AY2019 Master Thesis

Features and Performance of Sarsa Reinforcement Learning Algorithm with Eligibility Traces and Local Environment Analysis for Bots in First Person Shooter Games

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Bundik Bettina Vivien

Student ID	5118FG13-8
Date of Submission	January 29th, 2020
Advisor	Prof. Toshiharu Sugawara
Research Guidance	Intelligent Software

ABSTRACT

Reinforcement learning agents embedded in first person shooter video games provide a good test bed for training artificial intelligence in complex environments. This paper applies tabular Sarsa(λ) algorithm with eligibility traces together with ε-greedy selection to develop successful and well-performing separate controllers for two groups of tasks that are fundamental in first person shooting games: navigation and combat. After effective training processes of these controllers, a proposed method of analysis of local environments of agents is introduced which is based on assigning numerical values to all cells of an agent's surrounding features and making a decision according to this value in order to choose the more fitting and preferred controller to use in an upcoming situation. Due to the limited observations of behaviours of learning agents in first person shooter games, the navigation and combat controllers are corresponding to stealth (commando) and aggressive styled combat strategies accordingly. A variety of experiments are concluded in the navigation and combat controllers and later the ultimate training with differing sets of parameters. It is shown that in the environment on this paper, a better performing navigation controller is achieved in a maze-like map layout. Also, evaluation of the ultimate training results states that similar levels of performance can be reached to those of the referenced papers, in terms of counts of collisions and absolute distances travelled being higher when the discount factor and the eligibility trace is also high. Agents can clearly develop stealth and aggressive combat strategies through ultimate training, even though their overall statistics stay on an average level and show no improvement to other methods. There is no clear convergence or pattern between parameters of the proposed idea and performance of agents. Although in one case, a new and effective strategy is developed where FPS bots are able to realize the healing abilities of the collectible items, thus collecting more in order to survive for a longer time in combat.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Throughout the years, various fields of different artificial intelligence algorithms implemented in video games have been researched and investigated thoroughly. Specifically reinforcement learning methods deserve a highlight to have shown their outstanding influence over numerous kinds of one-, two- and multiplayer games. These research topics are dedicated to investigating performance, features and behaviour of computer learning methods and to finding the most optimal solution to a defined problem. The basic goal in all these research topics is to achieve the highest quality of performance that the investigated video game allows. However, selecting the most perfect learning algorithm or method for an environment specified by the problem highly depends on attributes and the likeness of said environment and the nature of the video game.

My primary motivation for picking the above introduced research topic is mainly due to the fact that video game developing institutions occasionally apply complex artificial intelligence techniques to in-game characters (particularly first person shooter or strategy games) or test their game environments with artificial intelligence induced bots with the intention of finding unexpected errors, unreachable paths, dead ends and loopholes in their layouts. Also, learning bots of video games share plenty of common features with robotics and multi-agents systems; therefore, they can be applied to a diverse field of real-life situations, making this topic truly promising in the present and future as well. In spite of the huge amount of previous groundwork, experimentations on performance of various artificial intelligence bots are yet to be perfected, implying the existence of unexplored possibilities.

First Person Shooter (FPS) is a genre of video games which accommodates a human player acting as a specific character inside a 2- or 3-dimenson video game, who is equipped with one or multiple types of guns and engages in combat with computer-controlled bots or other human participants inside the game. Robotics and multi-agent systems especially share lots of similarities with FPS games, hence making them a good testbed for implementing artificial intelligence in situations with complex environments.

Bots in first person shooter games require the basic skills of navigating through an environment, initiating combat with opponents and aspects of survival. FPS agents must be properly trained with reinforcement learning and additional helping methods to acquire such abilities. It has been proven in previous research that the quality of performance of bots is higher and their goals are achieved slightly easier if agents learn the necessary skills for navigation and combat in separate training processes. Also, different ways have been tried in a fully simulated game environment for combining the skills learned in advance – either by further learning or based on predefined rules. Through various experiments, it was concluded that FPS bots are able to gain the abilities to perform almost as skillfully as hard-coded bots and state-machine opponents.

On the other hand, several factors still have a high influence on the effectiveness of FPS agents, such as parameters of the training process, difficulty levels of enemy bots of the game or density of obstacles in the aspect of environmental features etc. So it can be assumed that improving performance and obtaining constant stability for the bots in their game-play are still achievable, yet problematic and difficult.

Another general goal in this field of research is to create a first person shooting agent with qualities such as being efficient, being able to perform on a level high enough to match human players or hard-coded bots and having a diversity and unexpectedness in behaviour so that the bot can not be distinguished from human players during playing the game.

This paper aims to boost the overall qualities of first person shooter bots in various aspects with a proposed method based on a new idea of combining navigation and combat skills of an agent.

2 Bots in First Person Shooter Games

The following section presents the necessary background, related research, algorithms in use, definitions of the environment, states, actions, reward systems, the additional proposed method and its experimental results and discussion in a simulation of training agents in a first person shooting game.

2.1 Background

There is already plenty of successful research on reinforcement learning (RL) [1] embedded in robotics [2] and multi-agent systems (MAS) as well as in strategy games [3]. Recent experiments have shown that training artificial intelligence in shooter games became an efficient method for improving performance in complex environments altogether, due to the fact that FPS video games share a great deal of features with MAS and robotics. In cases where agents are equipped with sensors to interpret their environment and act according to specified rules and policy is a good example for this.

When discussing game-play and performance of artificial intelligence in first person shooter games, it generally includes performing simple tasks – as pathfinding, navigating through obstacles, picking up items, using items, shooting with weapons and changing between weapon-types depending on the current situation, if there is a variety of weapons to choose from – and finding a well-working and efficient policy through a training method which satisfies conditions and goals of game-play and performance by a fair amount of time into a few episodes of the game.

These conditions might differ according to the environment and rules of a specific game. For example, colliding with obstacles in the environment should be minimal, distance travelled and number of items collected should reach higher and of course the main goal of every FPS game: achieve a larger count of kills and a smaller count of deaths of the agents. Another condition would be to achieve a diversity of game-play styles such as stealth (commando), sniper or aggressive behaviour – these may be recognized mainly by an analysis of paths that the agents take, their preferred distance when shooting an enemy and spending a lot of time out in the open or hiding alongside obstacles etc.

Increasing quality of performance and game-play can be viewed in a different light. In a related field of research, the goal is to design an FPS game bot in a way that human players can not recognize that it is a computer, which is termed human-like non-player characters. Even a competition was held for measuring the human-like behaviour of FPS bots created by participating teams, which was called BotPrize and was set up in 2008. This could also be considered a form of Turing Test for the competing artificial intelligence bots of their projects. In addition to having diverse and adaptive features, this aspect of intelligent bots means that such a non-player character of a game would also include "human errors" and learning from its mistakes. This implies that development and adaptation of shooting skills is necessary over time but actually difficult to achieve due to the complexity of training.

With the aforementioned perspective, a game can develop into something more challenging and unpredictable, resulting in a more enjoyable experience. Opposingly, the use of artificial intelligence in commercial games has been defined by hard-coded factors for most of the time, such as scripting, rule-based methods or state-machines, which has drawn up problematic issues such as being predictable, their time-consuming development and maintenance when it comes to creating different styled bots in an FPS game. Few of the many reasons that instead of hard-coding non-player characters, training FPS bot behaviours by various learning methods should be the preferred choice.

Artificial intelligence induced bots have to deal with a large collection of tasks to learn and perform which is more easily handled if actions are distributed into groups – separate controllers as previously investigated in [4]. As results have shown, splitting navigation and combat tasks can improve the learning bots' performance by a fair amount.

As in many other fields of research, agents in FPS games train best by learning from experience, therefore FPS bots do not require absolute knowledge of their environment. Plenty of previous experiments have proved that agents can be trained to act intelligently by analysing only their local surroundings. Furthermore, reinforcement learning is flexible and it allows to continue updating the policy online which is an excellent solution, for example to the problem of time-consuming development and maintenance of various bot behaviours.

Reinforcement learning [1] is a successful machine learning method for an agent or agents to be able to interact with the environment, work (work together) towards a specified goal or goals and learn from the experience by receiving rewards or punishments depending on the situation and the actions executed. It was inspired by and is quite similar to human-like learning. As a numerical representation of the environment, a finite set of states can be acquired (state space), therefore a state holds processable information of the environment. Also, the agents have a finite set of executable actions defined for them (action space).

At each step T in time, an agent performs an action A in state S, receiving a reward R as a feedback from the environment in a numerical form (reward function). Agents are able to be trained for a policy which consists of state-action pairs that describe a mapping between states and how the executed actions perform in it. The policy can evolve or converge by a fair amount of time, resulting in the agents getting higher rewards and finally reaching the target condition(s).

The generalization and tabular approaches are the most frequently used policy representation methods. The generalization approach has a function approximator for modeling the state-action mapping but it is not the most optimal way to properly represent a complex problem such as an artificial intelligence in video games. Instead, the tabular approach provides a better solution in which a lookup table is used to map numerical values to state-action pairs. Although there is an issue with scalability in complex continuous domains [5], there have been plenty of solutions presented, one of which is data abstraction [6] and it is basically abstracting necessary information from a state S, instead of storing all of its data in the table.

To improve reinforcement learning even more, ε-greedy selection has been proven a well-working method when it comes to balancing between exploration and exploitation during the agents' training period. It is a dilemma, a tradeoff between exploring the environment further by having a chance to choose a different action, and exploiting the already built-up knowledge of the state-action mapping. A parameter ε is responsible for randomly choosing between the two scenarios – a random action is chosen (ε) of the time, and the action preferred by the policy (1 – ε) of the time.

Tabular Sarsa (which stands for State-Action-Reward-State-Action) algorithm is one of the most used and acknowledged reinforcement learning methods for similar research with multi-agent systems and robotics, and also with video games [7] [8]. Other reinforcement learning algorithms have been developed, such as temporal difference (TD) and Q-learning, but these approaches were proven to be successful in slightly different areas of research. Tabular Sarsa is an on-policy algorithm which allows the agent to learn a value based on the agent's current action A derived from its current policy. When an approach is off-policy (for example Q-learning), the agent learns this value by the greedy policy.

The referenced work [7] [8] and also this paper applies Tabular Sarsa with eligibility traces, termed Sarsa(λ), to evolve a policy good enough for the agents' navigation and combat strategies. Eligibility traces allow the agents to train faster, to learn sequences of actions and past actions can also benefit from the current reward. Sequences of actions are significant when building an effective combat strategy because performing combined actions is the key to defeating enemy bots successfully.

At each episode, the current state S is acquired from the environment (using data abstraction) and the agent selects an action from the executable action space using an action-selection policy (ε-greedy is preferred in this paper and in referenced work as well). In the pseudocode of Sarsa(λ), $Q(S, A)$ stands for the policy and $e(S, A)$ for the eligibility trace for a given state-action pair (S, A). γ is the discount factor (between 0 and 1), α the learning rate (how fast we are approaching the goal) and λ the eligibility trace (for updating each state-action pair of the table).

At first, all $Q(S, A)$ and $e(S, A)$ are set to zero. After choosing a preferred action to be executed at the current state S according to the chosen policy, a reward R is calculated together with a next state S'. Then, all state-action pairs and their eligibility traces are updated in the table according to this update equation.

$$Q(S_T, A_T) \leftarrow Q(S_T, A_T) + \alpha[R_{T+1} + \gamma Q(S_{T+1}, A_{T+1}) - Q(S_T, A_T)]$$

The pseudocode for the Sarsa(λ) algorithm with eligibility traces is as follows.

```
1: $Q(S, A) = 0$, $e(S, A) = 0$ for all $S, A$ initially
2: Initialize $S$
3: Choose $A$ from $S$ using policy derived from $Q$
4: For each update step $T$
5:          Take action $A$, observe $R$, $S'$ (reward, next state)
6:          Choose $A'$ from $S'$ using policy derived from $Q$
7:          $e(S, A) = 1$
8:          For all $S, A$:
9:                    $Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A)) e(S, A)$
10:                   $e(S, A) \leftarrow \gamma\lambda e(S, A)$
11:         $S := S'$
12:         $A := A'$
13: Until $S$ is terminal
```

The research in this paper uses the following policy derived from Q: choosing the highest Q-value with ε-greedy selection (lines 3 and 6 in the pseudocode).

In addition to the policy described above, a small learning rate α is included and linearly decreased throughout the agents' training period. This discount rate is D, which is applied at every iteration and is described with the following equation:

$$D = \alpha_i - (\alpha_e/n)$$

α_i stands for the initial learning rate, α_e for the target end learning rate and n means the total number of iterations.

2.2 Related Work

The following part is dedicated to explain significant groundwork related to this paper's research.

2.2.1 *Separating Controllers*

The authors of the work referenced by [4] created a 3-dimenson first person shooter environment for their experiments with the purpose of training agents as FPS bots with reinforcement learning. Sarsa(λ) training has a history of being effective when artificial intelligence is applied to video game environments, as proven by previous studies. Therefore the main core of the training was tabular Sarsa(λ) algorithm together with eligibility traces to allow sequences of actions to be learned and allow past actions to be rewarded by current actions as well. Which is fundamental when well-working navigation and combat strategies are needed to play a game.

The environment contained simple shooting game elements such as walls, obstacles and items. Agents were equipped with easily executable tasks and sensors to interpret their local environment for a definition of a state and for a later decision on actions. The authors created enemies for the agents to compete and fight against – non-reactive and hard-coded bots.

Their method of splitting up the tasks of agents into two separate controllers resulted in a better performance – less collisions with obstacles, more distance travelled and more items picked up, also commando and aggressive styled behaviours could be achieved according to their experiments. The navigation controller was responsible for pathfinding in a maze-like environment while collecting valuable items. The combat controller focused on shooting skills and survival aspects of the agents. In this paper the controllers were trained separately with their own tables of state-action pairs.

When training the navigation controller, experiments showed that collisions with obstacles were less when the eligibility trace was higher (set to 0.4 or 0.8), and more when lower (0.0) which indicates that performance improves when planning is used. On the other hand, when more collisions occurred (lower λ) agents could travel farther, and with a higher eligibility trace, agents occasionally got stuck near walls and could not gain much distance. Results were the same for items collected during the test run – agents that did not travel far could collect more items, although this meant that the condition for optimal collision count would not be satisfied.

Experiments were carried out on the combat controller as well and results state that agents were able to stand their ground against state-machine opponent bots quite well. In addition, different combat strategies could be trained – commando and aggressive style depending on parameters (eligibility trace and decay factor in the update function) which were observed through visualization of the paths that agents took during training and replay.

2.2.2 *Combinations of Separated Controllers*

Based on the aforementioned paper, the following research [7] extends its ideas, builds on them and combines the previously learned navigation and combat controllers with different methods.

First, the authors provided more detailed results of the trials that were described and carried out in the paper mentioned above. In addition to the already explained results of the navigation controller on collisions, distance travelled and items collected, tables showed that agents received more punishment during earlier iterations and more rewards later which concludes that agents were improving their policy through time and this policy was converging. Also, they managed to learn to navigate through the maze-like and arena-like environments, as shown by visualized paths of the agents. If the eligibility trace had a lower value (a little planning included or none at all), policies could be reached that were seemed to be more optimal.

About the combat controller – a bot accurately hitting a target or a bot dying did not happen often but agents were able to reach a better policy through time due to the fact that the second half of iterations had much more rewards than punishments. The authors also included data on shooting accuracy of bots and their kill-death ratios and according to these and previous results, fair combat strategies (commando and aggressive) could be learned when the eligibility trace was either 0.0 (lowest) or 0.8 (highest).

The basic skills for an FPS game were covered by the controllers explained above. In this paper, three kinds of combination of controllers were tried against a state-machine bot and a non-reactive bot. The Hierarchical RL bot used a separate reward scheme for learning to choose which controller to use in an upcoming situation. The Rule-Based RL bot substituted this training with hard-coded predefined rules (combat controller when an enemy is in sight, navigation controller otherwise). The simple RL bot was set to learn all tasks from zero without any previous knowledge.

According to test runs, the learned navigation policy of the simple RL bot was quite unstable, the agent was ignoring enemies and was unable to reach a good combat strategy. In terms of conditions for navigation policy and average kill count, both the Hierarchical RL bot and Rule-Based RL bot performed similarly well to state-machine bots. On the other hand, the simple RL bot managed to collect plenty of items while the other two methods resulted in lower item counts.

In conclusion, these different combining strategies were able to produce differently behaving agents and bots could learn a combat strategy good enough to outperform the state-machine bot.

2.2.3 *Adaptive Shooting for FPS Bots*

Various first person shooting combat styles exist among human players of these video games (for example aggressive and brave, or cautious and sneaky). The general aim in [8] is to create such combat designs for bots, resulting in less predictable non-player characters and a wider variety of game-play. Their chosen environment is the game Unreal Tournament 2004 with Pogamut 3 extension which provides for creation of bots in the game.

To reach their goal of FPS bots' adaptation of shooting skills over time, a method is introduced for reinforcement learning agents – Sarsa(λ) algorithm (with a policy of choosing highest Q-values together with ε-greedy method) with a new reward scheme that includes a dynamic reward signal representing the amount of damage the bot caused to its enemies. Adapting shooting over time also implies that a bot's behaviour can and will change later during a game because of it being still in training, thus making the game-play unpredictable and varying.

States are defined by categorizing and storing data of currently visible and close enemies around the learning bot (enemy's distance, direction, relative rotation and speed to agent etc.) and engaging in combat with said opponent. These factors are helping in anticipating the opponent's future movements and actions which is quite similar to human thinking. Actions were divided into six groups depending on the weapon-type that the agent can shoot with, implying that functionality of different weapons were taken into account by this.

A good combat strategy always includes the condition that a bot should cause the most damage possible to an opponent. The previously mentioned dynamic reward system provides for this –

hit points achieved by the bot gives the numeric value of positive rewards. Experiments were carried out between one previously defined learning agent and three state-machine opponents in a series of three games relating to the three difficulty levels of state-machine bots. Statistics showed that although the learning bots' shooting accuracy is quite low, they were still able to perform at a competitive level against their state-machine opposition. Obviously as difficulty increased, the reinforcement learning bots had less chance to pick up weapons or move through the map so their performance worsened in these cases.

All in all, there was no clear pattern of improving in quality of game-play.

2.3 Purpose of Research

Modern video games, especially first person shooter games are becoming more complex along with computer-controlled bots scripted in-game. This has brought along various artificial intelligence research with itself but these fields of work have plenty of opportunities yet to be explored.

The first two of the above mentioned related work had a self-created and implemented FPS environment for training reinforcement learning bots, while the third paper embedded learning agents into an environment from an existing video game which brings it closer to modern commercial FPS games. All related research states that it is significant to properly train both navigation and combat skills for agents but in case of FPS games, shooting carries a slightly higher importance in order to reach successful FPS bot strategies.

Therefore, my purpose of research in this paper is to generally increase quality of game-play for first person shooter bot agents trained with reinforcement learning, mostly by resembling the behaviour of the bot to that of human players. This implies that combat strategies of agents become unpredictable and adaptive which are key elements for an enjoyable game with non-player characters that have varying and changing game-play styles.

Improved navigation, path-finding and item-picking skills are also a fundamental part of building a good FPS bot strategy. Although unfortunately, even after a diverse field of research, a navigation training algorithm that is more optimal and faster than A* pathfinding algorithms is yet to be found. In order to raise the quality of performance of a bot's navigation skills, more traversing through the map and less colliding with environment geometry and obstacles are necessary, as explained in several reference works as well [4] [7]. Considering the bot's combat strategy skills, a higher survival rate and higher hit and kill counts of opponents is preferred [4] [7] [8].

These are the general goals for an FPS bot, as explained above. This creates a complex problem space that reaches high difficulty levels for computer-controlled agents, which calls for an aforementioned solution already researched and proven successful – data abstraction for the state space of the environment. After creating a state-action-reward structure and training FPS environment similar to that of the referenced papers' [4] [7], my purpose is to build onto their previously defined and tested navigation controller and combat controller by combining them with an alternate method – data abstraction. It is used to determine which controller is more suitable to execute, for a local environment that the agent is currently in. This decision is carried out by an analysing method of the specified local area of the FPS bot.

The reason for this idea was that features of smaller local areas of a whole map can differ depending on the location – a widespread map can include larger open areas (where an aggressive combat style is preferred) and areas filled with walls and obstacles as well (where stealth style is favourable). If a variety of FPS bot behaviours is previously trained and available for use, agents can certainly benefit from being able to choose between different styles of strategies depending on their current surroundings and agents might reach a higher quality game-play.

Once again, the general purpose of this paper is to recreate navigation and combat controllers similar to that of the related work, focus on perfecting the combat strategy by defining and implementing new ideas, and improving performance of FPS bots by the aforementioned general conditions of a high-quality game-play.

2.4 Methodology

Following is the description of elements, factors and algorithms necessary to successfully create a state-action-reward structure that is fit for artificial intelligence bots to be trained by reinforcement learning in a self-created first person shooting game environment and to work towards a goal to be satisfied.

The problem defined above and its environment require numerical representation that holds all necessary information for the agents to be able to accurately interpret the environment and the end-state conditions. Reinforcement learning allows agents to gain experience and learn from their mistakes along the way by storing previously encountered data and assigning numerical values to them in order to decide how a current action affects (positively or negatively) a current state.

This paper aims to create and train well-working FPS bots with the goals of proper navigation, path-finding and item-collection skills, as well as an adaptive combat strategy and shooting techniques. While this goal is general in the field and acting only as a base, new additions and ideas are going to be introduced, investigated and discussed about later.

A method of separating the whole processes of navigation and combat training is used, since it has been proven more optimal and faster for FPS bot training, referencing other research [4] [7]. These processes are referred to as Navigation Controller and Combat Controller, have separate training cycles and their own groups of tasks to appropriately train agents for – the Navigation Controller focuses on navigation and item-picking skills while the Combat Controller concentrates on shooting technique and survival.

In light of this scheme, the following attributes need to be defined: the environment, policies to be used, state space representation, executable actions and a fitting reward scheme are crucial for a proper reinforcement learning agent-system. These definitions highly depend on the problem to be solved and goals to be reached.

Below are the details that were used in the research of this paper.

2.4.1 *Representation of Environment and Agents*

Investigation and experiments were carried out in a self-built first person shooting environment that involves all common elements of a commercial FPS video game – predefined map layouts, bots that keep track of their own data fields (position, health, shooting cooldown etc.), functions for shooting mechanisms and management of respawning of items and agents.

In all referenced work, investigations were carried out in a 2-dimension environment such as in this paper as well. Only *X* and *Y* planes are dealt with due to video games designed in 3 dimensions being overly complex, also with a 2D layout, learning agents are able to focus on the actual training of fundamental tasks more.

Four different map layouts were created due to previous research stating that a variety of environment design is necessary for training, if agents are given multiple tasks to learn. If the features and usage of said tasks are differing, it is easier and more optimal to create a layout for each group of tasks and specifically train them in their corresponding environment – which was proven useful in several previous experiments [4] [7].

Attributes of maps used in this paper are as follows. The size of each layout was set to 50 by 50 metres. Hypothetically, the unit of distance measurement is metres, although during implementation the term ‘cell’ (here an element of an array) is used accordingly. Walls are a significant building part of maps – they make up the four borders of a map and some layouts include walls inside the borders as obstacles, resembling a maze-like design. Each cell of a map spreads about 1 by 1 metre and is assigned a predefined type: wall-, item- or empty-type. The figures below denote walls with X-es, items with blue squares and agents with their ID numbers.

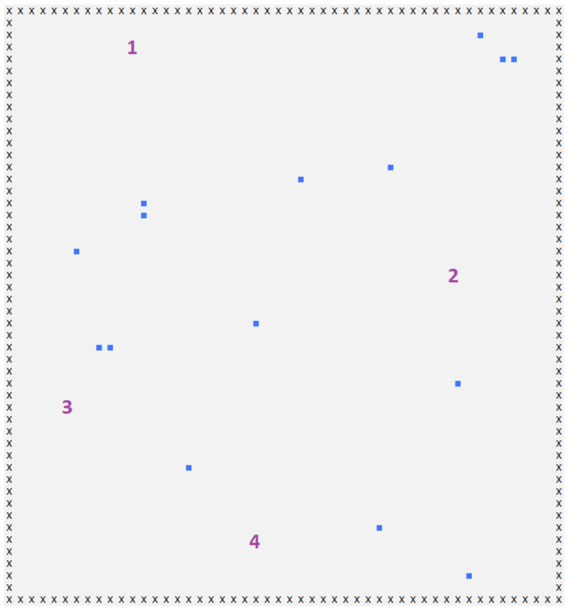

Fig. 1. The Arena map

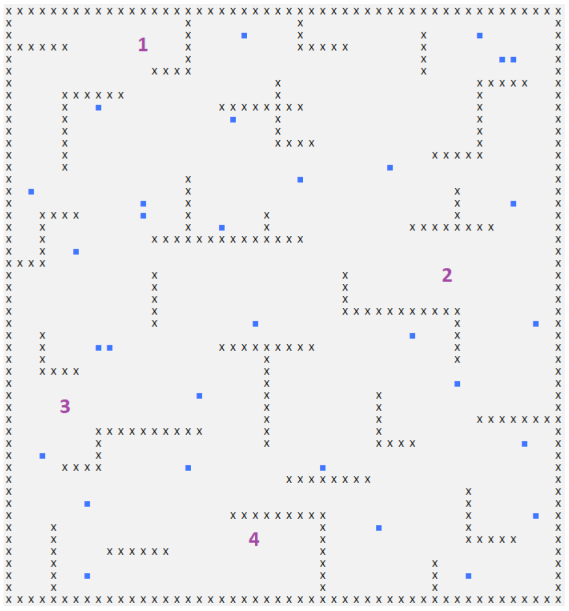

Fig. 2. The Maze map

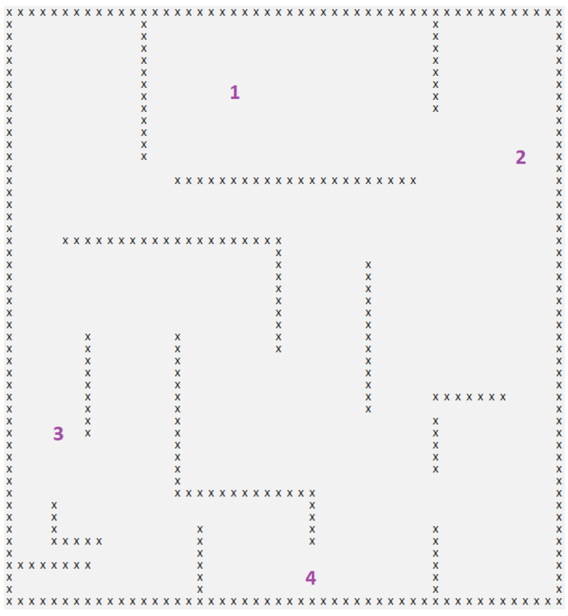

Fig. 3. The Combat map

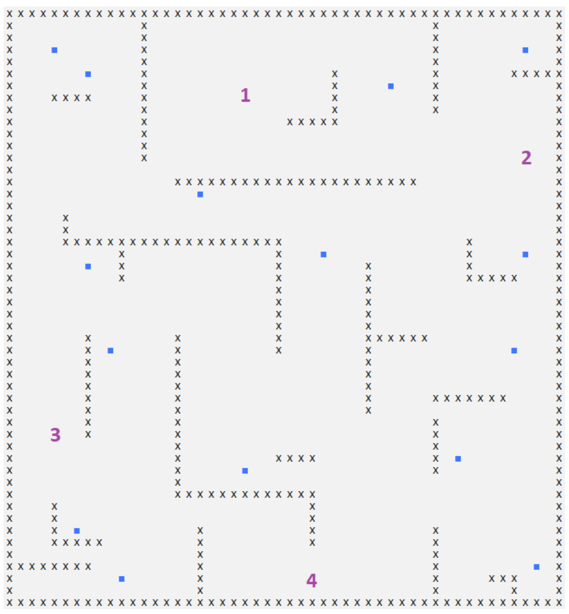

Fig. 4. The Ultimate map

As seen on illustrations, each design includes four starting positions for agents. The Arena map (Fig. 1.) has walls on its four borders but no obstacles (walls) otherwise, 15 collectible items placed throughout the layout and shooting weapons is not allowed for bots. The Maze map (Fig. 2.) has a maze-like design with borders and plenty of obstacles, 30 items planted and shooting is still not allowed for the agents. The Combat map (Fig. 3.) is similarly maze-like – obstacles were placed but not so densely, items are excluded and shooting is allowed. The Ultimate map (Fig. 4.) brings about all possible actions with having the design of the Combat map's obstacles, including 15 items and allowing shooting for agents.

The Arena and Maze map are specifically created to train agents for the skills of navigation, path-finding and item-collection, whereas the Combat map provides for training agents to shoot and to survive. Finally, as mentioned before, the Ultimate map covers all tasks for an ultimate testing environment.

The representation of agents is as follows. An FPS bot carries and updates its own data fields which are necessary to know during training cycles. Agents have a predefined starting position which also equals their respawn position after they are defeated by an enemy bot.

Data fields of a bot are composed of its ID number, absolute location on the map, direction (0 to 360 degrees), current health points (maximum is 100 points), a timestamp for shooting cooldown (1 second) and various counting values for statistical reasons (items picked up, opponents killed, count of own deaths etc.).

Other features that had to be determined include a bot's moving speed which corresponds to its position moving 1 metre forwards or backwards, turning angle which is always 5 degrees left or right and view range which is defined to be exactly 120 degrees. An agent's view range spreads from +60 to –60 degrees relatively to its current facing direction. This detail contributes a lot to modern commercial FPS games and their human-like perspective for its characters.

Also, the positions of agents' have x and y values due to the 2D representation and they are implemented by floating-point arithmetic (double values) in order to have more freedom during their movements. Due to complex calculation and analysis processes during training, a double representation system is included for agents. They keep track of their absolute position which refers to coordinates in a matrix (n, m) of a program in which case the origin is at "the top left".

$$x_{abs} \in [1..n] \quad y_{abs} \in [1..m]$$

Also, this location is converted into a relative position which relates to the Cartesian coordinate system where the origin is at "the center".

$$x_{coord} = y_{abs} - y_{origin}$$

$$y_{coord} = x_{origin} - x_{abs}$$

The above defined double representation of agents is significant during implementations of detection for sensor values, detection of closest visible enemy, the bot's shooting mechanism and calculating the bot's chances at getting shot.

For each game and each map layout, all the agents' learning method, functions, data fields and features are identical.

Valuable collectible items are also present in some map layouts. When shooting is an allowed action in the training, items also act as medical kits, giving +20 health points to the agent that picked up the item – meaning that items are essential for survival and better game-play in a shooting game. After an item is collected, it respawns after a 100 iterations of the game on the same location that it was placed initially.

2.4.2 *States*

The application of reinforcement learning requires a state space of the environment to be defined. A state is an element of the state space and its main function is to extract information from the environment for it to be interpreted by the agents.

An accurate definition of state is crucial due to the fact that results and performance of learning agents highly depend on states in all fields of artificial intelligence research. State representation differs depending on the controller currently in training – navigation or combat.

In case of the Navigation Controller, the following features of the environment need to be considered for state representation: obstacles and items. Using other related work as a base [4]

[7], FPS bots in this paper are equipped with three separate sensors to anticipate walls and items of the map nearby the agent. Agents have a 120 degree view range, from +60 to –60 degrees relative to the agent's direction *angle*, which affects their sensors as follows.

Table I – Angle ranges for sensors of agents

Sensor	Range of angles
Left	$+60° \leq angle_A < +20°$
Center	$+20° \leq angle_A < -20°$
Right	$-20° \leq angle_A \leq -60°$

Agent *A* has its left sensor from +60 to +20 degrees, the center sensor from +20 to –20 degrees and the right sensor from –20 to –60 degrees. A sensor returns two different values – one for walls and one for items, so a state for Navigation Controller has six sensor values altogether.

An approximate distance of obstacles and items within reach need to be accounted for as well, resulting in the previously mentioned values being 0 (none), 1 (close) or 2 (far). Close is considered 4 metres or less and Far is between 4 to 10 metres.

Table II – Values and distances for sensors of agents

Sensor value	Sensor category	Distance
0	None	–
1	Close	4m or less
2	Far	Between 4m and 10m

The Combat Controller needs different traits of the environment for its proper training. Other bots, enemies are the most important factor, implying that a search needs to be performed on the map for nearby (the closest will be considered) and visible opponents (visible means the potential enemy is inside the bot's view range and not blocked by obstacles of the map).

Therefore, a state for combat training consists of the following data fields: information about the nearest visible opponent – if there is one and its estimated distance from the acting agent –, a logical value whether the agent's weapon is on shooting cooldown and the bot's own health points.

If an opponent exists that satisfies the aforementioned conditions, a decision is made about its relative distance to the agent and it is categorized into either Close (between 0 and 15 metres), Medium (between 15 and 50 metres) or Far (above 50 metres) [8].

Table III – Categories and distance of opponent bots

Category of enemy	Distance
Close	15m or less
Medium	Between 15m and 50m
Far	50m or above

These attributes can help properly determine an agent's strategical step – its success rate and estimated outcome – in a potential combat situation.

2.4.3 *Actions*

Reaching progress and proper training of agents in a reinforcement learning environment can be accomplished by the agents executing actions to transfer from one state to another, eventually arriving at an end-state with the necessary conditions satisfied.

For the FPS bots in this paper, various actions are available which resemble the basic operations that human players and non-player characters are able to execute in a common FPS game. In related work [4] [7], similar to the following representation of actions is used.

Table IV – Actions of agents

Actions
Turn left
Turn right
Move forwards
Move backwards
Collect item
Shoot

Agents have the ability to turn either left or right and it is always by 5 degrees. Turning is essential for adjusting a weapon's crosshair to a potential target when an opponent is found inside the view range. Bots are able to move forwards or backwards in their own current direction and one step equals 1 metre in length. The following equations are used when calculating the next possible location (x', y'), plus or minus *distance* (1 metre in case of moving a cell) in the agent's current direction (angle) from the current position (x, y).

$$x' = x + \cos(angle) * distance$$
$$y' = y + \sin(angle) * distance$$

There is an action for picking up an item which results in the agent gaining +20 health points if shooting is allowed in a training game. If this act is performed, the collected item disappears from the map and respawns on the same location after 100 iterations.

An agent can shoot with its equipped weapon which is defined as an instant-shooting laser type and it causes –10 health point damage when an opponent is hit. If shooting is instant, then there is no time required to reach a target when the weapon is fired, also, a laser type weapon is easier to handle in-game because there is no need to implement weapon-reloading and picking up extra ammunition. These being said, agents can fully concentrate on the actual learning of successful shooting and combat strategies. In addition, a weapon has to be put on a one second shooting cooldown once it is fired, in order to avoid a constant rain of fire on a target and to have a fair battle – for comparison, bots are implemented in a way that they can execute 10 actions in one second.

Altogether, agents are given 6 different actions in a situation to choose from during their training. More specifically, when the Navigation Controller is active, agents have the actions of turning left, turning right, moving forwards, moving backwards and picking up an item, while the Combat Controller adjusts these to turning left, turning right, moving forwards, moving backwards and shooting. As mentioned earlier, separating actions in a specific way makes them easier and simpler to train agents for. This isolation is corresponding to the predefined maps explained before (picking up items is allowed in the Arena and Maze maps, while the Combat map provides for shooting).

2.4.4 *Reward System*

As proper representation of states and actions is fundamental for a reinforcement learning environment, its reward scheme has the same weight in affecting the outcome and performance of the training. It is a depiction of the overall goals that the agents work towards.

Therefore, functionality of a reward scheme is to assign positive or negative values (rewards or punishments) to agents when an action is executed in a given state – the value depending on how the action alters the state and the agent in the process. If an action affects a state and agent in a way that it benefits reaching the goal, then a positive reward is given. If it works against, a negative punishment has to be delivered.

The following definition of a reward system is based on referenced work and their experiments on its usage and successfulness with FPS bots [4].

There are two isolated reward schemes for the Navigation Controller and Combat Controller due to their separate training processes and differently prioritized goals. Once again, the Navigation Controller aims to perfect the skills of navigation, path-finding and picking up items while the Combat Controller trains shooting and surviving strategies of FPS bots.

According to this, goals of the Navigation Controllers include minimizing the count of collisions with obstacles, maximizing the distance travelled and maximizing the count of items collected. As for the Combat Controller, each bot has a higher priority for gaining more kills of opponents and for keeping their own death count lower.

The following tables show two separate reward schemes that were applied to the training agents.

Table V – Reward scheme of Navigation Controller

Action	Reward
Collision	– 0.000002
Moving	+ 0.000002
Item collected	+ 1

Table VI – Reward scheme of Combat Controller

Action	Reward
Hit	+ 1
Kill	+ 1
Miss	– 0.000002
Killed	– 1
Wounded	– 0.000002

During training navigation, a small punishment is delivered when an agent collides with walls and obstacles in the environment (–0.000002) and a small reward is given for moving (+0.000002) – the purpose of these aspects is to force the agents to move farther in the environment and explore more. These values became quite small due to their high frequency of happening during experiments. On the other hand, an agent collecting an item rarely occurs compared to colliding and moving, so a larger reward is assigned in this case (+1).

The reward system for the Combat Controller focuses on shooting (hitting, killing or missing a target) and being shot at (wounded or killed). The aim for every combat strategy is to increase accuracy of shooting at opponents, reaching a higher kill count while avoiding enemies' attempts at shooting at the agent and staying alive for as long as possible. Which is why a large positive reward is given for hitting (+1) or killing (+1) an opposing bot, a large punishment takes place when the acting agent is killed (–1), and a small punishment is delivered for shooting but missing (–0.000002) a target or being wounded (but not killed) during a shooting attempt (– 0.000002).

Details about functions and methods related to agents colliding with walls and their shooting mechanism of the environment are stated below.

An analysis needs to be executed at each iteration to determine whether an agent is colliding with an obstacle and its threshold for distance is specifically set to 0.4 metres. The reason for allowing such a close range is that the positions of agents' are represented by floating-point arithmetic in implementation and this provides them freedom in their movements.

The shooting mechanism of agents is as follows. With the purpose of calculating possible rewards and punishments in a given state where one agent takes a shot at another agent, a process is needed to determine feedback of an agent executing the shooting action and feedback of an agent getting shot at.

In case of an FPS bot shooting its weapon, the following method is used. Shooting action takes the agent's current direction into account and the vector of this angle acts as the trajectory of the fired bullet. Every cell along this line is checked, increasing distance by one metre until a border of a map, an obstacle or an opponent is finally reached. If a bullet goes out of range (hits an end of the map) or it gets blocked by an obstacle of the map, then a 'miss' punishment is added to the shooting action. If another enemy bot is found along this vector, 'hit' or 'kill' rewards are given to the agent according to the opponent's remaining health points.

The previously described method is performed and its outcome is hypothetically calculated for each learning agent at each iteration when the reward lists of all executable actions are needed to decide on the actions' Q-values and which to choose. Agents consider these Q-values when updating the learning table of state-action-pairs and when the best action is selected in the given state – if the chosen action turns out to be shooting and an enemy bot is in the crossfire, the opposing agent's health is reduced by –10 points and if its health points reach zero, the enemy bot dies and respawns on its starting position.

In case of the possibility of an agent getting shot or killed, other enemy bots' locations and directions must be studied for considering all potential outcomes of a situation. A learning agent has the means to calculate other agents' shooting trajectories, only when said enemies can be found inside the agent's view range – which method corresponds to a human player in FPS video games anticipating the opposition's movements and possible future acts.

Using a similar calculation process as explained above, a vector is constructed for every (visible) opponent along their current directions and the learning agent checks whether its own position will be in the crossfire of an enemy's bullet. If an agent finds itself in range of an opponent and along its shooting trajectory, a 'wounded' (the agent still has enough health points after –10 damage) or 'killed' (health points reached zero and the agent dies) punishment is given.

Same as the previous, this mechanism is also carried out at each iteration, for every playing agent and for every executable action – due to the fact that each task results in a different state, which all have to be examined for potential fire from other enemy bots.

Agents are able to prepare and accomplish a fair level of perception by the above method.

2.4.5 *Training Algorithm and Modifications*

Representation of the environment, states, actions and reward systems have been established and explained in detail for the reinforcement learning FPS bots of this paper's research.

The aforementioned tabular Sarsa(λ) algorithm with eligibility traces is the chosen method for the agents to be successfully trained and their performance increased in terms of navigation, path-finding, item-picking and adaptive shooting skills. According to related work on adaptive shooting [8], the actual policy of their tabular Sarsa(λ) algorithm is selecting the action of the highest possible Q-value in a given state, also introducing ε-greedy method as an addition to the learning process – this paper applies the same mechanism to its FPS bots.

There was a paper published on eligibility traces and partially observable Markov Decision Processes [9] where the authors state that there is in fact a bug, an error in the general pseudocode of the Sarsa(λ) algorithm. There seemed to be problems occurring that punishments and rewards from previous episodes were affecting the results and eligibility traces of new episodes. States that the agent has visited in the episodes before are either rewarded or punished for the actions that the agent does in new episodes. As all Q-values are updated after every step of the agent, eligibility traces should not be, which is why a correction was made – eligibility traces need to be reinitialized after every episode and the problem disappears.

With the training algorithm and a slight modification introduced, the structure of the whole reinforcement learning process is as follows.

Simple tabular Sarsa(λ) algorithm with eligibility traces and ε-greedy selection is applied to train agents for the Navigation Controller. The same method provides for the Combat Controller of the agents, only with a separate table of state-action-pairs and their Q-values – thus training in isolation.

Although the algorithm is identical, the case of ultimate training on the Ultimate map proposes a new idea of data abstraction which will be termed Analysis of Local Environment and is used to decide on which controller to use in an upcoming state.

2.5 Analysis of Local Environment

The following section is dedicated to introduce the proposed method of this paper.

Data abstraction for reinforcement learning has been proven effective and advantageous in countless previous research with their purpose of reaching quality performance of agents and the goals of their environment, as mentioned earlier. In light of this, the paper proposes a new idea as data abstraction method to be added to the reinforcement learning environment of this paper.

After successfully training FPS bots with tabular Sarsa(λ) algorithm with eligibility traces for a Navigation Controller and Combat Controller using separate tables for state-action-pairs, an ultimate training of agents is required. Ultimate training is carried out using the Ultimate map layout and all previously defined actions are allowed to be chosen – meaning that collecting items and shooting are both available tasks.

The referenced work here, [7], presented three different new methods to combine and make use of the previously trained Navigation and Combat Controllers of FPS bots: their Hierarchical RL bot used training (a separate reward system) to learn which controller to use in a situation, their Rule-Based RL bot was given predefined rules for choosing a controller and their simple RL bot learned every task from scratch with zero former knowledge.

The proposed method of this paper, after several experiments and investigation, compares its results with performance of the above mentioned methods of combining Navigation and Combat Controllers.

The idea is to suggest an approach for data abstraction that chooses the more favourable controller at a given state during training based on an analysis of the agent's local environment and making the decision accordingly. In other words, FPS bots are trained by an analysis of their surroundings to select a previously learned behaviour for their next step which they consider more beneficial at the current state.

According to the proposed method's perspective, learning agents can achieve a variety of navigation and combat strategies in a shooting game by either different sets of parameters or implementing numerous algorithms – but in reality, commercial FPS games tend to have more complex map layouts, settings and environments. Which is why it would be more convenient if FPS bots were able to select the more fitting behaviour for their current state, with the purpose of increasing their quality of game-play and being prepared for any kind of situation.

The proposed method also takes map layouts into consideration, meaning that a large map of an FPS video game in reality may contain smaller or wider open areas as well as parts that have a higher density of obstacles, which requires learning agents to have the opportunity to choose a preferred behaviour.

The proposed method with its aims mentioned above examines and studies attributes and features of the current local area of an agent according to the following.

Let us consider a close environment with the currently learning agent in its center position, with a size about 7 by 7 metres (a matrix of cells around the bot). It is important to decide on the size of a local area by estimating it in the middle, because about 4 metres is viewed to be close and about 10 metres to be far in this paper's environment, due to the fact that the proposed method is supposed to focus on the nearby environment.

This local matrix of the agent contains attributes of the agent's surroundings such as the types of cells it holds: obstacles, items, empty tiles and enemy bots. The ratio of said types are taken into account by assigning a value on a 1 to 4 scale to each cell that the local area involves, as shown by the table below.

Table VII – Assigned values and behaviours of Local Environment Analysis

Type of cell	Assigned value	Controller	Behaviour
Wall	1	Navigation	Stealth ++
Item	2	Navigation	Stealth +
Empty	3	Combat	Aggressive +
Enemy	4	Combat	Aggressive ++

As a straightforward division cannot be decided upon, a range of preferred strategies is established corresponding the analysed cell-types.

In case of wall- or item-type, the Navigation Controller is suggested, while empty- and enemy-type cells consider the Combat Controller more beneficial to use. Moreover, wall-type cells have a slightly stronger claim for navigation strategy (indicated by ++) than item-type cells (denoted by +), also an enemy-type cell calls for combat strategy more (++) than an empty cell (+).

After the above defined values are calculated for the local environment of an agent, the average of them is taken – Navigation Controller is chosen if the value falls between 1 and 2.5, while Combat Controller is selected if the value is between 2.5 and 4 (2.5 included).

Table VIII – Value intervals for Local Environment Analysis

V: Value of local area	Controller	Behaviour
$1 \leq V < 2.5$	Navigation	Stealth
$2.5 \leq V \leq 4$	Combat	Aggressive

As seen on illustrations, stealth behaviour is associated with the Navigation Controller, while aggressive style is assigned to the Combat Controller. The reason for this is that several experiments in referenced papers have shown and discussed about the paths throughout map layouts that their learning FPS bots took during training and replay periods.

According to said paths of agents, traits like exploring the map more, avoiding shooting constantly and running away when there is a conflict refer to stealth behaviour (commando style strategy) of an agent, whereas if bots go after opponents for most of the time, engage in lots of conflict and exploration is not their highest priority then that describes an aggressive behaviour.

Also, general goals of training navigation involve traversing through the map, reaching a higher travelled distance and increasing the number of items collected. In case of the Combat Controller, the agents are able to train for either stealth or aggressive styled strategies (different use of parameters), but examining overall statistics state that an aggressive behaviour is more advantageous to satisfy the general purpose of combat – higher count of kills and lower count of deaths.

Therefore, a correlation can be based on the aforementioned points of view. A stealth kind of behaviour relates to FPS bots of the Navigation Controller, as an aggressive styled strategy associates with agents of the Combat Controller of proper parameters.

Once again, the proposed method requires previously and properly trained Navigation and Combat Controllers with their own separate tables of state-action pairs. Then, at every iteration during ultimate training on the Ultimate map, an analysis is executed for the agents' local environment acting as data abstraction and a decision is made according to the result of said method – at a given state for a given agent, Navigation Controller is selected if a stealth style action is preferred or Combat Controller is chosen if an aggressive behaviour is more fitting. Experiments with the proposed idea are carried out by ranging the size of local environments of agents and the training of agents turned on and off in order to see how these changes affect the learning agents' results through training and replay.

2.6 Experimental Setup

Following is the structure and setup of experiments that were carried out according to the research of this paper.

As recreation of the Navigation Controller and Combat Controller from related work is the base of the proposed method here, the training of said controllers took place before an ultimate training of FPS bots, all defined by aforementioned algorithms, descriptions of agents, states, actions and reward systems. Also, the previously introduced map layouts were put in use.

The Navigation Controller was constructed and tested on the Arena map and the Maze map consecutively, using the same table for state-action-pairs. Both the Arena and Maze map has items placed throughout their layout and no shooting is allowed. Next, the Combat Controller is trained on the Combat map with its own separate table for state-action-pairs. The Combat map involves no items, has a maze-like structure and allows shooting.

Once both controllers are prepared, the ultimate training of agents follows, using the Ultimate map which holds items, obstacles and allows shooting as well.

In addition to the simulation parameters shown below, reward schemes were integrated for navigation and combat that were specified in previous sections. Each map and each episode runs 4 learning agents and one episode's iteration limit is set to 10 000.

Table IX – Simulation parameters

Parameter name	Value
Map size	50
Number of agents	4
Agent's view range	120
Agent's health	100
Shooting cooldown	10
Item-respawn iterations	100
Medical kit	20
Collision threshold	0.4
Enemy close	15
Enemy far	50
Enemy hit	–10
Local environment size	Changing
γ	Changing
λ	Changing
Initial α	0.2
Target-end α	0.05
ε	0.2
Iterations	10 000

A table shows the necessary parameters for all test runs, including attributes of agents, features of game-rules and parameters of the learning algorithm.

One episode consists of 4 agents playing the FPS game for 10000 iterations. All map layouts are 50 by 50 metres.

The agents share the same traits: 120-degree view range, 100 health points, 1 second shooting cooldown (10 actions' worth). Items respawn after 100 iterations and give +20 health points when collected. An agent's position is considered a collision with walls of the environment when its distance is less than 0.4 metre from an obstacle.

An enemy bot falls into the Close category when it is between 0 and 15 metres from the current learning agent, and it becomes the Far category when between 15 and 50 metres. Shooting action causes –10 points damage to an agent's health.

In initial tests, the size of a local environment of an agent is set to 7 by 7 metres, later 10 by 10 metres and 15 by 15 metres are tried as well.

Following is an explanation of the learning rate of the algorithm used in experiments.

Tabular Sarsa(λ) algorithm with eligibility traces requires the following parameters to be determined: γ, λ, α, and ε.

α is the learning rate of the whole process and before the training starts, it is initially set to 0.2. According to referenced work [4] [7], α is linearly decreasing during a training period corresponding to the following equation.

$$D = \alpha_i - (\alpha_e/n)$$

D is the discount rate through an episode, α_i is the initial rate (0.2), α_e is the target-end learning rate (0.05) and *n* means the number of iterations in an episode (10000).

The learning algorithm used in this paper also applies ε -greedy selection – a tradeoff between exploration and exploitation – at every iteration and ε is set to 0.2 which means that a randomly chosen action is executed 20% (ε) of the time, the best action selected by the algorithm 80% of the time (1 – ε).

γ is the discount factor (between 0 and 1) and λ stands for the eligibility trace (used to update every state-action pair of the table at every iteration). The values of γ and λ have the most influence on the outcome of the learning process. For example, in related work it was stated that the agents' collision count corresponded mostly to the value of λ – smaller λ meant almost no collisions, although overall distance travelled was higher when λ was smaller which concludes that with more distance, more items could be collected.

In case of their Combat Controller, FPS bots training on the following sets of parameters developed aggressive behaviour: γ being 0.4 and λ being 0.8, or γ being 0.8 and λ being 0.8. Stealth behaviour was accomplished with γ set to 0.4 and λ to 0.4.

Those being said, the simulations of this paper also present trials with varying γ and λ values for each separate training of the Navigation and Combat Controllers as seen in the next table.

Table X – Trial parameters for Navigation and Combat Controllers

Trial	**γ**	**λ**
1	0.0	0.0
2	0.0	0.4
3	0.0	0.8
4	0.4	0.0
5	0.4	0.4
6	0.4	0.8
7	0.8	0.0
8	0.8	0.4
9	0.8	0.8

The ultimate training is executed with the proposed method of analysing an agent's local environment and choosing a controller according to this result for its next action. Therefore, a set of experiments are carried out for the ultimate training where the size of local environments of agents are either 7 by 7, 11 by 11 or 15 by 15 metres. In addition, the case is tried when training the agents is stopped and only the previously created Navigation and Combat Controllers are used, and it is also tried when agents are still learning during ultimate training and updating the same tables of state-action-pairs that they have trained for before. Trials for ultimate training are found below.

Table XI – Trial parameters for Ultimate training

Trial	γ	λ	Size of env.	Training
1	0.4	0.0	7 x 7	Off
				On
			11 x 11	Off
				On
			15 x 15	Off
				On
2	0.4	0.4	7 x 7	Off
				On
			11 x 11	Off
				On
			15 x 15	Off
				On
3	0.4	0.8	7 x 7	Off
				On
			11 x 11	Off
				On
			15 x 15	Off
				On
4	0.8	0.8	7 x 7	Off
				On
			11 x 11	Off
				On
			15 x 15	Off
				On

The trials performed are evaluated according to the following standards that are also referred to in other related research [4] [7].

The aim of creating a well-working Navigation Controller is to minimize collisions with obstacles and maximize distance travelled and the number of items picked up. A properly trained Combat Controller works towards a higher count of kills and lower count of deaths, survival. In light of these, statistics of experiments have to be drawn and analysed such as overall count of collisions, absolute distance travelled, number of items collected and kill-death counts (clearly the kill-count equals the death-count of a game).

Considering the ultimate training with the proposed method of the paper, an additional evaluation is mandatory – a percentage ratio to which controller is selected, preferred more by the agents during an episode, depending on the size of local environments and whether training is turned on or off.

In conclusion, quality of game-play and performance of learning FPS bots and features of their strategies are going to be studied and investigated by the aforementioned standards.

2.7 Experimentation and Analysis

As specified by the experimental setup, tests were carried out to construct a Navigation Controller, a Combat Controller and an ultimate training process to create well-working learning FPS bots with quality strategies and performance. Investigation of said experiments took place according to standards precisely defined in the previous section.

Later, this paper's experimentation is compared to performance of the Navigation and Combat Controllers of [4] and the methods of Hierarchical RL bot and Rule-based RL bot of [7].

Multiple test runs were executed. One test run was divided into four segments as stated by the following table.

Table XII – Simulation parameters and outline

	Training	**Map**	**Agents**	**Iterations**
1.	Navigation	Arena	4	10 000
2.	Navigation	Maze	4	10 000
3.	Combat	Combat	4	10 000
4.	Ultimate	Ultimate	4	10 000

The Navigation and Combat Controllers were created for the ultimate training that followed, each with training cycles of 10 000 iterations. Experimental results shown and discussed below were derived from replay episodes of the training with the same parameters as listed above.

2.7.1 *Results of Navigation Controller*

First, a successfully trained Navigation Controller is needed. Four learning FPS bots were let loose in the Arena map to execute their learning process for 10000 iterations. Then, with the same settings, the bots were trained even more in the Maze map, using the same table for state-action-pairs.

The outcome is inspected and evaluated by the agents' counts of collision, absolute distance travelled and numbers of items collected.

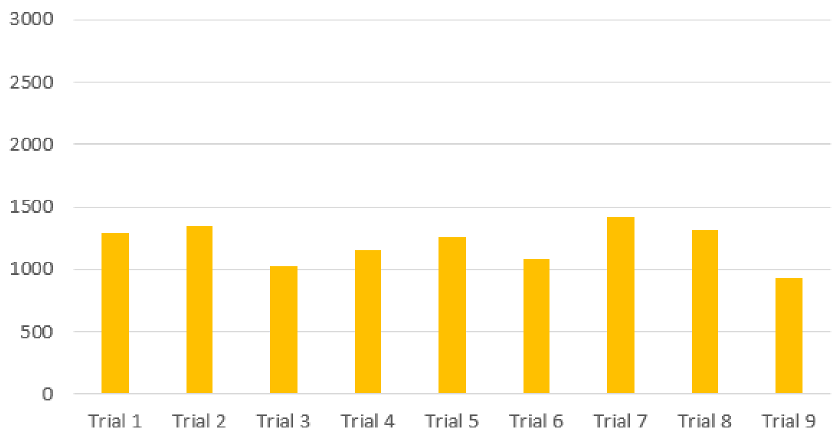

Fig. 5. Collision counts of Arena map

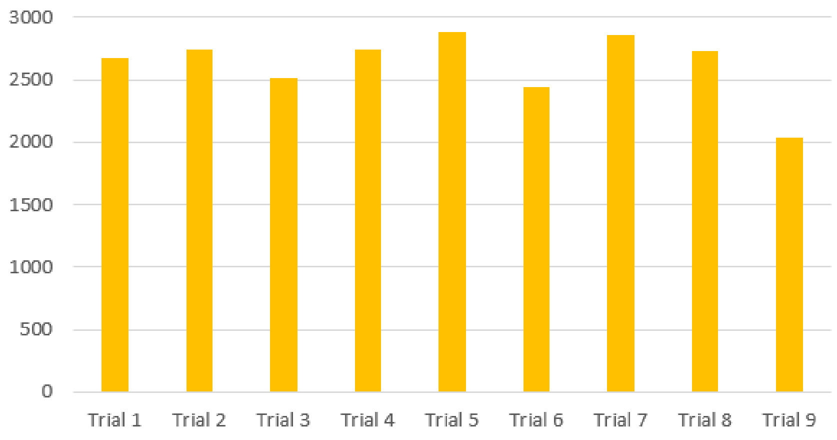

Fig. 6. Collision counts of Maze map

Fig. 5. shows how many times the agents collided with walls during training in the Arena map and Fig. 6. in the Maze map.

Although it is a general goal in several related work to minimize the count of collisions, after observing the agent's behaviour and the paths that they took, in others' experiments [4], the bots were able to collect more items and travel farther when their collision count was higher, meaning that in some cases agents might sacrifice their goal of less colliding and prioritize traveling and item-picking.

Trials of this paper have similar results. During trials 3, 6 and 9 where λ is higher (0.8), it can be noted that counts of collisions have decreased compared to other trials, both in the Arena and Maze maps. Also, overall collision counts in the Maze map are ~1.5 times higher than those of the Arena map which is due to the density of obstacles of the Maze map.

The next aspect is absolute distanced travelled by the agents.

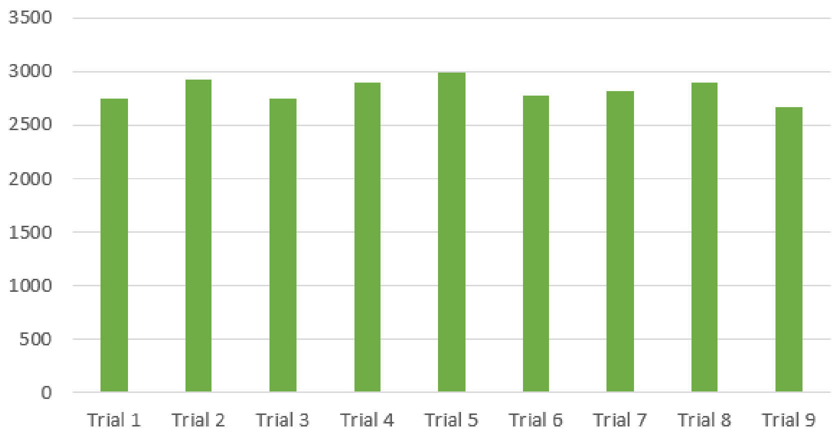

Fig. 7. Distances travelled on Arena map

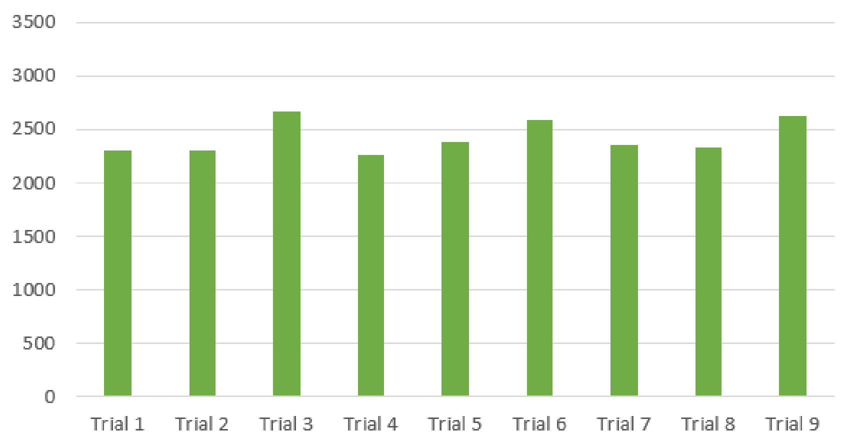

Fig. 8. Distances travelled on Maze map

Fig. 7. observes the absolute distance travelled by the agents on the Arena map, Fig. 8. on the Maze map.

Experiments of this paper show that collision counts and distance travelled are indeed connected. Although this correlation is slightly differing in cases of the Arena and Maze maps. Learning bots were able to travel ~500 metres more in the Arena map because of the lack of obstacles. Also, when λ is higher (0.8, trials 3, 6 and 9), collisions are lower but distance travelled decreases with them, which was told in related work that agents sacrifice their collision counts in order to travel more.

In spite of this, results of the Maze map state the opposite. Trial 3, 6 and 9 have higher distance values and lower collisions which means that FPS bots were able to learn a navigation strategy a lot better in a maze-like layout than in an open area (Arena map). During replay, agents getting stuck occurred more often in the Arena map as well which might be another reason for this result.

Not surprisingly, FPS bots in referenced work could also slightly increase their absolute distance traveled and achieved a decent strategy [4]. Unfortunately, not theirs and not this paper's results seem more effective than an A* algorithm for navigation.

The following condition is overall counts of items picked up (by one agent).

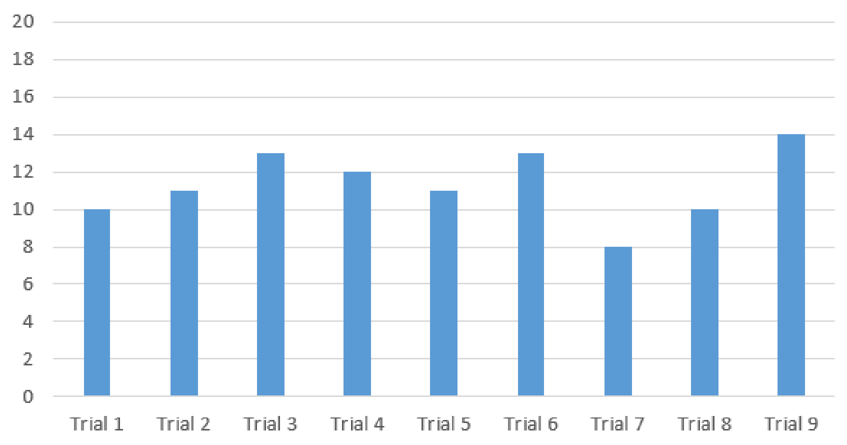

Fig. 9. Items collected of Arena map

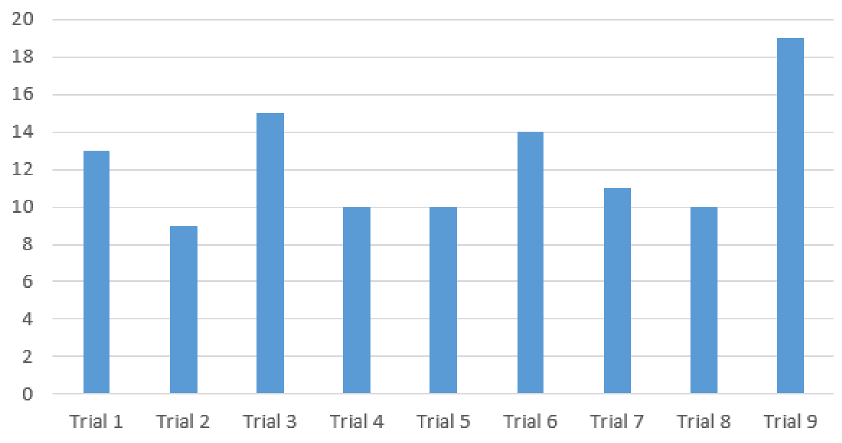

Fig. 10. Items collected of Maze map

Results of overall items collected are shown by Fig. 9. for the Arena map and Fig. 10. for the Maze map.

It is clear that most items picked up happened in trial 9 when γ was 0.8 and λ was 0.8. There is also a correspondence between collision counts, distances travelled and item-collection: with a higher λ (0.8, trials 3, 6, 9), agents could gather more items as well as travel farther. This aspect is significant in terms of a good navigation strategy and it seems like agents were able to develop such a thing through their training.

Other research papers have a similar outcome, that learning agents were able to collect more items when they travelled more and collided with obstacles more [4]. Referenced works also stated that the highest item-collection happened with both γ and λ being 0.8.

2.7.2 *Results of Combat Controller*

Following is the training of the Combat Controller of the FPS agents. Results are being observed by counts of kills and deaths.

Compared to related work [4], aggressive behaviour was achieved for FPS bots with γ set to 0.4 and λ to 0.8, and γ set to 0.8 and λ to 0.8. Stealth behaviour was observed when γ was 0.4 and λ 0.4. Another paper [7] stated that although quality of performance of their learning bots improved, agents could not win over state-machine bots but achieved an almost identical level.

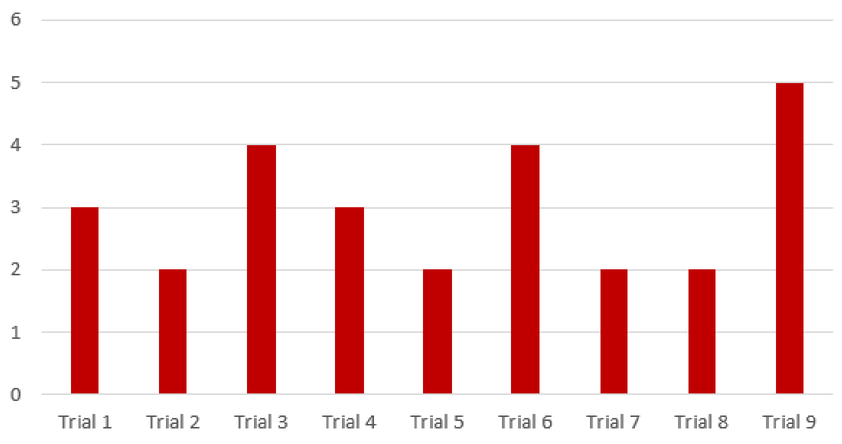

Fig. 11. Counts of kills / deaths of Combat map

Fig. 11. illustrates the overall numbers of kills and deaths that occurred during training the Combat Controller.

Corresponding to the above mentioned referenced paper's results, test runs show that agents engaged in plenty of combat and achieved the highest kill-death count when λ had a higher value (0.8) – trials 3, 6 and 9 again. After observing agents through replay, FPS bots of this training were able to develop aggressive strategy during these trials. With a λ of 0.0 or 0.4 (trials 2, 5, 7 and 8), a lower kill-death count can be seen which implies that agents rather explored the map than looked for a fight. This kind of behaviour is similar to a stealth style combat.

There was a phenomenon in both navigation and combat training which occurred quite often, and it was also discussed in related research. During the training process, these happenings were rare but the replay period and its statistics showed that agents got stuck between two alternate states quite commonly where the discrete output values of the table were turn left and turn right actions. Even enabling a small rate of learning during replay could not solve this problem completely.

The test runs that this paper concluded also had experiences with the problem stated above. Especially during replay, agents seemed to get stuck between two states which unfortunately altered results and statistics of the tests.

At the moment, there is still no solution to completely erase this phenomenon.

2.7.3 *Results of Ultimate Training*

Using these previously trained Navigation and Combat Controllers, the proposed method – analysis of agents' local environments – was implemented and experiments were carried out of the ultimate training process on the Ultimate map. The table of trials and parameters can be found in the previous section.

In this environment, all actions are executable – items are placed throughout the map layout and shooting is allowed for the agents. Which implies that all previously stated evaluation standards need to be examined such as counts of collision, distance travelled, items collected and counts of kills and deaths. Also, the proposed method suggests that according to its analysing process on agents' local environments, a decision is made on which controller (behaviour) to choose: Navigation or Combat Controller (stealth or aggressive style).

The statistics of this selection is shown below.

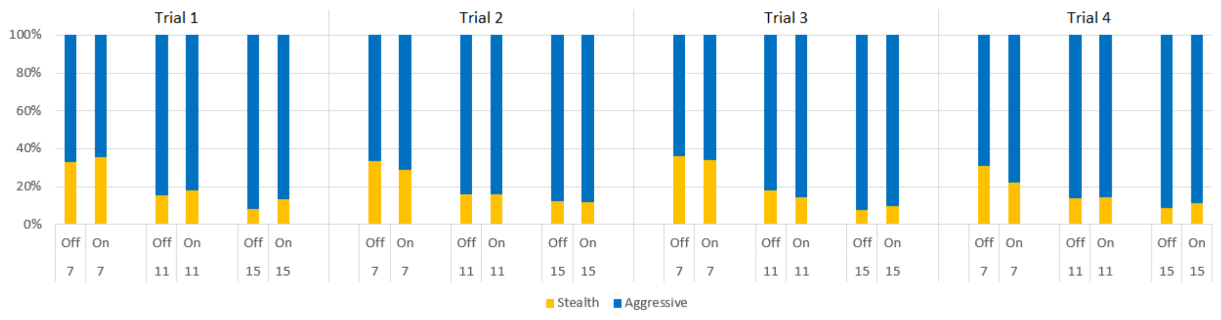

Fig. 12. Percentages of chosen behaviour of Ultimate map

The values of Fig. 12. demonstrate how often agents have chosen stealth or aggressive behaviour based on analysis of their current surroundings and situation. Trial 2 had 0.4 for γ and 0.4 for λ meaning a stealth Combat Controller, while trials 3 (γ was 0.4, λ was 0.8) and 4 (γ was 0.8, λ was 0.8) had an aggressive Combat Controller. The following factors are varying during the trials: size of agents' local areas (7 by 7, 11 by 11 or 15 by 15 metres), and agents' training process turned on or off.

It is shown that as the local environment's size grows from 7 to either 11 or 15 metres, the aggressive behaviour (Combat Controller) is more likely chosen by the FPS bots in both cases whether training is on or off. Although for almost all cases, usage of both controllers may seem slightly more balanced when agents are still in learning mode. "Balanced" means that there is no overusing one behaviour (for example, 95% aggressive and 5% stealth-style usage is not balanced) but applying both combat styles at a fair amount of rate during training and replay.

The overall goal of the proposed method was to create adaptive learning FPS bots that can choose a behaviour depending on their close environment of the map layout, in other words, to have a choice which relies upon the map itself. This is why the data displayed above is highly based on the current environment. During these test runs, the Ultimate map was used whose layout has more (smaller but plenty) open areas and is not so dense with obstacles. Therefore, as observations have proven as well, agents' would rather decide on the aggressive style at most times during training.

In conclusion, agents should be allowed to stay in training with a smaller rate, due to the fact that it is considered more optimal to make good use of both Navigation and Combat Controllers and have a healthy balance in the long run.

Next, all the general conditions of a well-working system of FPS bots is presented.

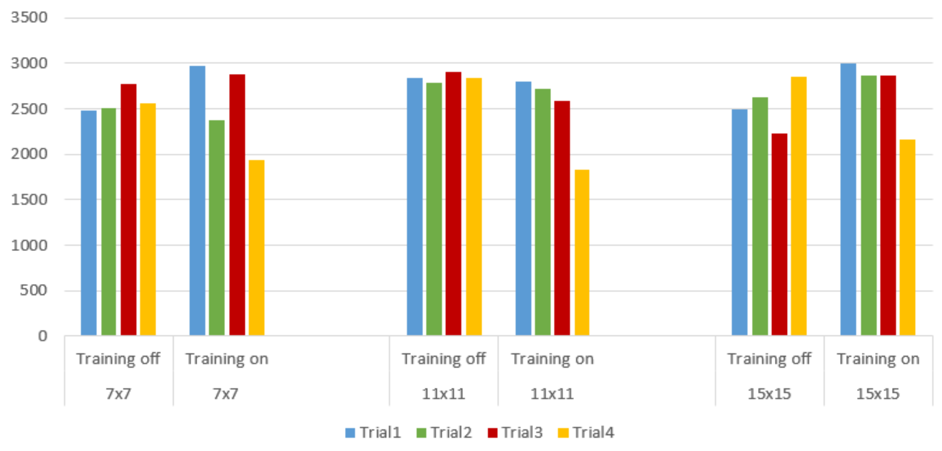

Fig. 13. Collision counts of Ultimate map

Fig. 13. observes the counts of collisions of the agents during ultimate training.

In some cases, a fairly low collision count could be achieved. For example trial 4's parameters (γ and λ were 0.8), training turned on and a local environment of 7 or 11 metres had the lowest values in this aspect. On the other hand, the highest collision counts can be seen when γ and λ were both 0.0 (trial 1), training of agents is turned off and local environment size is either 7 or 15 metres.

There seems to be no correlation between the size of local environments and how much the agents collided with obstacles. In addition, collision counts are still more related to values of γ and λ as explained in the section of training the Navigation Controller. Although, results are slightly lower and closer to a satisfactory level when agents stay in training during the ultimate phase.

Observing trials 1 and 2, their counts of collisions increased for most cases and they decreased for trials 3 and 4 – during previous experiments, the first set of parameters created a Combat Controller with stealth strategy and the second an aggressive one. This implies another correspondence between γ and λ values (combat behaviour) and collisions of the ultimate training.

A potential reason for not so optimal counts of collisions is that agents chose the Combat Controller to use most of the time, as seen on Fig. 12. They could engage in combat more, thus priorities of FPS bots did not include minimizing collisions anymore. Instead they were focusing on achieving more hits and kills and their own survival.

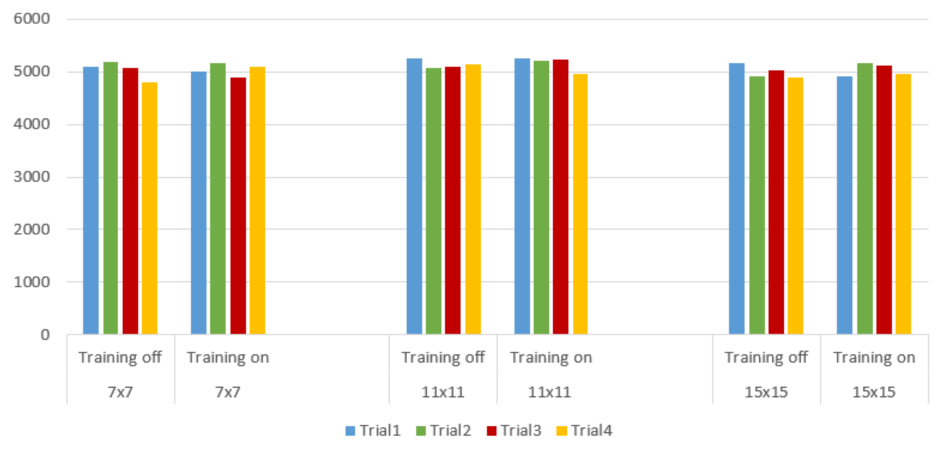

Fig. 14. Distances travelled on Ultimate map

Absolute distances travelled by agents on the Ultimate map are shown on Fig. 14.

Overall distance values seem much higher on this graph, ~1.5 times more than those of the simple Navigation Controller (previous experiments). Which concludes that when learning agents included local environment analysis in their training, they were able to explore the map even more.

Related work [4] stated that FPS bots usually sacrifice reaching a minimal count of collision and would rather choose travelling through the environment. The same observation can be found in this paper's research too. Trials 1 and 2 had slightly increased collision counts while their absolute distance values were corresponding to that. Trials 3 and 4 did not travel as much, although the difference is quite small. Therefore, another correlation can be concluded between γ and λ (trial parameters and combat behaviours) and agents' overall distance values.

All the values of Fig. 14. are quite close, meaning that there is no significant change through modifying simulation parameters. Although a few peaks can be highlighted: for one, during trial 1 (γ being 0.4 and λ being 0.0) and agents' training turned off, the highest distance values can be seen. Which corresponds to a higher percentage of stealth behaviour (Combat Controller) chosen by agents (almost 40% when local area size is 7) than with other settings. The lowest absolute distances were at trial 4 (γ being 0.8 and λ being 0.8) with all possible parameter settings, mainly due to the fact that FPS bots preferred an aggressive style strategy (higher percentage) in these cases, which refers to agents engaging more in combat than prioritizing exploration through the environment.

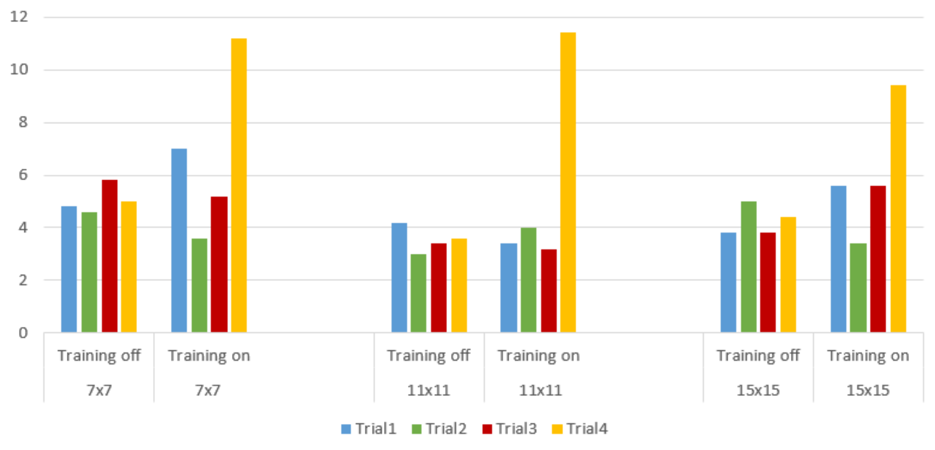

Fig. 15. Items collected of Ultimate map

Fig. 15. describes the numbers of picked up items during the ultimate training.

There seems to be no clear pattern or converging in the data of this graph. In spite of this, cases that had agents still in training had slightly more items collected than those with a turned off training.

Surprisingly high peaks are shown at test runs with trial 4's parameters (γ being 0.8 and λ being 0.8), the training turned on and for all local environment sizes. These cases seemingly outperform every other trial. Even though these settings grow into an aggressive styled combat behaviour where item-collecting is not high priority, agents were able to develop a new and improved strategy which is mainly about agents realizing that they can survive in combat for a longer time if they manage to collect more items in order to heal themselves. This was also observed through trainings and replays.

In addition, the previously stated new strategy might be the most effective in an environment of ultimate training that this paper implemented and investigated, for such a complex set of tasks of navigation and combat. Unfortunately, only these parameters could achieve such a behaviour, whereas other trials with different settings performed on an average level.

Also, trial 1 with agents' training turned on and a local environment size of 7 might be the closest to a simple navigation controller in training, due to it reaching the second best number of items collected, right after trial 4.

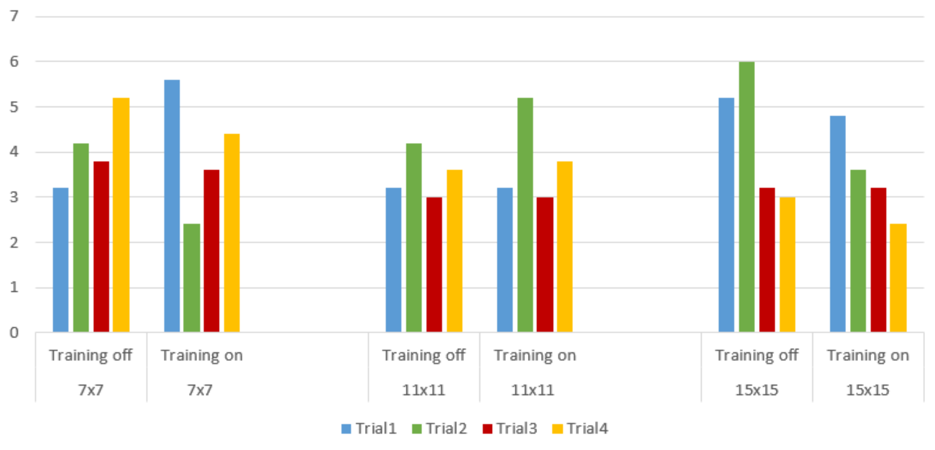

Fig. 16. Numbers of kills / deaths of Ultimate map

The numbers of kills and deaths of FPS bots that happened during ultimate training is displayed on Fig. 16.

This evaluation shows no clear convergence as well. It can be observed that the counts of kills and deaths of agents in higher on average during trials 1 and 2, due to FPS bots having a stealth combat strategy trained previously and the inability to collect enough items for survival. Stealth behaviour actually refers to less engaging in combat with enemies and more running away (or exploring the map) which is expressed by the graph above.

The previous figure stated that agents with trial 4's parameters were able to collect the most items during test runs which corresponds to their kill-death count being a lower to some extent on Fig. 16. due to all the healing they acquired.

Unfortunately, in cases of trials 1 and 2 (stealth behaviour) there is no pattern to be found, although trials 3 and 4 (aggressive behaviour) have a slight converging when considering the size of agents' local environments: as this size grows, less kills and deaths seem to occur. Therefore, it appears that the FPS bots were able to develop stealth and aggressive combat styles, even though they could only reach an average level on overall performance and quality of game-play. Comparing to the Hierarchical RL bots and Rule-Based RL bots of [7], these methods could not be outperformed by the learning agents of this paper's research.

The reason for this may be that the data abstraction method of local environment analysis does not completely, or accurately represent the information that the environment holds. In other words, this method still needs improving.

Two occurrences in particular might be affecting the above described test results. If the agents' training processes are turned off and only the previously trained Navigation and Combat Controllers are used, the phenomenon of an FPS got getting stuck between two alternate states rises again. As mentioned before, there is still no complete solution to this. In the case where agents are still learning during the ultimate training, an ε-greedy selection is included as well, just as in all other trainings. Which brings with itself a small opportunity for random actions executed. Therefore, the above shown results might not be a 100% accurate.

In conclusion, an ultimate training with analysis for agents' local environments had an average performance in terms of counts of collisions, absolute distance travelled, number of items collected and counts of kills and deaths, and the trials were dependent on values of γ and λ rather than the newly introduced method's parameters. In spite of this, a new strategy was developed where agents make use of the items' healing abilities.

3 CONCLUSION

A problem of reinforcement learning agents in first person shooter video games was the main theme in this paper. The necessary definitions for the environment, map layouts, states, actions, reward systems and applied algorithms were explained and described in detail as well as the referenced groundwork on reinforcement learning and adaptive shooting skills in first person shooter games. In a 2-dimension setting of a self-created project that incorporates simple and fundamental features of a commercial first person shooter video game, learning agents acting as FPS bots of the environment were trained in a thorough manner, using tabular Sarsa(λ) algorithm with eligibility traces altogether with ε-greedy selection.

The basic tasks of a first person shooting game were divided into two groups of actions to learn and strategies to develop. The Navigation Controller was responsible for the navigation, path-finding and item-collection operations, whereas the Combat Controller provided for the shooting skills and combat strategies of the learning FPS bots. The Arena map and Maze map acted as the required environment in order to properly and effectively train the Navigation Controller of the agents, while the Combat map was contributed to achieving a well-working and adaptive Combat Controller. Also, slight modifications were added to the referenced papers' work, such as a double representation system for the positions of agents and a small detail changed in the pseudocode of the Sarsa(λ) algorithm.

Experimentations were concluded on separate training processes of the Navigation and Combat Controllers, creating individual tables of state-action-pairs for each. Evaluation standards were defined and the general purpose of training the Navigation Controller involved minimizing the counts of collisions with obstacles of the environment, maximizing the absolute distances travelled by the agents and reaching numbers of collected items as high as possible. In terms of the Combat Controller, the main goal of FPS bots was to achieve fair amount of hits and kills of enemy bots and maintain survival by a minimal count of death.

Through these results and their discussion, trials of this paper came to a similar conclusion that related research did. Counts of collisions decreased when λ of the eligibility trace was higher both in the Arena and Maze maps. Also, colliding with walls occurred ~1.5 times more in the Maze map due to its high density of obstacles. Trials in the Arena map showed that absolute distances travelled by agents were also lower, corresponding to collisions when λ was set to a higher value. On the other hand, agents trained in the Maze map were able to travel more and collide less, with a higher λ, implying that they achieved a better navigation strategy through a maze-like environment rather than in a layout with more open areas. In case of the Arena map, FPS bots sacrificed their lower counts of collision in order to explore more. In addition to these, more items could be collected with a higher λ eligibility trace, relating to the other two evaluation standards. Although, agents of this paper were unable to reach a more optimal solution for navigation than an A* algorithm. In case of the Combat Controller, stealth behaviour could be accomplished with λ set to 0.0 or 0.4 which meant prioritizing the exploration of the map, and aggressive styled combat was produced by trials with λ being 0.8.

After successfully training Navigation and Combat Controllers for FPS bots, the proposed method of the paper was combined with said controllers. An analysis of the local environment of an agent was concluded and a preferred behaviour, stealth (Navigation Controller) or aggressive (Combat Controller), was chosen according to the outcome of the proposed method. This was termed ultimate training and its main purpose was to analyse an agent's nearby surroundings by types of cells, objects, enemies and decide on which controller to apply in the upcoming situation.

Four main trials were executed and these varied even further by changing the size of agents' local environments and switching between a continued training and turning it completely off.

The ultimate training uses the Ultimate map where all actions are executable, including collecting items with a healing ability and shooting weapons. Evaluation of results of the ultimate experiments were examined through aspects of counts of collisions, distances travelled, items collected and overall counts of kills and deaths, also percentages of preferred behaviours. In terms of chosen controllers, there is a clear correspondence between the size of agents' local environments growing and preferring the aggressive style even more. Although, usage of both controllers were slightly more balanced with a smaller local environment size of 7 and their training turned on which would be more optimal in the long run through all kinds of environmental layouts. The proposed method depends highly on the current layout but managed to achieve fair adaptiveness.

Based on counts of collisions of agents, with γ and λ both being high (0.8) the lowest colliding could be reached and the highest with γ and λ being small. There was no specific correlation between collisions and the size of agents' local environments, although collisions slightly decreased with the agents still in training. The agents choosing aggressive behaviour in most of the time was another reason for a not so optimal count of collisions, due to not prioritizing minimal colliding. In terms of absolute distance travelled, agents achieved ~1.5 times more distance than a simple Navigation Controller could. Also, where agents collided more with the environment, they could travel farther which is another correlation to γ and λ values, which corresponded to previously learned stealth and aggressive behaviours. Considering the numbers of items collected, agents still in training were able to pick up more, although no clear convergence was found in any of the aspects. In the case of trial 4 (γ being 0.8 and λ being 0.8), a surprisingly high peak was noticed which indicated a newly developed strategy of the agents. Through observation of trainings and replays, it was clear that agents were able to realize the healing ability of the items, thus collecting more and surviving longer during combat, and such a strategy proved to be quite effective and useful. Other sets of parameters finished at an average level, corresponding to simply trained Navigation and Combat Controller, whereas trial 4 managed to outperform them. The counts of kills and deaths showed that lower γ and λ values resulted in more killing and dying and higher values in less, which also relates to the agents developing stealth and aggressive styled strategies. In case of trial 4 with the new strategy, due to plenty of healing items were picked up, the number of kills and deaths decreased.

Experimentation of the ultimate training appeared to have no clear pattern or strict converging between performance and parameters of the proposed method (size of local environments and training turned on or off). Instead, it was corresponding to values of γ and λ as did the Navigation and Combat Controllers. Despite a new and interesting strategy of combining healing items and combat was discovered, most of the concluded trials performed on an average level and could not improve quality of their game-play comparing to those of the related groundwork.

4 Future Research

As mentioned before, the proposed method of ultimate training with local environmental analysis might not be the best-performing solution for combining navigation and combat tasks of learning FPS bots, due to the fact that it could not surpass the quality of experiments of related works. Training of which controller to use in a situation or state-machine bots still perform on a slightly higher level.

The data abstraction method of this paper is just one of many possibilities waiting to be investigated. Results showed that the information that this approach is subtracting from the environment might not be specific enough. Taking average of assigned values of an agent's surroundings also includes high and low peaks which can affect the result of the analysis. Other methods, such as taking median value, may turn out more successful in terms of performance of learning FPS bots.

Furthermore, after assigning a numerical value to the local area, decision-making based on a rule ($A < value < B$) could be substituted with training for which controller to choose by using the value of the agent's surroundings.

Another approach could be to apply the definition of states (both for Navigation and Combat controllers) as it was explained in this paper and at every step of the game, agents would make the decision on controllers considering what said states include – based on an agent's sensors for items, walls and enemies. Also, this method has both a rule-based approach and another through training.

In conclusion, the proposed method performed and showed game-play quality on an average level. Even though it could not exceed its competitions, endless variations of an ultimate training of FPS bots are yet to be investigated.

REFERENCES

[1] Stuart J. Russell and Peter Norvig, "Artificial Intelligence, A Modern Approach, Third Edition," published by Prentice Hall in 2009, ISBN-13: 978-0-13-604259-4, ISBN-10: 0-13-604259-7.

[2] Petar Kormushev, Sylvain Calinon and Darwin G. Caldwell, "Reinforcement Learning in Robotics: Applications and Real-World Challenges," in Robotics 2013, 2(3), pp. 122-148.

[3] Christopher Amato and Guy Shani, "High-level reinforcement learning in strategy games," in AAMAS '10, volume 1, pp. 75–82, 2010.

[4] Michelle McPartland and Marcus Gallagher, "Learning to be a Bot: Reinforcement Learning in Shooter Games," Artificial Intelligence Interactive Digital Entertainment, Stanford, CA, 2008.

[5] J.H. Lee, S.Y. Oh and D.H. Choi, "TD Based Reinforcement Learning Using Neural Networks in Control Problems with Continuous Action Space," IEEE World Congress on Computational Intelligence, Anchorage, USA, 1998.

[6] J. Bradley and G. Hayes, "Group Utility Functions: Learning Equilibria Between Groups of Agents in Computer Games By Modifying the Reinforcement Signal," Congress on Evolutionary Computation, 2005.

[7] Michelle McPartland and Marcus Gallagher, "Reinforcement Learning in First Person Shooter Games," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 3, No. 1, 2011.

[8] Frank G. Glavin and Michael G. Madden, "Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 7, No. 2, 2015.

[9] John Loch and Satinder Singh, "Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes," ICML, pp. 323-331, 1998.