Q-Boltz Millennium Wars

A Campaign version of Quake which features enemies controlled by a learning AI.

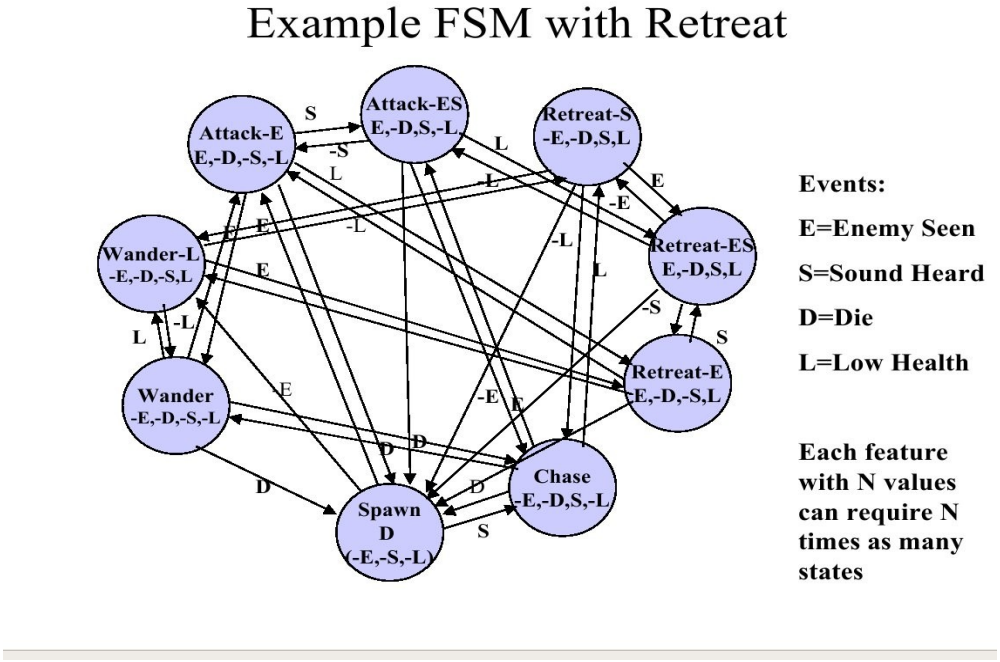
By: Emuye Taylor and Brandon Lees

Software Engineering Analysis

In first person shooter games currently available, the artificial intelligence used to control non-player characters (NPC) is static. Even the most advanced AI present in first person shooters, such as Half-life 2, is very limited in it's ability to adapt. While NPC's in Half-life 2 are able to react to actions of the player in an almost human-like manner, the reactions are pre-defined. An example of this behavior is that while an NPC may be able to duck from a player shooting at them, they would never learn to adapt to the player's specific pattern of shooting.

Many traditional approaches to AI for first person shooters is based on a Finite State Machine (FSM) design. This approach models the AI's functionality (Wander, Attack, Retreat, Chase, etc) as states in an FSM. In order to create more complicated AI, more states are added to the FSM to represent more fine grained actions. In order to create AI that is more difficult to play against, the AI's accuracy is improved. Another method used to make more challenging AI is to allow the AI to perform actions which the player is unable to. For example, the AI might be given a speed bonus or a damage bonus. From a player's point of view, the shortcoming of this approach is that regardless of difficulty, the AI will play the same. An additional shortcoming is that at higher levels of difficulty, for the player, the

game play can feel as if the AI is cheating.



To address these problems in current Artificial Intelligence we plan to implement a more dynamic AI which is based on learning. The goal of this implementation is to offer a game which provides a more realistic and challenging experience to the gamer. This AI will be able to react not only to a player's individual actions but also to patterns in these actions. The end result will be a game in which the player will be able to play against NPCS that are able to adapt and react to the player's style of play.

Software Engineering Design

Below is a description of the design plan for several of the components for this project.

Game Model:

In order to create an AI that can adapt to the player, we will model the game as a Markov Decision Process (MDP). An MDP consists of a set of states, a set of actions, a matrix of probabilities of state transitions, and a reward function. Using a reinforcement learning algorithm (see below), we will generate and optimal strategy for the AI to follow.

States:

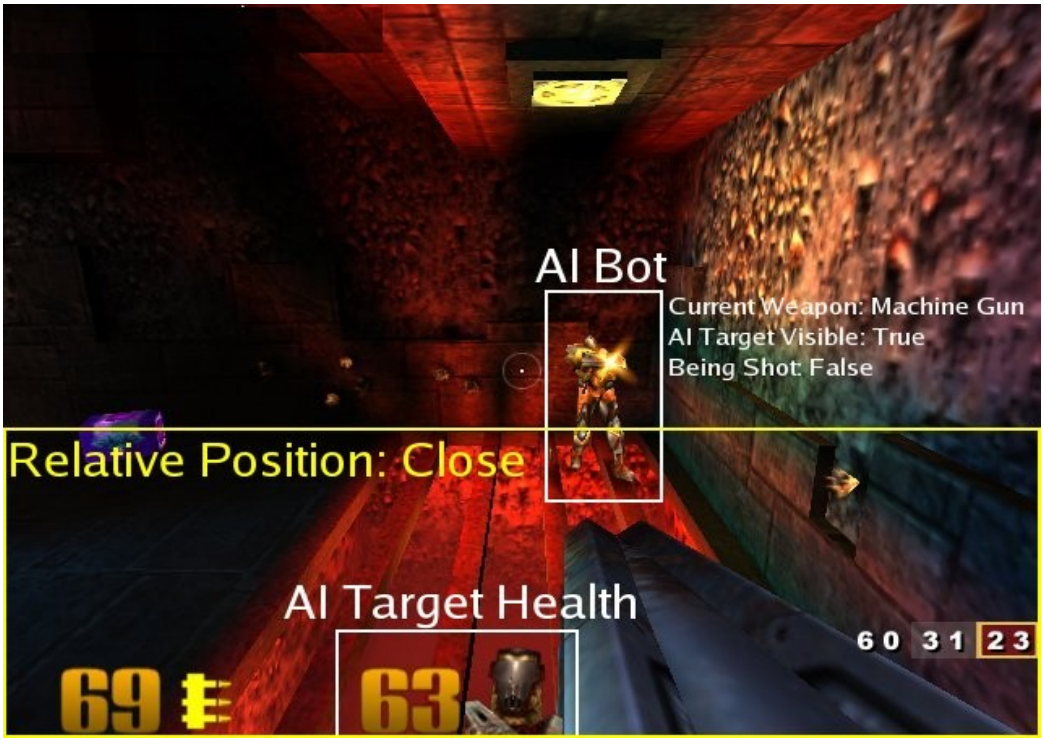
The goal of the states is to create an accurate representation of the game while containing only information relevant to the AI's actions. A trade-off exists between the size of the state space and the speed and accuracy with which the AI will be able to learn from it's environment. A smaller state space would imply that each state will be visited more often, which would result in more accurate learning of optimal actions from each state. Additionally, it would take less time for the AI to learn

these optimal actions. However, a small state space would group many slightly different situations into a single state. This might lead to the AI doing something less that optimal in certain situations. A larger state space would allow for more differentiation and therefore more fined-grained control of the AI's actions although there is a drawback of potentially slower learning. We have created a state representation that we think accurately represents the world but is limited enough to allow efficient learning.

The states which will be used to model the game will be structures we define. During game play information regarding the players activity will be used to create the states. In order to limit the number of states values such as enemy count, health level, etc. will be in predefined levels as opposed to actual counts.

State Content:

- Relative Position to Target : Far, Medium, Close
- Relative Position to Cover: Far, Medium, Close
- AI Health Level: 90, 80, 70, ...
- AI Target Health Level: 90, 80, 70, ...
- Current Weapon: Shotgun, Rocket Launcher, Rail Gun, ...
- Target Current Weapon: Shotgun, Rocket Launcher, Rail Gun, ...
- Being Shot
- Is AI Target Visible



Relative Position to Enemies/Cover:

In order to determine relative position, the area surrounding the AI will be represented by a grid. This grid will be a 2d array and we will define the position of the target and cover as shown below. We will map all possible locations to only the positions shown below using predefined thresholds.

AI Health Level:

To model the AI's current health level we will use predefined threshold levels.

AI Target Health Level:

To model the AI target's current health level we will use predefined threshold levels.

Current Weapon:
In order to model the AI's current weapon selection we will map a number to each weapon, and each game state will include this number.

Target Current Weapon:
In order to model the AI target's current weapon selection we will map a number to each weapon, and each game state will include this number.

Being Shot:
We will use a boolean value that indicates whether or not the AI is being shot.

AI Target Visible:
A boolean value to indicate whether the AI's target is visible.

Actions/Transitions:
Move towards target and cover
Move towards target and away cover
Move away from target and towards cover
Move away from target and cover
Move towards target
Move away from target
Move towards cover
Move away from cover
No movement

Sub-Actions:
- no sub action
- shooting
- changing weapon

Action Directions:
In this game we will not consider absolute direction. For the purposes of the AI, only movements relative to specific objects are important, ie towards/away from target. Therefore, we will use a grid representation in combination with movements relative to object positions to account for AI direction.

Reward Function:
The reward function generates a utility value for a given state and is used by the learning algorithm to determine the relative value of different actions. For our game AI, the primary goal will be to kill the

player while losing as little health as possible. To represent this, our reward function will be based on the AI's health and the AI target's health. Since the ultimate goal is to kill the target player, there will be a small bonus added to states where the AI's target is dead. Additionally, to promote survival behavior for the AI, a death penalty will be incorporated into the reward function for states where the AI is dead. In order to promote good weapon choices, a weapon selection bonus will be added which is determined based on surroundings and distance to the AI's target. For example, a bonus would be added if the AI were using a shotgun in very close combat or a rail gun when very far from it's target.

R = AI health – AI target health + Kill Bonus + Death Penalty + Weapon Choice Bonus

Markov Decision Process:

A Markov Decision Process (MDP) is a discrete time stochastic control process characterized by a set of states, actions, and transition probability matrices that depend on the actions chosen within a given state. It has the Markov property - the past is irrelevant for predicting the future, given knowledge of the present.

Learning Algorithm:

We will be using Q-learning for our learning algorithm. To calculate Q-values we will repeat the following algorithm.

1. From the current state s, select an action a. This will cause a receipt of an immediate reward r, and arrival at a next state s'.
2. Update $Q(s,a)$ based upon this experience as follows:

   $$Q(s,a) = Q(s,a) + x[r + y \max(Q(s',b)) - Q(s,a)]$$

   where x is the learning rate and $0 < y < 1$ is the discount factor

   x: We will start with a learning rate of 1 and make adjustments as necessary.

   y: It will require experimentation to determine an appropriate value of the discount factor. We will initially start with a lower discount factor.

To determine the action in the first step, we will use the Boltzmann distribution strategy. This selection strategy favors actions which have high Q-values, however all actions have a nonzero probability. This will ensure that the entire action space is explored which guarantees that the optimal strategy will be found. Additionally, this strategy allows the AI to adjust to changes in the target's play to allow adaptations to the optimal strategy.

Boltzmann Distribution:

The AI will try actions probabilistically based on their Q-values using a Boltzmann distribution. Given

a state s, it tries out action a with probability:

$$P_s(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_j e^{\frac{Q(s,a_j)}{T}}}$$

The temperature T controls the amount of exploration (the probability of executing actions other than the one with the highest Q-value). If T is high, or if Q-values are all the same, this will pick a random action. If T is low and Q-values are different, it will tend to pick the action with the highest Q-value. High values of T would cause the AI to take more random actions, even if some actions have high Q-values. A low value for T could result in less exploration of the state space and sub-optimal policies. T values that decrease with time could better ensure that the optimal policy is found, and that once this occurs, the AI will continue to use this optimal policy. We will experiment with both high and low constant values of T as well as values that decrease with time.

AI Actions:

In order to determine the actual AI's actions we will need to convert the abstract action from the Q-learning to actual movements. This convergence will need to take into account object avoidance and simple path-planning.

Quake Integration:

We will be using the open-source Quake III engine to develop our game. The significant change we will be making to the engine is to alter the AI to use our learning algorithm. The engine includes a basic AI which is used in the game to control bots. We will be creating an alternative AI which we will use instead to control the enemies.

Because our game will be a single player campaign as opposed to a multi-player like Quake we will need to make additional changes. We will utilize existing Quake III maps which are more suited for campaign levels as a starting point. We will modify these maps to create the actual maps which we will use in our game.

Testing

Overview:

Testing this project will be completed through out the entire life cycle of the project. All of the individual components for this project will be tested once as stand alone pieces, and then again after each step of integration for this project. This level of testing will consist of functionality testing for all classes and code written for the components. Another level of testing which must be completed for this project consists of testing the code through game play. While the first level of testing will focus on testing individual components, the second level will focus on testing the project as a whole.

State Generation:

Once the state generation code is fully integrated into the project the function will be to query the game for all of the data which must populate the state. However, to test this code as a stand alone component we will hard code game scenarios with which to compare the generated states.

AI Learning Algorithm:

The focus of this testing is to ensure that the AI algorithm correctly changes the Q-values associated with transition actions based on the reward function. To test this component we will randomly pick an action and simulate its occurrence, passing values to the AI algorithm as if it occurred in the game. We will run many iterations of this test and verify that the Q-values associated with the actions have changed to signify the rewards of each action.

Action Convergence:

The focus of this testing is to ensure that wall detection, path-planning, and other AI checks are working properly. Additionally, we will use this test to ensure that the QI is appropriately performing actions as determined by the Q-learning. This testing will involve actually playing the game and observing the AI's actions and debugging the Q-learning output.

Testing of AI with different discount and learning factors:

To ensure that we are using the most effective discount and learning values we will experiment with many different values during this phase of the testing. We will observe the AI actions and demonstrated learning given these different values. Additionally we will compare this behavior with a control policy which simply picks random actions for the AI.

Game Play/Performance Testing:

The focus of the first round of game play testing is to test that the AI adapts to various game play strategies. Also, we will test the overall performance of the AI over several matches between the AI and human players. In addition to the developers playing the game, other individuals in the target age group for this game will be brought in to play and demonstrate their strategies. An additional major concern to address with this testing is the map layout. Some issues which will be addressed are the placement of enemies and cover, and the paths used to complete the map.

Final Testing:

The focus of this testing will be to verify the overall game experience. We will spend a lot of time playing the game and trying to break our code. Additionally, we will bring in other people to play the game to identify any final bugs or problems with the code.

Timeline

Date	*Goal:*
3/10	Project Proposal final draft approved
3/17	State generation code completed and tested
3/24	Begin AI Learning Algorithm
3/31	AI Learning Algorithm completed and tested
4/7	2 completed maps
4/14	Action convergence code completed and tested
4/21	Testing of AI with different discount and learning factors
4/28	Game play testing complete/Performance
5/5	Final testing and modifications mostly complete
5/12	Game complete

Rubric

While the overall performance of the AI is important to this project. This is not criteria which should be used to evaluate our grade. The important element to this project is that the AI learns and adapts to different users. We certainly hope that our approach will create AI which is more effective in killing human players than an FSM approach, or one that uses random actions. However, because this is a new, innovative approach, we can not make any guarantees about the extent to which this will happen. The criteria we define to evaluate our project is centered around the AI's ability to learn and adapt. Unfortunately, the most efficient criteria which we can use to demonstrate our AI is its ability to reduce the human player's health. As a result, we will incorporate this as criteria to the A grade, but B and C will be primarily centered around the AI learning without this demonstration.

A: The state generation and AI learning algorithm code is complete. These game components have been integrated with Quake III. The AI demonstrates more advanced learning, which means that over time the Q-values change dramatically. Additionally, the AI shows observable improvement over random actions in reducing the players health. During game play there are no observable major bugs.

- Code completely
- Quake III integration
- AI demonstrates more advanced learning/Q-values change dramatically over time
- AI shows observable improvement over random action AI
- Code has been fully tested and contains no major bugs

B: The state generation and AI learning algorithm code is complete. These game components have been integrated with Quake III. The AI demonstrates minimal learning, which means that over time the

Q-values associated with state transitions change. Game play shows that the code has some bugs.

- Code complete
- Quake III integration
- AI demonstrates minimal learning/Q-Values change over time
- Code has some bugs

C: The state generation and AI learning algorithm code is complete, however these game components have not been integrated. Although each component has been completed, instead of a game we have developed components which could be used in a game.

- Code complete
- Components but no game integration

Breakdown

Because there are only two people working on this project a breakdown is not necessary. Most of the work will be done in collaboration. Usually there will be one person coding and other looking on to ensure code integrity. Both members will serve as both programmer and overseer.

Related Work

As discussed previously, the AI the we will develop for this game will differ from existing first person shooter enemy AI. Although it is difficult to find specific information on the implementation techniques used to create the more advanced AI in newer commercial games, we have compiled some information.

Halo 2 AI:

From an Interview with the designer of the Halo 2 AI, Chris Butcher:

The AI functionality for enemy bots in Halo 2 is modeled as actions in an FSM. The four states used in this FSM are:

- Idle
- Guard/Patrol
- Attack/Defend
- Retreat

Each character has a specific set of actions which it can perform. Actions are chosen based on the AI's environment and its character type. "... decisions about what the most appropriate thing to do is made based on the knowledge of what is going on in the world and knowledge about the type of character the AI is. For example, at that top-most level, it should say, 'If there are enemies I can see, then I should be engaging them in combat.' But if the AI is a cowardly character, it might say, 'If I am faced with

overwhelming force, then I will retreat.' That is the level where our game designers come into play. They have access to all this game information and to all the numerical quantities that control the behavior of the AI."

Additionally, the AI's actions are completely static. The developers tried to create AI which would be predictable. "We don't do things by random chance very much. The goal is not to create something that is unpredictable. What you want is an artificial intelligence that is consistent so that the player can give it certain inputs. The player can do things and expect the AI will react in a certain way. "

F.E.A.R.:

The AI for F.E.A.R is far more advanced then most AI for first person shooters, in some ways. The focus for this project was to have realistic team interactions for the enemy players. Additionally, the AI can perform a variety of actions in any particular setting as opposed to formulaic responses. Although, these actions may be determined by the human player's actions in part, the AI does not learn from the human player. This approach improves current AI in that it allows a more dynamic feel to the enemy players. However, the extent to this dynamism is limited to action response and does not incorporate adaptation over time.

Quake 3 AI:

The AI functionality for enemy bots in Quake 3 is modeled as actions in an FSM. This approach is very similar to the approach used for Halo 2. However, Quake 3 AI also uses Fuzzy Logic to determine the AI's specific goal. Although this method is somewhat more advanced then that used for Halo 2, the Quake 3 bots demonstrate very static behavior.

Soar Quakebot:

The Soar Quakebot is an AI based on the Soar AI architecture developed by John Laird. Soar is an architecture for developing systems that exhibit intelligent behavior. The Quakebot uses the Soar architecture to control a character in a Quake II death match that reacts to other players and incorporates tactics via hierarchical goals. The AI is based on a set of rules that are grouped into tactics such as Collect powerups, Attack, Retreat, Chase, Ambush, Hunt. For each tactic, there are several hundred rules which the AI uses to determine what action to take. For example, to Attack, there might first be a sub-rule to face the target, then move towards the target, then fire at the target. The Soar architecture uses these rules to make intelligent decisions about how to react. This differs from the approach of our AI since our AI defines only basic actions and the AI will learn how to react based on a reward function, not on predefined rules.

N.E.R.O.

Neuro-Evolving Robotic Operatives (N.E.R.O.), a game developed my researchers at UT-Austin allows players to train a team of robots and then have them fight robots trained by other players. The algorithm used for the AI, Neuroevolution, is based on a neural network and a genetic learning

algorithm. The algorithm rewards agents in the game that perform the best and punishes those that perform the worst. The rewards and punishments are specified by the player based on how the player wants the agents to act. The AI for N.E.R.O. is similar to the AI for our project since it allows the agents in the game to learn and adapt based on actions of the player. Our game, however, is a first person shooter whereas N.E.R.O. is a real-time strategy game. Also, the goal of our AI is to learn the best strategy while the goal of the AI for N.E.R.O. is to learn how the player wants the AI to respond.

Full Spectrum Warrior:

Full Spectrum Warrior is a combat game with AI based on research done by Michael van Lent. The goal of the AI is for the computer controlled agents to emulate real soldiers as much as possible. A large focus is placed on team interactions to allow the player to act as part of a real squad of soldiers. While the AI allows for very realistic team interactions, the AI does not learn from the player. The goal of our project is different since we have no interest in realism, we want our AI to learn the best strategy for playing the game.

Q-BMW vs. Existing Games:

The AI for Q-BMW will be drastically different from existing first person shooter AI. Q-BMW will feature AI which is dynamic. The control policy to determine specific actions will be a form of reinforcement learning. This will allow the AI action policies to changer depending on a users style of play. This approach will create enemies which have less predictable behavior and adaptive strategies. Additionally, to create different difficulty settings in this game we will be able to adjust the learning for the AI instead of using AI which cheats. These changes will create improved AI that offers a more fun, challenging, and new game experience.

Related Work Links:

http://www.idsoftware.com
http://www.planetquake.com/quake3/
http://en.wikipedia.org/wiki/Quake_III_Arena
http://half-life2.com/

References

http://www.cs.ualberta.ca/%7Esutton/book/ebook/node65.html
http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/zah/article2.html
http://www.cs.northwestern.edu/~forbus/c95-gd/lectures/Game_AI_Introduction.pdf
http://en.wikipedia.org/wiki/Markov_decision_process
http://www.planethalflife.com
http://www.computing.dcu.ie/~humphrys/PhD/ch2.html
http://www.stuffo.com/halo2-ai3.htm