Cerberus: Applying Supervised and Reinforcement Learning Techniques to Capture the Flag Games

Ahmed S. Hefny[1], Ayat A. Hatem[2], Mahmoud M. Shalaby[3], Amir F. Atiya[4]

Computer Engineering Department - Faculty of Engineering, Cairo University
[1] ahefny@eng.cu.edu.eg, [2] ayat.dawood@nileu.edu.eg
[3] mmshalaby@aucegypt.edu, [4] amir@alumni.caltech.edu

Abstract

Applying machine learning techniques to real-time computer games is an active research field. In this paper we present Cerberus, a machine learning framework for team-based **C**apture **T**he **F**lag (**CTF**) games. This framework utilizes reinforcement learning to select high-level actions that achieve best team behaviour and utilizes neural networks to control fighting behaviour of team individuals. Our proposed framework also combines waypoints and influence maps for effective path planning.

Keywords

Machine Learning, Reinforcement Learning, Neural Network, CTF, Capture the Flag, FPS, First Person Shooter, AI Architecture.

Introduction

Computer games have become a very popular mean of entertainment. Many advances in computer science were driven by problems confronted in games. Earlier games were usually turn-based, such as chess and backgammon. Currently, real time games are getting more advanced and more realistic. Not only do these games constitute a huge industry of billions of dollars, but they have also attracted the attention of AI researchers because they are a perfect test-bed for simulating real world problems. AI techniques of real-time games have advanced significantly in order to create more challenging games; however, their advancement lagged somewhat the progress in graphics technology. Expert systems, fuzzy logic and finite state machines have been successfully applied in real time games (van Waveren, 2003). However, these techniques usually result in a static behaviour that does not adapt to the opponent's strategy. If the human is able to exploit a weakness in the game's AI, the player will lose interest after playing a few games due to the waning challenge. Moreover, as games become more sophisticated, developing a game's AI becomes a more difficult task and the chance of introducing weaknesses becomes greater. Therefore, there is a strong motivation to use machine learning techniques in real-time computer games, such as First-Person-Shooter games (FPS). There are two models of applying machine learning in games:

- Offline Learning:

In the offline learning model, the learning process occurs only during the game's development. The AI system can learn from an expert or by trial and error until it reaches the desired performance level. In the released game, the AI system only exploits what it has learned. Here, learning is used as an alternative to the development of a complicated AI program.

- Online Learning:

In the online learning model, the agent still learns during the operation of the released game. This continuously adapts the game AI to the opponent's strategy and skill.

In this paper, we present a system that applies both models in the Capture the Flag game (we call this system Cerberus). We chose the CTF game because it represents a sophisticated game that requires intelligent team management to achieve two contradicting sub-goals (getting the enemy's flag and defending the team's flag) as well as intelligent individual behaviour. We develop a machine learning framework to satisfy these requirements. Specifically, we develop a reinforcement learning model as our online learning model to have AI agents learn high level actions to achieve the best team behaviour, and we use neural networks to have an AI agent learn the low level behaviour from a human trainer in an offline learning setup. We show that applying machine learning techniques can result in a competitive CTF team without complicated AI hard-coding or scripting. We start by a brief overview of related work. Then, we illustrate the overall AI architecture of the proposed framework which we follow by a detailed explanation of each component. Finally, we present the experimental results and the conclusion.

Related Work

Application of machine learning methods in games has been investigated by many previous research works such as (Ahmed, 2002), (Stanley, Bryant & Miikkulaine, 2005), (Ponsen et. al., 2005) and (Ponsen et. al. 2006). Machine learning has been successfully employed in commercial titles as well, either by the game developers or by a

dedicated package (Funje et. al. 2007). FPS games have received attention as a machine learning test-bed due to their popularity and applicability as a model for real-life situations. Benjamin (Benjamin, 2002) studied the performance of different supervised learning techniques in modelling player behaviour in Soldier of Fortune™ FPS. He showed that neural networks (NNs) with a large dataset generally outperformed other supervised learning techniques (decision trees, k-nearest neighbour and Bayesian classification). Zanetti and ElRahlibi (Zanetti & El-Rhalibi, 2004) applied neural networks to learn basic actions of a Quake 3 Arena™ bot. They considered 3 behaviours: *Map Navigation, Movement in Fight* and *Aiming and Shooting.* For each behaviour, a neural network was trained. It is assumed that Movement in Fight and Map Navigation NNs would be trained for each map. The work of Zanetti and Rhalibi focused mainly on 1-on-1 death match games, where the player scores only by killing the enemy. They concluded that NNs need to be combined with other AI techniques to become effective.

(Bakkes & Spronck, 2005) used symbiotic learning to manage a team of four agents playing CTF in Quake 3 Arena™. They employed a joint action learning model. The learning algorithm checks the state of the flags and selects a joint role-assignment for the four agents. Each agent is assigned the role of attacker, defender or roamer.
(Smith, Lee-Urban, & Munoz-Avila, 2007) proposed a reinforcement learning algorithm, RETALIATE, which they used to manage a team of three agents playing *domination* game in Unreal Tournament™. Like any action-value based reinforcement learning algorithm, RETALIATE calculates a *reward* for each executed action and uses it to learn a *Q-function* $Q(s, a)$, usually represented by a *Q-table*, that indicates how good an action a is at state s[1]. RETALIATE also employs a joint action learning model; it evaluates the state of the whole team and selects a joint action for the three agents. This joint action is an assignment of each domination point to a certain agent.

We use reinforcement learning for adaptive team management. Unlike (Bakkes & Spronck, 2005) and (Smith, Lee-Urban, & Munoz-Avila, 2007), we employ an individual learning model, in which each agent independently learns an individual policy. Individual learning has the advantage that the state/action space, and hence the size of the Q-table, does not depend on the number of agents. This allows for adding more detailed actions without a massive increase in the Q-table size and consequently the learning problem.

[1] Formally, $Q(s, a)$ is defined as the expected life-long reward when the agent executes action a at state s and follows an optimum policy afterwards.

Problem Definition

In a CTF match, the players are divided into two teams. Each team has a flag at the team's base. A team scores if the players of that team get the enemy's flag to their base while keeping their flag from being taken by the enemy. Therefore CTF is a sophisticated game because it involves two often contradicting sub-goals: taking enemy's flag and defending the team's flag (and restoring it if taken). Players can kill each others using weapons. When a character dies, it re-spawns after a specified period of time.

AI Organization

In Cerberus, each agent is controlled by a multi-layer pluggable AI consisting of three layers, as shown in the following figure:

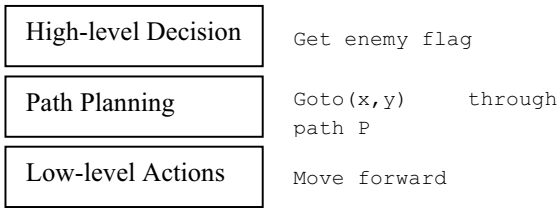

Figure 1 – AI Decision Layers

The top layer, controlled by reinforcement learning, provides high level decisions. The middle layer deals with path planning, and the lower layer handles low-level actions, mainly movement and shooting.

In our prototype we assume a fully observable environment and a planar play field (map) containing obstacles. We also assume that each player has a single limited range weapon of infinite ammo. These assumptions do not affect the reinforcement learning module because, as will be shown, it is not concerned with the fine-grained details of the game state. Also, the concepts applied in lower modules can still be applied in more complicated environments.

Reinforcement Learning Module

Our framework uses reinforcement learning to select a high-level action for each team member. As mentioned before, we use individual learning model in which each agent learns an individual policy that maps its individual state to an individual action in the hope that the agents together develop a cooperative behaviour that corroborates team performance.

The state of each agent is represented by a tuple of 5 binary values, namely:

- Is the agent's flag at base?
- Is the agent's flag dropped?
- Is the opponent's flag at base?
- Is the opponent's flag dropped?

- Is the agent carrying the enemy flag?

Thus the Q-table includes 32 states. Only 12 of them are valid states and have to be learnt. This makes the learning process easier. The agent can perform 8 high level actions, each with certain preconditions and termination conditions. These actions are *GET ENEMY FLAG*, *WAIT AT ENEMY BASE*, *KILL FLAG CARRIER*, *DEFEND THE BASE*, *RESTORE OUR FLAG*, *GET ENEMY FLAG FROM GROUND*, *SUPPORT FLAG CARRIER* and *RETURN TO BASE*.

We use an ε -greedy policy (Sutton & Barto, 1998) for action selection. Hence, the algorithm can be outlined as follows:

ALGORITHM Select_Action(State *s*, Events *E*)
If(Current Action met termination condition)
{ *a* = Current Action
p = State in which action a started
R ← CalculateReward(*a*, *p*, *s*, *E*)
$\delta \leftarrow (r + \gamma \max_{a'} Q(s,a') - Q(p,a))$
$Q(p,a) \leftarrow Q(p,a) + \alpha\,\delta$
Set_A ← GetPossibleActions(*s*)
Generate random number *x* such that 0<*x*<1
If($x < \varepsilon$)
{ new_a ← Select random action from Set_A}
Else
{ new_a ← $\arg\max_{a'} Q(s,a')$}
Return new_a
}
Else { Return NO DECISION }

There are two kinds of rewards, global and local. A global reward is received by all team members when the enemy flag is captured as this is the ultimate goal of the game. Similarly, a global punishment is received by all team members when the team's own flag is captured. Local rewards are individual intermediate rewards (or punishments) that are received by each team member at the end of its action. The local reward is determined according to the state and events upon action termination, as shown in table 1.

Action	**Agent's Condition**	**Reward**
GET ENEMY FLAG	Agent took enemy's flag	240
	Enemy flag is taken & #attackers < #team / 2	160
WAIT AT ENEMY BASE	Agent took enemy flag	80
	A teammate took enemy flag	120
	ELSE	-32
KILL FLAG CARRIER	Agent took enemy flag	80
	A teammate took enemy flag	40
	ELSE	-80
DEFEND	Agent's flag at base & #defenders < #team / 2	160
	ELSE	10
	Agent is flag carrier	-80
RESTORE FLAG	Agent Restored Flag	160
	A teammate restored flag	120
	ELSE	40
GET FLAG FROM GROUND	Agent took enemy's flag	80
	ELSE	160
SUPPORT FLAG CARRIER	Enemy's flag captured	80
	Above condition not satisfied	8
RETURN TO BASE	Agent captured the enemy's flag	320
	Agent is flag carrier	80
	ELSE	-120

Table 1: Rewarding conditions for each action. The reward values are the values used in the experiments. "ELSE" means that none of the other two conditions is satisfied.

Path Planning Module

In this section we describe path planning in Cerberus. The map exterior and obstacles are represented by polygons. We use the waypoint system approach (van Waveren, 2003) for map analysis and path planning. A waypoint system is a graph $G=(W,E)$ with each node in W representing a waypoint at a place where the agent can exist and each link (w_1, w_2) in E indicates a possible path for the agent to use (when going from w_1 to w_2). We use undirected links because, in our prototype, the agent can move in both directions of any link. Figure 2 illustrates a sample waypoint system.

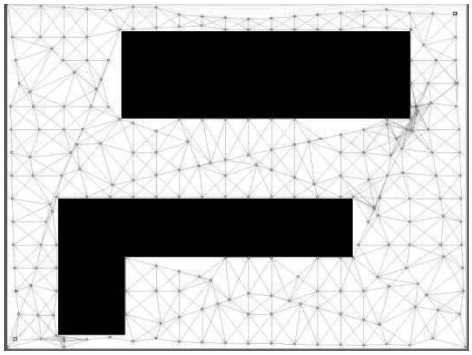

Figure 2 – A waypoint system generated by the map analysis module

Path planning module is responsible for offline map analysis as well as online path planning.

Map Analysis

The map analysis module accepts the polygonal representation of the map and automatically generates the waypoint system. We have developed an algorithm that guarantees a lower bound on waypoint density regardless of the number of polygons in the map. To achieve this, the algorithm divides the map into rectangular cells and guarantees that each reachable cell contains at least one waypoint. This allows for fine-grained path planning for both closed-door and open-door maps. The steps of our algorithm are detailed in (Hefny et. al. 2007).

Online Path Planning

The agent uses A* algorithm (Russel & Norvig, 2003) to plan a path through waypoints. To account for game dynamics, the path is replanned on regular intervals. The game dynamics are represented by influence maps. An *influence* can be applied to a waypoint. This influence is a cost that propagates to neighbouring waypoints and can be considered in path planning. In our implementation, we use character influence, which is an influence that is applied to the nearest waypoint to each character. The cost introduced by this influence can be used to plan paths that avoid enemies, avoid allies (to perform a cooperative attack through different paths) or tend to allies (for protection).

A* is CPU intensive and must be controlled. Therefore we do not allow agents to directly invoke A*. Instead, we have implemented a central module that accepts A* requests from agents, executes them in a controlled manner that avoids CPU consumption bursts and then returns the resulting paths to the agents. We assume that the interval between issuing an A* request and receiving the result is negligible that the agent can follow the old path during this interval. The same concept is applied in the calculation of influence maps.

Neural Networks Module

We use neural networks to control the agent behaviour during fight. These neural networks are trained using recorded samples from a human player. We follow (Zanetti & El-Rhalibi, 2004) in constructing a separate feedforward network for aiming and shooting and another one for movement. We do not employ a neural network for map navigation.

Each network contains one hidden layer. The number of nodes in the hidden layer is chosen as double the number of nodes in the input layer.

The neural networks are enabled or disabled according to an *activation condition*. Because we assume limited range weapons, we have used the distance to the nearest enemy as an activation condition. We use a hysteresis effect (double thresholding) to enable or disable the networks to avoid oscillatory behaviour (i.e. repeated enabling and disabling of the networks) when the distance to the enemy is close to weapon's range.

Tables 2 and 3 show the features used by each network and the output provided by each one. As shown, we introduce *temporal features* that relate the current time step to the previous one such as MyLocationDifference and EnemyRelativeMotion. MyLocationDifference is a feedback input that is intended to help the network learn a sequence of actions while EnemyRelativeMotion is intended to let the network take enemy motion into consideration.

It should also be noted that the movement network outputs are calculated assuming that the agent is looking at the enemy regardless of the actual direction of the agent. We believe that during fight, the human player is concerned, for example, with moving to or away from the enemy rather than moving forward or backward. During game play, the low-level actions module converts the network output to the correct stepping and strafing actions, depending on the actual direction of the agent.

RelativeEnemyPosition: The distance between the enemy and the agent, divided by the weapon range, for normalization. *DirectionToEnemy_1*: The absolute rotation angle required for the agent to face the enemy. *DirectionToEnemy_2*: A binary feature that decides if the enemy is standing to the right or left of the agent.
Output: Shoot, turn left and turn right
Table 2: Shooting Neural Network Features

DistanceToEnemy: Same as in Table 1. *DirectionToPlayer:* Enemy's rotation angle to face the agent. *MyLocationDifference:* Difference between the agent's last location and current location. *EnemyRelativeMotion:* Difference between the enemy's last location and current location with respect to the location and direction of the agent.
Output: Move forward/backward, Move left/right
Table 3: Fight Movement Neural Network Features

The neural networks are trained offline by recording a human teacher performing the desired behaviours and then sampling the state / action pairs. Only the samples that satisfy the neural network activation condition are used for training.

When the agent is engaged in a fight while navigating to a goal, we do not let the movement network control it completely. Instead, the agent moves in the direction determined by the linear combination:

$$V = V_p + \alpha V_m$$

Where V_p is the path following motion under no fighting condition and V_m is the motion determined by the neural network. This balances between moving towards the goal and dodging enemy attacks. In our implementation, we set α to 0.3.

Experimental Results

Reinforcement Learning

In this section we provide an empirical evaluation of the RL module. We ran a test of 60 games of 5-minute duration each. In each game, a team of three adaptive agents played against a static team of two offensive agents and one defensive agent. An offensive agent always tries to get the enemy's flag or supports the flag carrier, whichever applicable. The defensive agent always defends the team's flag and tries to restore it whenever it is taken. Thus, the static team represents a moderate policy which is a typical AI policy in CTF games. We set the learning rate (alpha) to 0.1 and the discount factor (gamma) to 0.9995. Epsilon value was initialized to 1.00 and was decreased to 0.80, 0.60, 0.50, 0.40, 0.30, 0.20, 0.10, 0.08, 0.06, after 2, 2, 2, 2, 2, 4, 4, 4, 4, 4 games respectively.

We set the global reward to 1600 and the global punishment to -400. Local rewards are set according to table 1.

The experiment was run on 3 phases:
Phase 1: Running a static team vs. another static team to test the bias of the map.
Phase 2: Running an adaptive team vs. a static team. The adaptive team starts with a defensive strategy, which is a very poor strategy in a fast-paced game like CTF.
Phase 3: Making use of the policy developed by the adaptive team against a static team. In this experiment the adaptive team executes only the actions with the highest Q-value.

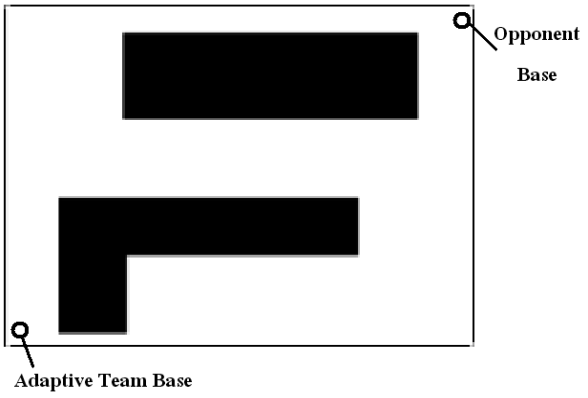

Figure 3 – Test Map

As shown from the experiment results, the map is biased to the black team (Figure 4). The total score differences in the 3 phases making the black curve as our reference were -88, -19, and +30 respectively. This shows clearly that the adaptive team could modify its poor strategy to overcome the static team despite the map bias.

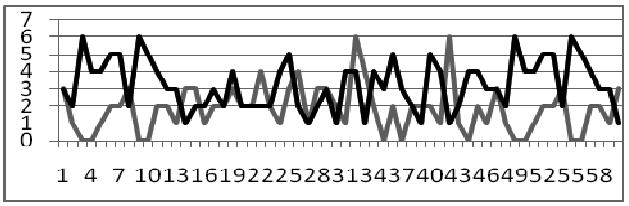

Figure 4 - Static Teams vs. each other

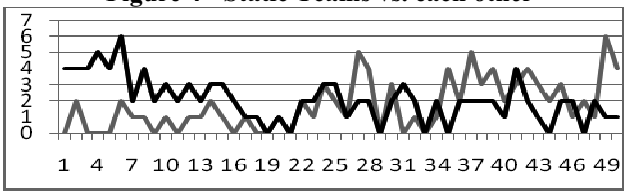

Figure 5 - Adaptive team (grey) vs. Static (black)

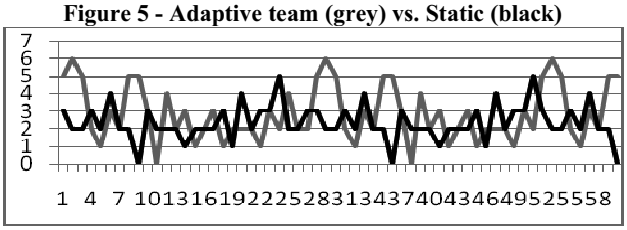

Figure 6 - Final Strategy (grey) vs. Static (black)

We played against the adaptive team to observe the strategy it developed. We found that the team tends to be aggressive, focusing on taking the enemy flag. Even if the team's flag is taken, focusing on taking the enemy's flag is preferred to focusing on killing the flag carrier. It is also frequent that one of the agents takes the action of waiting at enemy's base when its team takes the enemy's flag. By waiting at the enemy's base, the agent can retake the enemy's flag once it is returned to the base to achieve an easy capture.

Neural Networks

Because we are concerned with the general behaviour rather than the coincidence of NN output with sample outputs, we provide subjective comments on our observation of the performance of the NNs. The shooting NN was trained to turn towards the enemy and start shooting as long as the enemy is in its angular range. It could produce the required behaviour.

The movement NN, as shown in figure 7, could imitate regular behaviours such as circle strafing (i.e. moving around the enemy) to a very high degree of accuracy. Irregular and oscillatory behaviours such as linear strafing (i.e. strafing left and right) were not imitated at the same degree of accuracy, but the resulting agent is still adequate.

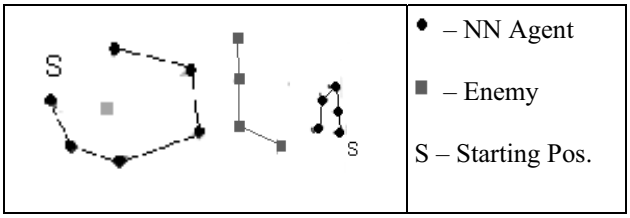

Figure 7 – NN agent motion trajectories for circle strafing (left) and linear strafing (right)

Conclusions and Future Work

In this work we have demonstrated a framework that applies machine learning techniques in capture the flag game as an example of a team-based real time game. We have used neural networks as a supervised learning module that learns offline and reinforcement learning as a module that learns online.

We have shown that neural networks can learn to imitate human behaviour and provide an adequate computer character control. Because we used simple maps, the trained NNs performed well in new maps. A future enhancement possibility is to develop a method that encodes geographical context in a general manner to enable NNs to behave well in complicated maps without the need of training for each map.

We have shown that individual reinforcement learning can be used to develop team behaviour that adapts to the enemy's team strategy. Specifically, individual learning allows for cooperative individual actions rather than a high level team formation, making it possible for an RL agent to play with a static or even a human player in the same team.

Another problem that needs investigation for the applicability of learning is the enemy arbitration problem (i.e. selecting which enemy to fight when there are multiple enemies and supporting allies in range).

We are also interested in scaling up this system to more complicated CTF environments including complicated, fully 3D playfields, multiple weapons, pickup items ... etc.

References

Ahmed A., "Reinforcement Learning for Multi-agent Teamwork", M.S. Thesis, Cairo University, Egypt, 2002.

Alpaydin, E. "Introduction to Machine Learning", Prentice Hall of India, 2005.

Benjamin, G. "An Empirical Study of Machine Learning Algorithms Applied to Modelling Player Behaviour in a First Person Shooter Video Game", University of Wisconsin – Madison, USA, 2002.

Bakkes, S. and Spronck, P. "Symbiotic Learning in Commercial Computer Games", 7th International Conference on Computer Games, 2005.

Funge, J., Musick, R., Dobson, D., Duffy, N., McNally, M., Tu, X., Wright, I., Yen. W. and Cabral, B. "Real Time Context Learning by Software Agents". US Patent 7296007.

Hefny, A., Hatem, A., Shalaby, M., Khalifa, Y. and Atiya, A. "Cerberus: The First Person Shooter Game with Machine Learning", Technical Report, Cairo University, Faculty of Engineering, Computer Engineering Department.

Ponsen, M., Muñoz-Avila, H., Spronck P., and Aha, D. "Automatically Generating Game Tactics with Evolutionary Learning", AI Magazine, Vol. 27, No. 3, pp.75-84, 2005.

Ponsen, M., Spronck, P., and Tuyls, K. "Hierarchical Reinforcement Learning with Deictic Representation in a Computer Game", Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence, 2006.

Russel, S. and Norvig, P., "Artificial Intelligence, A Modern Approach", Prentice Hall, 2003.

Smith, M., Lee-Urban, S. and Munoz-Avila H. "RETALIATE: Learning Winning Policies in First-Person Shooter Games", Proceedings of the 17th Innovative Applications of Artificial Intelligence, 2007.

Stanley, K., Bryant, B. and Miikkulaine, R. "Evolving Neural Network Agents in the NERO Video Game", Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games, 2005.

Sutton, S. and Barto, G. "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA, 1998.

van Waveren, J. "The Quake III Arena Bot", University of Technology Delft, Faculty ITS, Holland, 2003.

Zanetti, S. and El Rhalibi, A. "Machine Learning Techniques for FPS in Q3", Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, 2004.