what the potential for adaptation of the same system for other genres is. We will then discuss the areas that were not covered and potential solutions followed by the hurdles faced in the development of this dissertation.

5.1 Performance

No tests were created to evaluate the performance overheads from adding the opinion system. As the software was built from the ground up it was built around always having the opinion system. Reviewing the code we can see that there was only one main function to update the opinions. updates would not be occurring generally at a very high rate. The opinion that would be updated most would be the shooting opinion. Not all opinions would need to be checked and updated every frame and the rate of update could be adjusted by the developer. Compared to Fable [36], which used an opinion for a large number or AI agents, this project only needs to handle one opinion for the companion. In this way we can see that the addition of one opinion should not cause much overhead but it should be of interested in a game with a larger number of agents all needing an individual opinion.

5.2 Improvement of FPS & TPS Genres

Within the First Person Shooter and Third Person Shooter genre the opinion system seemed to fit well. These games typically are all about the pace of action. Events are scripted to happen to recreate explosive scenarios in movies. It is therefore important to that the player should not have to pause this flow of action in order to select which action the companion should make, e.g. "Regroup on me", or to bring up a menu and scroll through it to select your companion's weapon for them (seen continually in Mass Effect [11]). The opinion system can be used to take this control out of the player's hands and instill in the player to have confidence in their companion to make decision how they would and to make decisions complementary to how they would.

5.3 Addition in Other Genres

The opinion system could potentially be added to a lot of different genres of video games. The first thing that we need to discuss is whether the teaching system would work in all genres. The second discussion is what is the opinion system going to be used for. In this dissertation the teaching system was always intended to have future used in games where a companion character would be available to the player from the beginning of the game. Games that would fall under the heading of *Action Games*, including FPS and TPS games and that include a companion character could use the opinion system in conjunction with the teaching system very easily. Games that fall under all other genres and do no have companion characters would probably find a challenge in using the teaching system. These games would probably be better using an observation to adjust the opinions. As explain in Section 3.1, there are several instances in different genres of games where the potential application of the opinion system could be applied. Each of these opinions could use the observation system and have no need for the teaching system. In the case of the fighting game, there is no need for the AI to be complementary to the player. Any actions the AI character picks are not going to annoy the player as they are competing. In the real time strategy game the Generals would be learning over an extended time before they are given the chance to actually perform their own actions. Because of this the opinions do not need initial data and are given enough time to adjust.

5.4 Success of Teaching System

The teaching system seemed to work as intended. The idea for including it during it is definitely the best place for its inclusion into the main video game. As was seen from the created game, it only needs to ask one question per question and the time taken to give an answer to the question is very quick. The real advantage that is returned from using the teaching system is that we can get initial data for the opinions. By making all opinions observable we step into a position where the companion does not have any preferences for any of the opinions. One thing discussed in Chapter 2 was that without any initial data the learning agents will have to random choose which actions to make. This would lead to the companion performing action that the player themselves have no preference to and thus annoying the player. This was one of the main situations trying to be avoided and trying improve by using the opinion system.

For this reason I do see a place for the teaching system along side the observation system. It would be better to not have the player teach the companion. It does give the sense that maybe the agent is really dumb if it needs the player to teach it. Using the observation system predominantly would fix this problem but as mentioned above, getting that first initial opinion data helps form a stronger opinion much quicker.

5.5 Increasing Confidence

A goal from the offset was to create a companion that was helpful to weaker players. Through the player's teaching of the opinions they receive a complementary companion and one that takes the lead into situation, sticks with the player and increases their shooting skills if the player is lacking. It was hoped that the companion would complement the skills that a player looks for when player with a stronger player and it was shown that this is very possible.

5.6 Opinion System Improvements

One area that has not been corrected is the ability to easily switch certain opinions, e.g. from a preference of being the leader in the Leader opinion and changing to a preference of being a follower in the same opinion. The player is not allowed to teach

these opinions a second time and as such need to be observed during the rest of the game. This section will review the Leader, Aggression and Tactical opinions to see what kind of observations could be applied to them in order to see them move to the other end of the scale.

Leader - From Lead to Follow

The best way of adjusting this opinion is to observe when a player makes an action. When the companion is the leader it makes its own decision. Before it performs any actions it could set a timer. If the player has not performed any of their own actions before the timer reaches zero then the companion can assume that the player is waiting for it to make a decision. This would lead to the Leader opinion being adjusted in preference of following.

Leader - From Follow to Lead

Similarly to the above observation, the companion could observe how long it takes the player to performs actions. If the time is above a certain threshold then the companion could assume the player wants it to perform an action. When the companion performs the action then the Lader opinion is adjusted with a preference to lead.

Aggression - From Stealth to Action

The observation for this opinion could be updated less regularly than others. Each time the companion observes the player it can check the player's current position. If the companion previously had a preference for using stealth paths but saw the player sticking to action areas of the map with each observation it could then start adjusting the Aggression opinion to reflect this new preference.

Aggression - From Action to Stealth

Similar to the adjustment above, if the companion previously had a preference for using action areas of the map but saw the player sticking to stealth paths with each observation it could then start adjusting the Aggression opinion to reflect this new preference.

Tactical

A serious problem arises when looking at this opinion. If the companion is constantly trying to flank to a position opposite to the player there is no way for the player to get close to the companion. On the other side if the companion is always following the player how does the player get the companion to start flanking.

5.7 Impediments

The major impediment to this project was not having a testing environment available from the start. Various different solutions had been looked at as a way to build on top of previous work, including using the Unreal Engine with Gears of War [10] to create the opinion system. The engine only allows high level scripting and would not support the ability to add new code to it. For that reason a testing environment had to be built to the best of my ability which was a longer than expected process.

Chapter 6

Conclusions & Future Work

In this chapter we will focus on personal conclusions of the project and whether it fulfilled the goals set out in Chapter 1. Following the conclusion section we will discuss potential future work that could be done to improve the opinion system or other areas that the opinion system created in this dissertation could be used.

6.1 Conclusions

The main goals were to create a system that would: 1, annoy the player less; 2, increase player confidence. It is believed that a complementary companion combined with the opinion system will annoy players a lot less. It always considers the player preferences first and this is a key component to improving companion AI in future work. The design to increase player confidence was to allow the to create a companion that they could tell to take on the roles they are not very good at. We saw from the software created that when the player chose a companion that was a leader it moved around the map on its own and moved to tactical attack points ready to engage in a fight with the enemy. Having to worry less about being in complete control, the player can concentrate on try to figure out the game, what they need to do, their playing style etc. and progress as a player. It is hoped that the second time these weaker players play the game from the start that they will revisit the same levels but this time they have gained confidence to lead their companion. Subsequently it is hoped that when they do play with a second, stronger human player that they can provide tips to them

that they learnt themselves.

The opinion system turned out to be a good method of improving the choices the companion made in the game. As was shown at the start of the game, the companion instantly changed its playing style to complement and in cases help the player. Given the programming skills, every player would design their own companion. When not trying to design one behaviour for a companion to suit all players they would design a behaviour that suits only them. The opinion system gives the player this power and it is very important that it does as when the player is playing a game they are priority number one and the AI of their companion should be designed to fit them.

One of the most appealing results of the opinion system was the short amount of time needed to implement it once the right opinions were decided upon. Although it does expect that more care is taken when coding the companions action selection code, it seems to be worth it. It shows that integration of the opinion system into other genres of games would not be too difficult.

This dissertation was able to show that a cooperative companion in a video game does not have to have rigid behaviour. It shows that given a small amount of player input and through the addition of player behaviours each player can be given their own companion. This companion will better suit each player's playing style and as such it is hoped that this improves the player's own opinion of their companion for the better.

There is a sense that for this system to work fully, it relies heavily on the player to complete the scenarios in a truthful manner. Although a player choosing completely random attributes to given the agent will not break the system in any way, they will just get a companion that doesn't complement their playing style in any way. Through constant observation of the player though, the companion should eventually adjust to start playing complementary to them. The quickness that this would happen is not known as all observations were not fully finished but it is assumed that the opinion system increase this rate of learning vastly compared to other methods like neural networks and reinforcement learning.

6.2 Future Work

In the teaching environment, the player can choose an action for the companion to perform and can play and they get to see the resulting action from the companion. They never get to see how the other action plays out. Therefore a nice addition would be the inclusion of a rewind system. If they player picks an action and is dissatisfied with the outcome or just wants to see what the other action for the opinion could be then they could press a button and rewind back. This shouldn't be limited to the last opinion either. The player should be allowed to rewind back to the very first opinion if they like.

When the player is being asked a question in the teaching scenario, a slider could be used to increase the players personal preference for an opinion. For the Leader opinion the player can currently only respond to which opinion they prefer more. Players may like to use a combination of both. Using a slider the player can say that they have a small preference for playing in stealth, and likes to use action sometimes. Or else they could push the slider all the way to show a large preference for stealth which says that they only like to play in stealth.

Could possibly remove try to remove the need for a teaching system. I still see the advantage of the training system as giving the companion an initial evaluation of the player. This means that the companion can say from the get go that it knows for certain that the player likes to use stealth and stay in cover. Knowing this straight away will prevent the companion for taking actions that will see it not using moving along stealth paths and perhaps getting noticed by an enemy the player was trying to avoid. To prevent these types of situations, maybe it would be better for the companion to select safer actions until the corresponding opinion has been trained formally.

A precaution that may need to be taken is if the opinion returns to its original default value (1.0f). In this situation the companion could just randomly choose to increment or decrement the opinion. But this could prevent the opinion from ever moving past the default value. The proposed solution would be to record the last update for each opinion, be it an increment or a decrement. So when an opinion value is checked and it is 1.0f and the last adjustment to it was +0.1f, we could then chose to

repeat this increment. It is possible that this increment was a once off and the player would prefer a decrement but it would be safe to assume that a majority of the time this would be the right action to take.

[46]Bungie's lead AI programmer said, "The problem is not that mistakes are made, but there are no ways to correct it. I have no way of saying, 'No, try again'". He said that is was more of a design problem rather than an AI problem. A suggestion of his was to tell the character, through conversation, that they had or had not made a good decision in their last action. The design of this project is designed so that the player has told their companion that, in situation X do Y. The problem here is when the agent performs action Y but dies. Was it down to the player that the agent died? Was it the decision the player had given the companion that was to blame? In cases like these there needs to be a change made.

One of the intentions of this dissertation was to create companions that would increase the confidence of weaker gamers. These gamers may be used to playing with a relative and as such like how this person plays the game with them. As such it would be a good idea to have the computer create opinions of all the people who played the game. When the players finish playing the game, they can save this created opinion under a profile, e.g. Dad, Mum, Johnny etc. The player could then select this opinion to be their selected companion when they play the game the next time and that other person is not around.

Lastly it would be interesting to see how the companion could form opinions of each of the different types of enemies in the game. As a player journeys through the game, they learn that each type of enemy has different strengths and weaknesses. They know for instance that a smaller enemy is likely not going to be very strong so they might be able to get close to them in a gun fight. Adversely, a larger enemy should be avoided in close combat. Enemies also have proficiency with certain weapons or have weak spots on their body, which if attacked, cause more damage. Opinions of all this information is learnt by the player as the game progresses, it something game developers work hard on to try and make obvious [23]. In the same way, companions could form these opinions are the game progresses. So just as the player might lose a

fight when faced with a new enemy, the second time they encounter this enemy they will have a new strategy of how to kill them.

Appendix A

Appendix

Attached is a CD with the source code for this dissertation project.

Appendix B

Appendix

Figure B.1: Tile Map Text File

Bibliography

[1] M. Buckland, "Neural networks in plain english," http://www.ai-junkie.com/ann/evolved/nnt1.html [Online; accessed 2-July-2008].

[2] K. O. Stanley, B. D. Bryant, I. Karpov, and R. Miikkulainen, "Real-time evolution of neural networks in the nero video game," in *In Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006.

[3] F. Liu and Guangzhou, "Multi-agent cooperative learning research based on reinforcement learning," in *Proceedings of the 10th International Conference on Computer Cupported Cooperative Work in Design*, 2006.

[4] P. J. Gmytrasiewicz and E. H. Durfee, "Rational coordination in multi-agent environments," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 4, pp. 319–350, 2000.

[5] H. Qiao, J. Rozenblit, F. Szidarovszky, and L. Yang, "Multi-agent learning model with bargaining," in *WSC '06: Proceedings of the 38th conference on Winter simulation*, pp. 934–940, Winter Simulation Conference, 2006.

[6] D. Suryadi and P. J. Gmytrasiewicz, "Learning models of other agents using influence diagrams," in *UM '99: Proceedings of the seventh international conference on User modeling*, (Secaucus, NJ, USA), pp. 223–232, Springer-Verlag New York, Inc., 1999.

[7] B. Yue and P. de Byl, "The state of the art in game ai standardisation," in *CyberGames '06: Proceedings of the 2006 international conference on Game research and development*, (Murdoch University, Australia, Australia), pp. 41–46, Murdoch University, 2006.

[8] J. Rossignol, "The joy of co-op," July 2008. http://www.rockpapershotgun.com/2008/07/28/the-joy-of-co-op-2/ [Online; accessed 28-June-2008].

[9] M. G., "Blacksite: Area 51 drops online co-op play like a bad habit," 2006. http://ve3d.ign.com/articles/news/35007/BlackSite-Area-51-Drops-Online-Co-op-Play-Like-a-Bad-Habit [Online; accessed 15-July-2008].

[10] E. Games, "Gears of war," 2006. http://www.gearsofwar.xbox.com/ [Online; accessed 10-August-2008].

[11] BioWare, "Mass effect," 2007. http://masseffect.bioware.com/ [Online; accessed 30-August-2008].

[12] Valve, "Left 4 dead," 2008.

[13] Capcom, "Resident evil 5," 2008. http://www.residentevil.com/ [Online; accessed 21-August-2008].

[14] P. Sweetser, D. Johnson, J. Sweetser, and J. Wiles, "Creating engaging artificial characters for games," in *ICEC '03: Proceedings of the second international conference on Entertainment computing*, (Pittsburgh, PA, USA), pp. 1–8, Carnegie Mellon University, 2003.

[15] D. B. Fogel and T. J. Hays, "A platform for evolving characters in competitive games," in *In Proceedings of 2004 Congress on Evolutionary Computation, 14201426. Piscataway*, pp. 1420–1426, IEEE Press, 2004.

[16] Ubisoft, "Tom clancy's splinter cell: Double agent," 2006. http://www.splintercell.com/ [Online; accessed 04-August-2008].

[17] P. Sweetser and P. Wyeth, "Gameflow: a model for evaluating player enjoyment in games," *Comput. Entertain.*, vol. 3, no. 3, pp. 3–3, 2005.

[18] S. Zanetti and A. E. Rhalibi, "Machine learning techniques for fps in q3," in *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, (New York, NY, USA), pp. 239–244, ACM, 2004.

[19] M. Buckland, "Genetic algorithms in plain english," http://www.ai-junkie.com/ga/intro/gat1.html [Online; accessed 2-July-2008].

[20] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, p. 2005, 2005.

[21] Wikipedia, "Genetic algorithm," 2008. `http://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=233736792` [Online; accessed 25-August-2008].

[22] id Software, "Quake 3," 1999. www.idsoftware.com/games/quake/quake3-arena/ [Online; accessed 10-August-2008].

[23] J. G. Chris Butcher, 2002. The Illusion of Intelligence.

[24] A. Champandard, "The secret to building game ai that learns realistically," 2007. http://aigamedev.com/architecture/learn-realistic [Online; accessed 25-August-2008].

[25] P. Huang and K. Sycara, "Multi-agent learning in extensive games with complete information," in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, (New York, NY, USA), pp. 701–708, ACM, 2003.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

[27] K. Driessens, "Relational reinforcement learning: Thesis," *AI Commun.*, vol. 18, no. 1, pp. 71–73, 2005.

[28] M. Bowling, "Convergence problems of general-sum multiagent reinforcement learning," in *In Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 89–94, Morgan Kaufmann, 2000.

[29] D.-Y. L. Jing Huang, Bo Yeang, "A distributed q-learning algorithm for multi-agent team coordination," in *Proceedings of the 4th International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005*, 2005.

[30] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," tech. rep., 2004.

[31] Wikipedia, "Prisoner's dilemma," 2008. `http://en.wikipedia.org/w/index.php?title=Prisoner%27s_dilemma&oldid=233297560` [Online; accessed 25-August-2008].

[32] T. Makino and K. Aihara, "Multi-agent reinforcement learning algorithm to handle beliefs of other agents' policies and embedded beliefs," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, (New York, NY, USA), pp. 789–791, ACM, 2006.

[33] C. Boutilier, "Learning conventions in multiagent stochastic domains using likelihood estimates," in *In Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pp. 106–114, 1996.

[34] Wikipedia, "Bayes' theorem," 2008. `http://en.wikipedia.org/w/index.php?title=Bayes%27_theorem&oldid=234104483` [Online; accessed 29-August-2008].

[35] A. Russell, "Opinion systems," *AI Game Programming Wisdom 3*, pp. 531 – 554, 2006.

[36] L. Studios, "Fable," 2006. http://lionhead.com/fable/Default.aspx [Online; accessed 10-August-2008].

[37] Namco, "Tekken," 2008. http://www.tekken.com/ [Online; accessed 29-August-2008].

[38] Namco, "Soul calibur," 2006. http://www.soulcalibur.com/ [Online; accessed 29-August-2008].

[39] Bungie, "Halo," 2008. http://www.bungie.com/ [Online; accessed 29-August-2008].

[40] D. Clarke and P. R. Duimering, "How computer gamers experience the game situation: a behavioral study," *Comput. Entertain.*, vol. 4, no. 3, p. 6, 2006.

[41] M. Buckland, *Programming Game AI by Example*. Wordware, 2005.

[42] A. Champandard, "Game ai innovation in fable 2," 2007. http://aigamedev.com/coverage/innovation-fable-2 [Online; accessed 20-August-2008].

[43] D. Extremes, "Dark sector," 2008. http://www.darksector.com/ [Online; accessed 30-August-2008].

[44] Haaf, "Haaf's game engine," 2008. http://hge.relishgames.com/ [Online; accessed 10-June-2008].

[45] S. Rabin, *AI Game Programming Wisdom*. Charles River Media, 2002.

[46] Gamasutra, "In depth: Bungie on eight years of halo ai," 2008. http://www.gamasutra.com/php-bin/news_index.php?story=19653 [Online; accessed6-July-2008].