Creating a Multi-Purpose First Person Shooter Bot with Reinforcement Learning

Michelle McPartland and Marcus Gallagher, *Member, IEEE*

***Abstract*—Reinforcement learning is well suited to first person shooter bot artificial intelligence as it has the potential to create diverse behaviors without the need to implicitly code them. This paper compares three different reinforcement learning approaches to create a bot with a universal behavior set. Results show that using a hierarchical or rule based approach, combined with reinforcement learning, is a promising solution to creating first person shooter bots that offer a rich and diverse behavior set.**

I. Introduction

Bot artificial intelligence (AI) in first person shooter (FPS) games generally comprise of path planning, picking up items, using items, and combat. Traditionally, hard-coded methods such as state machines, rule based systems, and scripting are used for bot AI in commercial games. The problem with these methods is that they are static, can be hard to expand, and time is needed to hand tune parameters. This paper investigates several methods based on reinforcement learning (RL) to create a multi-purpose bot AI with the behaviors of navigation, item collection and combat.

There are many advantages of using RL for FPS bot AI. Minimal code is required for a range of behaviors as shown in previous work [1]. RL can be used to find bugs in the code, for example the RL bots might learn an easy exploit in killing the enemies which will be prevalent when replaying games. As with the traditional bot AI methods, there are a number of parameters associated with RL algorithms which take time to tune. However, the tuning time can be minimized by batching experiments, and then further investigating the promising results. One of the main disadvantages of RL is the problem of scaling with increased complexity. In this paper we investigate two different approaches to this problem by splitting the problem space into smaller tasks, then finding ways to combine them to produce the overall behavior set. Another problem with RL is becoming trapped in a cycle of repetitious states. This problem can be caused by states not being visited enough in the training phase, or the reward function not being set up appropriately for the problem.

This paper expands on previous work where low-level FPS bot controllers were trained using RL [1]. The aim of this paper is to use the previously trained controllers and combine them to produce bots with a multi-purpose behavior set. Three different types of RL will be compared to investigate the differences in statistics and behaviors produced. The first method will combine a previously trained combat and navigation controller using hierarchical RL. The second method will use simple rules to determine when to use the combat or navigation controller. The third method will use RL to relearn the tasks of combat and navigation simultaneously. These three methods will be compared to a random controller and a state machine controlled bot.

This paper is organized as follows. First an overview of RL and hierarchical RL will be given. Section 2 will describe the game test environment, and each of the different bot AI setups used in the experiments. Section 3 presents the statistical results of the experiments, and a discussion of these results. Section 4 will display the navigation paths of the bots from the experiments along with observation of the bot's behaviors. The last section concludes the paper with possible future work.

II. Background

Reinforcement learning is a popular machine learning technique most commonly used in multi-agent systems [2], [3] and robotics [4]. The RL algorithm allows an agent to learn a task through interaction with an environment. The problem is divided into a set of states and actions the agent can perceive and act on. The agent then learns what action is best to take in a given state based on the reward it receives from the environment. The mapping of state-action values is called the policy. A popular approach to storing the policy values is the tabular approach. The tabular approach uses a lookup table to store the values of each state-action pair, and these values are altered as the agent interacts and learns in the environment. Another popular approach is using an algorithm to generalize the state to action mapping. A common generalization algorithm to use is artificial neural networks. One of the main problems with this approach is finding a suitable topology for the problem space. Recent work attempts to address this problem using evolving topologies [5].

In this paper we use a tabular Sarsa algorithm due to its success in similar work [6], [7]. The problem of increased complexity will be investigated by breaking the problem into smaller tasks, and comparing to a flat structured RL

M. McPartland is with the University of Queensland, St Lucia, Australia (61-2-62470267; fax: 61-7-33654999; e-mail: michelle@itee.uq.edu.au).

M. Gallagher is with University of Queensland, St Lucia, Australia (e-mail: marcusg@itee.uq.edu.au).

978-1-4244-2974-5/08/$25.00 ©2008 IEEE

setup. The Sarsa algorithm with eligibility traces [8], termed Sarsa(λ), is used for updating the policy values. Eligibility traces are a method to speed up learning by allowing past actions to benefit from the current reward, and also allows sequences of actions to be learnt. Sequences of actions are particularly important in combat behavior where the bot AI should be able to perform a combination of actions in order to defeat an enemy. Table I lists the steps of the Sarsa(λ) algorithm. $Q(s,a)$ is the policy consisting of all state-action pairs. $e(s,a)$ is the eligibility trace variable for a given state-action pair. Line 7 calculates the temporal difference (TD) error using the current reward, decay parameter (γ) and current and next state-action pair values. Line 8 sets the eligibility trace (λ) to one, to flag that state-action pair as just being visited. The update function is listed in line 10. The previously calculated TD error is used with the learning rate (α) and eligibility trace (λ) to update each state-action pair in the policy. Line 11 reduces the eligibility trace of previously visited state-action pairs. This reduction affects state-action pairs that have just been visited by allowing them to receive more of the current reward than states further in the past. The reduction is determined by the decay (γ) and trace (λ) parameters.

TABLE I
PSEUDO CODE FOR THE SARSA(λ) ALGORITHM

1:	Initialize $Q(s,a)=0$, set $e(s,a)=0$ for all s, a
2:	Repeat for each training game
3:	Repeat for each update step t in the game
4:	Set s' to the current state
5:	Select an action a'
6:	Take action a', observe reward r
7:	$\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
8:	$e(s,a) \leftarrow 1$
9:	For all s, a:
10:	$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
11:	$e(s,a) \leftarrow \gamma\lambda e(s,a)$
12:	$s \leftarrow s', a \leftarrow a'$
13:	Until end game condition is true

Hierarchical RL is widely used in agent based systems [9], [10] and robotics [11] to help handle the complexity in these domains. Hierarchical RL is the process of splitting the problem space up into smaller tasks, and then using a hierarchy to decide what task to perform in different situations.

A small amount of research has been performed in RL in games, such as Manslow's [12] application of RL to car racing AI and Merrick's [7] work investigating RL in role-play games. To our knowledge only Vasta *et al* [13] has investigated RL in FPS games, and this work looked at high-level coordination of team based bots, similar to multi-agent system work. This paper expands on previous work where two low-level controllers were trained using a tabular Sarsa(λ) RL algorithm. The most successful navigation and item collection controller, and a combat controller will be combined using methods based on RL. The aim of the paper is to investigate the effects they have on a bot's game statistics and behaviors.

III. METHOD

The test environment used for the experiments was a simple, purpose-built FPS game. The game test-bed has the basic features of commercial FPS games such as walls, power up items and bots. Bots are equipped with the ability to turn left, turn right, move forwards, move backwards, strafe left, strafe right, pick up items and shoot. All bots in the game have exactly the same capabilities and parameters, e.g., turn speed (0.1), speed (0.2 meters/update cycle), ammo clip (unlimited), weapon type (laser ray gun with five hit points per shot damage rating), and hit points (50). The laser guns have a cool down timer of one second, to avoid a constant stream of fire. Bots respawn ten update cycles after being killed, and they spawn in the same location each time with full health. Items respawn ten update cycles after being collected.

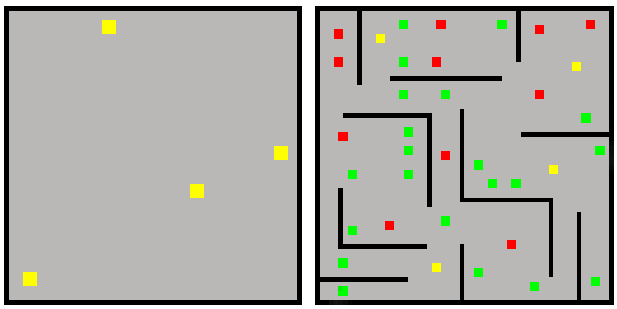

Fig. 1. (a) This figure shows the layout of the arena map to the left, and (b) the maze map to the right. Yellow represents the bots spawn positions, red and green represent item spawn points.

Two different level maps were designed for the experiments to test the AI in varying environments. The first map is the arena (Fig. 1a), which consists of four enclosing walls and no items to collect. The arena map is 100m x 100m. The second map is a maze environment (Fig. 1b), and was used to investigate how well the bots performed in a more complex environment where combat occurs in tight spaces, and navigating the environment is more integral to the bots success. The maze map is also 100m x 100m.

Each experiment included one RL type bot, one state machine controlled bot (termed StateMachine) and two non-reacting navigation bots. The StateMachine bot was programmed to navigate the environment, pick up items in its near vicinity, and follow and kill an enemy in sight. The StateMachine bot was coded with an aggressive type behavior as it would not back down from a fight, and would

pursue the enemy to the end. The two non-reacting navigation bots were included to give the RL bots practice targets to learn combat. These bots navigated the environment at all times. Two different enemy types were also included to compel the RL bot to learn a generalized combat strategy.

TABLE II
REWARD FUNCTION FOR THE HIERARCHICALRL TASK

Reward	Value
Item collected	1.0
Enemy kill	1.0
Accurate shot	1.0
Distance travelled (0.4m/step)	-0. 0000002

Three different RL setups were used in the experiments: HierarchicalRL, RuleBasedRL and RL. The HierarchicalRL setup uses hierarchical RL to combine a combat and navigation controller that were trained in previous work using RL [1]. The HierarchicalRL controller learns when to use the combat or navigation controller by receiving the rewards from Table II. Collision and death penalties were not included as preliminary runs showed better performance without these penalties. The better performance was due to the conflicting nature of the problem to maximize kills and exploration, while minimizing deaths and collisions. For this problem, maximizing kills and exploration inadvertently minimized deaths and collisions, therefore the penalties were not needed. A range of parameters were experimented with, and results were varying. In general, medium discount factors and eligibility traces were the best performing.

The state and action spaces for the HierarchicalRL controller were the combined set of the combat and navigation controller spaces as follows.

$$S = \left\{ \begin{array}{l} LObj_1, FObj_2, RObj_3, LEnemy_4, \\ FEnemy_5, REnemy_6, LItem_7, \\ FItem_8, RItem_9 \end{array} \right\}, s_i \in \{0,1,2\}$$

$$A = \{NavCntr_1, ComCntr_2\}$$

Where prefixes L, F and R stand for left, front and right respectively, and correspond to probe sensors relative to the bot's position and orientation. The number of state-action pairs in the policy table was 39366.

The RuleBasedRL controller uses the same previously trained navigation and combat controllers as the HierarchicalRL controller. The RuleBasedRL controller uses rules to determine when to use the combat or navigation controller. When an enemy is in sight the combat controller is used, otherwise the navigation controller is used.

The RL controller learns the entire task of combat and navigation from scratch, using a flat RL structure. The state space is identical to the HierarchicalRL state space previously listed. The action space consists of all possible actions the bot can take as follows.

$$A = \left\{ \begin{array}{l} Forward_1, TurnLeft_2, TurnRight_3, Backward_4, \\ StrafeLeft_5, StrafeRight_6, Shoot_7 \end{array} \right\}$$

The number of state-action pairs for the RL task is 137781, more than three times larger than the HierarchicalRL table.

TABLE III
REWARD FUNCTION FOR THE RL TASK

Reward	Value
Item collected	1.0
Enemy kill	1.0
Accurate shot	1.0
Collision	-0. 000002

Table III lists the rewards and corresponding values for the RL bot. A reward was given to the RL bot when it picked up an item, killed an enemy bot, and when it accurately shot an enemy. A small penalty was given when the RL bot collided with an obstacle. The death penalty and distance reward were removed to attempt to simplify the problem.

All three RL bots were trained for 5000 iterations in both the arena and maze environments, where an iteration was a single step of the game. The learnt policies were replayed for 5000 iterations, with learning disabled (α = 0), twenty times with random seeds ranging from 101 to 120. The three RL bots will be compared against the StateMachine bot from the RuleBasedRL experiments, as well as against a random bot (termed Random). The Random bot randomly selects actions from the previously defined action set in the RL controller setup.

IV. RESULTS

Initial experiments were performed with a wide range of parameters to find the best solutions. The RL parameters used for the HierarchicalRL controller were α = 0.2, ε = 0.2, γ = 0.35, and λ = 0.65, and the RL controller were α = 0.2, ε = 0.1, γ = 0.9, and λ = 0.2. The best scoring policy was then replayed twenty times with different random seeds and the average, minimum and maximum results for each bot were recorded. The recorded results were number of deaths, number of enemy kills, shooting accuracy, number of collisions with world geometry, distance travelled, and number of items collected (maze map only).

Table IV displays the results from the arena map experiment. The average number of deaths is similar for all five bots, while the number of kills varied greatly. The greatest variance in deaths is seen in the RL bot with a maximum of 13 deaths and minimum of 0 in one trial. The variance is due to the instability of the learnt policy, where the bot was more likely to ignore enemies and slide through the environment. The similarity of performance compared to the Random bot also demonstrates the variability of the RL bot. The HierarchicalRL and RuleBasedRL bots were more stable in deaths, showing similar results to the StateMachine.

The StateMachine bot clearly showed its skill in combat

TABLE IV
STATISTICAL RESULTS FOR ARENA MAP EXPERIMENT

Bot	Value	Deaths	Shots	Hits	Accuracy (%)	Kills	Collisions	Distance (m)
HierarchicalRL	Average	3.6	70.2	56.9	81.0	5.0	520.6	112.3
	Minimum	1.0	37.0	25.0	61.0	2.0	8.0	109.7
	Maximum	5.0	108.0	87.0	91.9	9.0	1430.0	114.0
RuleBasedRL	Average	3.6	110.1	84.8	76.4	7.4	27.7	111.3
	Minimum	2.0	68.0	40.0	58.8	3.0	0.0	101.9
	Maximum	5.0	154.0	137.0	89.0	12.0	510.0	114.9
RL	Average	4.2	155.0	27.0	21.4	1.9	2147.8	91.6
	Minimum	0.0	56.0	2.0	0.7	0.0	59.0	25.3
	Maximum	13.0	299.0	54.0	38.6	6.0	4604.0	144.1
Random	Average	3.9	292.0	4.5	1.5	0.1	127.0	142.5
	Minimum	0.0	281.0	1.0	0.3	0.0	0.0	140.3
	Maximum	9.0	306.0	12.0	4.0	1.0	244.0	146.6
StateMachine	Average	2.8	272.1	136.9	50.4	14.1	26.0	143.9
	Minimum	2.0	221.0	96.0	37.4	10.0	0.0	130.2
	Maximum	5.0	345.0	189.0	60.2	19.0	131.0	163.6

with a very high score for kills in the arena map. The RuleBasedRL bot followed with half as many average kills, but with a high score of 12 kills in one run. The HierarchicalRL bot was close behind the RuleBasedRL bot, with an average of 5 kills and high score of 9 kills. While the StateMachine bot recorded a high number of kills, we see that the kills were mainly from the two non-reacting bots. Since the number of deaths for the StateMachine bot was similar to the RuleBasedRL bot, we know that the RuleBasedRL bot killed the StateMachine bot as many times as it was killed. Similar results were seen against the HierarchicalRL bot. The similar death count therefore implies that the RuleBasedRL and HierarchicalRL bot were formidable opponents to the StateMachine bot when head-to-head, but the underlying combat controller was less effective against the non-reactive navigation bots. The RL bot managed a maximum of 6 kills, indicating that some form of combat behavior was learnt. This result was surprising considering the high number of deaths similar to the Random bot.

While the StateMachine outperformed the other bots in kills, the HierarchicalRL and RuleBasedRL bots significantly outperformed the StateMachine bot in shooting accuracy. The HierarchicalRL controller learnt the highest accuracy with an average of 81% and maximum of 91.9%, which was greater than the RuleBasedRL bot's average accuracy of 76.4%, and maximum of 89%. The HierarchicalRL controller had a higher shooting accuracy as it learnt not to use the combat controller when the bot would shoot and miss. This situation occurred when there were enemy bots far and to the right of the bot's current position. However, because the HierarchicalRL bot did not initiate combat in this state, the enemy would go out of sight range or have the advantage in following the fleeing bot, which is the main reason the HierarchicalRL bot killed less enemies than the RuleBasedRL. This behavior can be likened to an expert FPS player's type of strategy, where only near guaranteed shots are taken; otherwise a safe position is sought. The StateMachine bot had a much lower accuracy than the HierarchicalRL and RuleBasedRL bots. Although the StateMachine's accuracy was quite low, the bot shot many times more than the higher accuracy bots, therefore having the advantage in combat kills. Due to the large differences in shooting accuracies, it is expected that the HierarchicalRL and RuleBasedRL bots would perform better than the StateMachine bot in scenarios where ammo was limited which is common in most commercial FPS games. The RL bot had a very low accuracy rating. The RL bot learnt to shoot often, sometimes even shooting when no enemy was in sight. However, the RL bot's accuracy is still higher than the Random bot. This difference shows that the RL bot was able to learn a combat strategy which consisted of shooting as often as possible, generally when enemies were in sight, in the hope of taking down the enemy. This type of combat behavior is similar to beginner FPS players as they tend to constantly shoot haphazardly in times of danger.

The StateMachine and RuleBasedRL bots have similar very low average collision counts around 26 and 27.7 respectively. The HierarchicalRL bot has a much higher average count of 520.6. The RL bot had the highest number of collisions in this experiment, with an average of 2147.8, and maximum of 4604.

The StateMachine and Random bots travelled similar distances during the experiments. This result was expected from the similar number of collisions, as number of collisions is correlated with distance travelled. The exception to this rule is when the bots slide across a wall, which will increase the collision count, but the bot will still move a distance (although less than a full step forward). The RL bot was the only trial where distance was not consistent, again showing the instability of this controller. The HierarchicalRL and RuleBasedRL controllers have consistent distances in average, minimum and maximum indicating that the recorded collisions were sliding wall collisions as opposed to head on collisions where the bots

TABLE V
STATISTICAL RESULTS FOR MAZE MAP EXPERIMENT

Bot	Value	Deaths	Shots	Hits	Accuracy (%)	Kills	Collisions	Distance (m)
HierarchicalRL	Average	3.1	59.5	50.6	84.4	3.9	419.9	91.6
	Minimum	0.0	16.0	12.0	65.8	1.0	115.0	58.3
	Maximum	6.0	103.0	80.0	94.7	7.0	675.0	106.9
RuleBasedRL	Average	2.3	71.2	56.0	81.4	4.4	74.0	107.0
	Minimum	0.0	9.0	7.0	43.6	0.0	13.0	100.8
	Maximum	5.0	165.0	90.0	97.0	8.0	131.0	109.0
RL	Average	3.0	46.2	17.8	38.7	1.1	2011.6	110.3
	Minimum	0.0	6.0	0.0	0.0	0.0	125.0	20.3
	Maximum	7.0	81.0	38.0	76.9	3.0	4465.0	156.4
Random	Average	1.3	292.9	4.1	1.4	0.1	348.8	139.5
	Minimum	0.0	283.0	1.0	0.3	0.0	81.0	130.5
	Maximum	8.0	303.0	10.0	3.5	1.0	971.0	144.0
StateMachine	Average	1.8	187.6	102.4	54.9	9.8	763.2	121.5
	Minimum	0.0	121.0	57.0	40.9	5.0	218.0	82.2
	Maximum	5.0	289.0	151.0	67.9	15.0	2765.0	149.6

were stuck. The type of collisions was confirmed through observation of the replays.

The maze map environment was significantly more complex than the arena map, and therefore less contact between bots occurred than previously. The RL parameters used for the HierarchicalRL controller were α = 0.2, ε = 0.1, γ = 0.5, and λ = 0.2, and the RL controller were α = 0.2, ε = 0.1, γ = 0.4 and λ = 0.3. Table V displays the results for the maze map experiment with the additional statistic of item collection.

The average number of deaths is similar across all bots in the maze map experiment. The HierarchicalRL bot had the highest average deaths, very closely followed by the RL bot. The RuleBasedRL bot scored similarly to the StateMachine, while the Random bot had the least number of average deaths. Overall, the death statistics were similar for all five bots, indicating that interaction occurred evenly between the bots during the replay games.

A good performance is seen from the StateMachine bot in enemy kills, with an average of 9.8 kills per trial, and maximum of 15. The HierarchicalRL and RuleBasedRL bots performed similarly in average kills. The RuleBasedRL bot also scored slightly higher in maximum kills (8) than the HierarchicalRL bot (7). The gap between the RuleBasedRL and HierarchicalRL bot in average kills was greatly decreased in the maze map from the arena map. The RL bot was unable to learn a good combat strategy in the maze map, with an average of 1.1 kills per trial, and maximum of 3 kills. As in the arena map, both the kill and death scores of the RL bot are similar to that of the Random bot, indicating the RL bot had difficulty learning a good combat strategy.

Similar to the arena map, the HierarchicalRL and RuleBasedRL bots significantly outperformed the StateMachine bot in shooting accuracy. The HierarchicalRL bot had an average accuracy of 84.4%, and maximum of 94.7%. The RuleBasedRL bot performed slightly worse than the HierarchicalRL, with an average of 81.4%, but higher maximum of 97%. As expected the StateMachine bot had a similar accuracy as in the arena map. Again the StateMachine bot shot many more times than the HierarchicalRL and RuleBasedRL bot, explaining the higher kill rate. The Random bot shot even more times than the StateMachine, but achieved a maximum accuracy of only 3.5%. The RL bot achieved an average of 38.7%, and maximum accuracy of 76.9%, showing that it was able to learn some part of combat as these scores were significantly higher than the Random bot.

The collision count in the maze map has a different trend to the arena map. The results show the RuleBasedRL bot had the lowest collision count overall. The HierarchicalRL bot had significantly more collisions than the RuleBasedRL bot, while the StateMachine bot performed poorly in collisions in the maze map compared to the arena map. The worst performing bot in collisions was the RL bot, with an extremely high average, although the minimum score was quite low in comparison.

Despite the high number of collisions, the StateMachine bot managed to travel a good distance through the maze with an average of 121.5m, and high of 149.6m. The HierarchicalRL and RuleBasedRL bots performed similarly with averages of 91.6m and 107m respectively. However, the RL bot travelled the greatest distance out of all the bots in a single trial with maximum of 156.4m. The highest scoring in average distance travelled was the Random bot (139.5m), as random actions allowed the bot to move around in a small space, frequently choosing to move backwards and forwards.

Although the RL controller had difficulty learning combat in the maze area, it did surprisingly well in the item collection task. The RL bot performed the best in average and maximum items collected. The RL bot's success in item collection, distance travelled, and to a lesser extent accuracy, indicates the necessity to split different tasks of the AI into sub-controllers. The RuleBasedRL also performed well in the item collection task. The HierarchicalRL bot performed

very badly in the item collection task, which was surprising considering the good result of the RuleBasedRL bot. Even the Random bot performed slightly better than the HierarchicalRL bot. The StateMachine was able to collect an average of 7.2 items per run, with maximum of 11. Considering the rules forcing the StateMachine bot to collect items nearby, the RL bot was able to learn a very good item collection strategy.

V. Observational Results

This section examines the paths travelled by the bots during the experiments. The paths are displayed from the best scoring bots from the maximum row in Table IV and V. The number of runs displayed in the path figures varies depending how many times the bot died during the 5000 replay iterations.

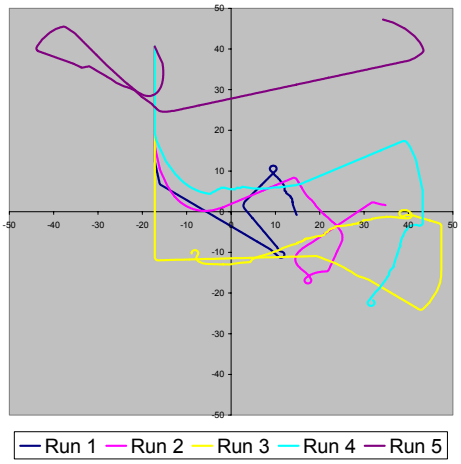

Fig. 2. HierarchicalRL bot paths from replay of the arena map experiment with random seed 102.

Fig. 2 displays the route taken by the HierarchicalRL bot. The HierarchicalRL bot's first run mostly used the navigation controller, even when multiple enemies were in view. The combat controller was occasionally chosen, and a few shots were fired, however the bot ended up navigating away from the fight with the enemy chasing and killing the HierarchicalRL bot quickly. The next run the HierarchicalRL bot took down three enemies in quick succession before being killed by the StateMachine enemy. The HierarchicalRL bot killed an enemy soon into the third run, then proceeded to navigate the walls until an enemy came into sight. The HierarchicalRL bot tracked the enemy from behind, but the enemy reacted and killed the HierarchicalRL bot first. The last two runs were similar, with the HierarchicalRL bot killing a few enemies, and navigating when no bots were in sight.

Although the RuleBasedRL bot used the same combat and navigation controllers as the HierarchicalRL bot, a different type of behavior was observed (see Fig. 3). While the HierarchicalRL bot tended to shy away from multiple enemies, the RuleBasedRL bot was very aggressive and would fight any in sight. Although it was expected that the RuleBasedRL bot be an aggressive fighter due to the controlling rules, it is interesting to note that the same underlying controllers can be used to produce varying behaviors by using different high level controllers. The RuleBasedRL bot had three runs in total, the first being the

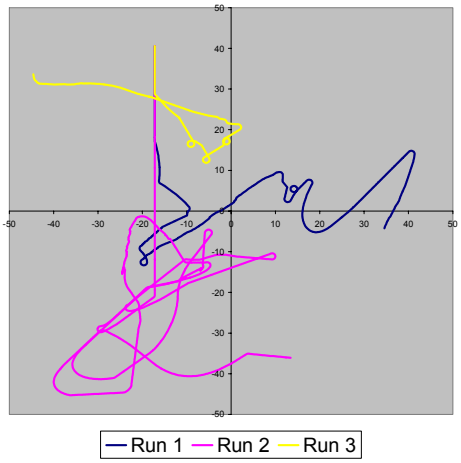

Fig. 3. RuleBasedRL bot paths from the replay of the arena map experiment with random seed 120.

shortest lived, with only a few kills before dying. In the second run, the RuleBasedRL bot went on a killing spree with a total of eight kills. During this run the RuleBasedRL bot tracked and killed the enemy while it was distracted shooting a non-responsive bot. In the third run the bot killed a few enemies before the iterations completed.

The RL bot displayed very different behavior from the previous two, although some similarities were seen during combat (see Fig. 4). The RL bot began by moving diagonally through the arena before encountering multiple enemies. The RL bot headed towards one of the enemies while shooting, and then did a loop and began travelling backwards. The bot continued on the backwards path until colliding with the eastern wall, where it became stuck and was shot down by the enemy. The second run the RL bot navigated the northern wall by strafing left, until

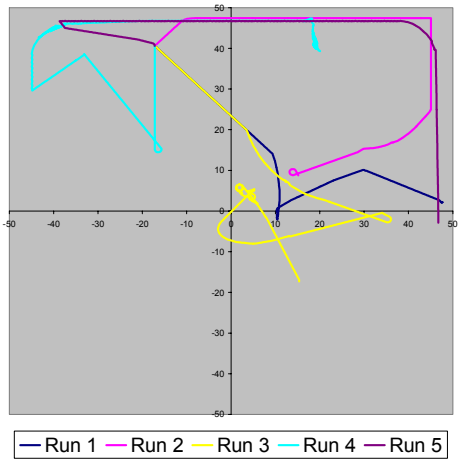

Fig. 4. RL bot paths from the replay of the arena map experiment with random seed 101.

encountering the corner where it turned and started moving forward along the eastern wall. An enemy came in sight a quarter of the way down the eastern wall, and the RL bot reacted by following and shooting. The RL bot was then killed by the enemy. In the third run the RL bot killed the StateMachine bot and a non-reacting bot, and took down a

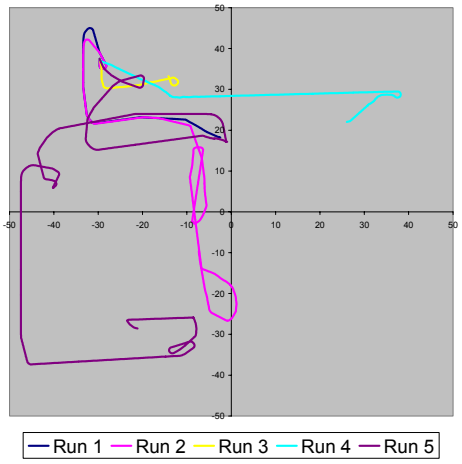

Fig 5. HierarchicalRL bot paths from replay of the maze map experiment with random seed 112.

third bot to 10% health before re-encountering the StateMachine bot and running away. At one point in the forth run the RL bot had trouble navigating the walls, deciding to shoot the wall and travel back and forth before continuing on. During the last run, the RL bot killed the enemy and then navigated the northern and eastern walls until the iterations were completed.

The HierarchicalRL bot in the maze map experiments started off using a combination of the navigation and combat controllers to navigate (see Fig. 5). The bot stayed close to the walls, sometimes sliding along, which caused a high collision count even though the bot did not become immovable on the walls. The first enemy encountered was the StateMachine bot, which proceeded in a close head-to-

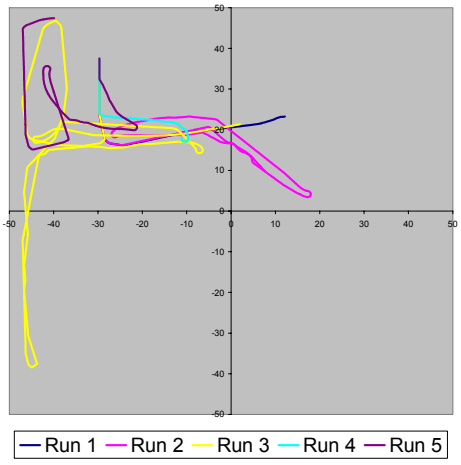

Fig. 6. RuleBasedRL bot paths from the replay of the maze map experiment with random seed 120.

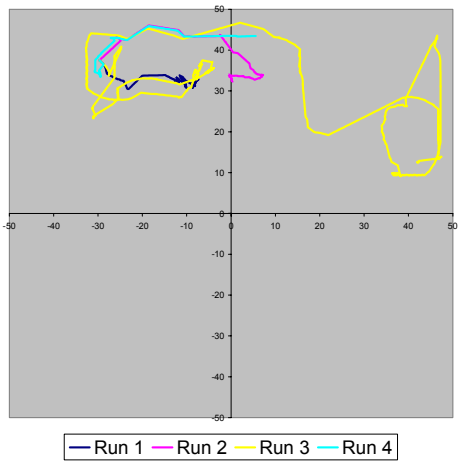

Fig. 7. RL bot paths from the replay of the maze map experiment with random seed 101.

head fight, but the StateMachine bot won with five hit points to spare. In the next three runs, the HierarchicalRL bot mainly navigated the area, killed the StateMachine bot once per run, and then was later killed. In the last run the HierarchicalRL bot killed the StateMachine bot, and two non-responsive bots. Between fights the HRL bot navigated the environment. At one point at the end of the run, the HierarchicalRL bot was heading towards an item using the navigation controller, but when it came close to the item, the combat controller was used which caused the bot to turn away.

The RuleBasedRL bot started off similarly to the HierarchicalRL, with a short run and close but unsuccessful head-to-head fight with the StateMachine bot (see Fig. 6). The second run was more successful, with three enemy kills in a short time, one of which was the StateMachine bot. The third run was the longest run of all five, with the RuleBasedRL bot navigating a fair portion of the maze, picking up items, and killing three enemies. In contrast, the forth run did not last long, with the StateMachine killing the RuleBasedRL bot soon after it respawned. During the last run the RuleBasedRL bot killed the StateMachine and a non-responsive bot, before navigating until the iterations were completed.

Fig. 7 displays the routes the RL bot took in the maze map experiment. As seen from the statistical data, the RL bot performed well in navigation and item collection in the maze environment. The RL bot navigated well in the first run, but showed the lack of combat skills when the enemy StateMachine bot came into view. The RL bot shot at the wall instead of the enemy until it was inevitably killed. The next run the RL bot had more success in combat, taking the StateMachine bot down to 20% health before being killed. During all four runs the RL bot showed an interesting navigation and item collection strategy. The bot generally favored strafing left and right to navigate close walls, and also used this zigzagging movement to collect items in the area. During the third run the RL bot successfully killed the StateMachine bot twice, showing that in certain state setups

it was able to learn a very good combat strategy. Both fights were head-to-head and the RL bot targeted and headed straight towards the enemy shooting accurately. The RL bot killed a non-responsive bot at the beginning of the forth run, travelling backwards while shooting at the enemy.

A distinct advantage of using RL over state machines and rule based systems is evident in the results. The state machine bot behaves the same under similar conditions; while the RL bots are also deterministic they are able to react differently in similar conditions. Because of how the state space is defined, the RL bots can react to multiple enemies (up to three) in its field of view, while the StateMachine bot only reacts to one enemy. For state machine controlled bots to react to multiple enemies, this behavior needs to be implicitly coded into the rules and may be difficult to tune.

VI. Conclusions

Both the arena and maze maps showed similar trends in combat and navigation statistics indicating the robustness of the RL controllers. The hierarchical RL and rule based RL controllers performed significantly better than the flat RL controller in the combat and navigation skills. The hierarchical RL bot performed best in the shooting accuracy objective, outperforming all other bots in the experiment. The rule based RL bot performed slightly better in the other objectives than the hierarchical RL bot, however the observational results showed that the hierarchical RL bot had a wider range of behaviors in combat scenarios. For example, in some situations with multiple enemies the hierarchical RL bot would flee the scene, whereas the rule based RL bot tended to stay with the fight.

The flat RL controller was able to learn some basic combat and good navigation behaviors, however not to the degree of the hierarchical RL or rule based RL. On the other hand, the flat RL controller was able to excel in the item collection task outperforming all bots including showing a slightly better performance than the state machine bot, which was explicitly coded to pick up items in its path.

The hierarchical RL and rule based RL controllers displayed characteristics of advanced FPS players, by initiating fights when they were confident of a win; while the RL controller displayed characteristics of beginner players, such as panicky shooting in all directions.

Both the hierarchical RL and rule based RL methods show potential for creating diverse behavior sets for FPS bots. Although the rule based RL bot showed slightly better statistical scores, the hierarchical bot has potential for creating more diverse and less predictable behaviors as the action set is not defined by hard-coded rules as the rule based RL bot is.

Although a simple hierarchical structure is used in the experiments, this can easily be extended to include other features of bot AI such as operating vehicles or using portals. Future work will attempt to increase the complexity of the bots including item and vehicle use. Player evaluation of the bot AI, including player enjoyment, will also be conducted in future work.

References

[1] M. McPartland and M. Gallagher, "Learning to be a Bot: Reinforcement Learning in Shooter Games," presented at Artificial Intelligence in Interactive Digital Entertainment (AIIDE), Stanford, USA, 2008.

[2] A. Burkov and B. Chaib-draa, "Reducing the complexity of multiagent reinforcement learning," presented at Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, Honolulu, Hawaii, 2007.

[3] Y. Matsuno, T. Ymazaki, and S. Ishii, "A Multi-Agent Reinforcement Learning Method for a Partially-Observable Competitive Game," presented at Proceedings of AGENTS'01, Quebec, Canada, 2001.

[4] C. M. Vigorito, "Distributed path planning for mobile robots using a swarm of interacting reinforcement learners," presented at Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, Honolulu, Hawaii, 2007.

[5] S. Whiteson and P. Stone, "Evolutionary Function Approximation for Reinforcement Learning," Journal of Machine Learning Research, vol. 7, pp. 877-917, 2006.

[6] T. Graepel, R. Herbrich, and J. Gold, "Learning to Fight," presented at Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Eductaion, 2004.

[7] K. Merrick, "Modeling Motivation for Adaptive Nonplayer Characters in Dynamic Computer Game Worlds," ACM Computers in Entertainment, vol. 5, 2008.

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

[9] S. Cohen, O. Maimon, and E. Khmlenitsky, "Reinforcement Learning with Hierarchical Decision Making," presented at Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), 2006.

[10] T. Watanabe and Y. Takahashi, "Hierarchical Reinforcement Learning Using a Modular Fuzzy Model for Multi-Agent Problem," presented at Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Montreal, Canada, 2007.

[11] M. Huber, "A Hybrid Architecture for Hierarchical Reinforcement Learning," presented at Proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Francisco, CA, 2000.

[12] J. Manslow, "Using Reinforcement Learning to Solve AI Control Problems," in AI Game Programming Wisdom 2, S. Rabin, Ed. Hingham, Massachusetts: Charles River Media, Inc, 2004.

[13] M. Vasta, S. Lee-Urban, and H. Muñoz-Avila, "RETALIATE: Learning Winning Policies in First-Person Shooter Games," presented at Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference (IAAI-07), 2007.