technologies

MDPI

Article

Reinforcement Learning as an Approach to Train Multiplayer First-Person Shooter Game Agents

Pedro Almeida [1,*], Vítor Carvalho [1,2,*] and Alberto Simões [1]

[1] 2Ai, School of Technology, Polytechnic University of Cávado and Ave, 4750-810 Barcelos, Portugal; asimoes@ipca.pt
[2] Algoritmi Research Centre, University of Minho, 4800-058 Guimaraes, Portugal
* Correspondence: a17564@alunos.ipca.pt (P.A.); vcarvalho@ipca.pt (V.C.)

Abstract: Artificial Intelligence bots are extensively used in multiplayer First-Person Shooter (FPS) games. By using Machine Learning techniques, we can improve their performance and bring them to human skill levels. In this work, we focused on comparing and combining two Reinforcement Learning training architectures, Curriculum Learning and Behaviour Cloning, applied to an FPS developed in the Unity Engine. We have created four teams of three agents each: one team for Curriculum Learning, one for Behaviour Cloning, and another two for two different methods of combining Curriculum Learning and Behaviour Cloning. After completing the training, each agent was matched to battle against another agent of a different team until each pairing had five wins or ten time-outs. In the end, results showed that the agents trained with Curriculum Learning achieved better performance than the ones trained with Behaviour Cloning by a matter of 23.67% more average victories in one case. In terms of the combination attempts, not only did the agents trained with both devised methods had problems during training, but they also achieved insufficient results in the battle, with an average of 0 wins.

Keywords: reinforcement learning; unity; first-person shooter games; curriculum learning; behaviour cloning

Citation: Almeida, P.; Carvalho, V.; Simões, A. Reinforcement Learning as an Approach to Train Multiplayer First-Person Shooter Game Agents. *Technologies* **2024**, *12*, 34. https://doi.org/10.3390/technologies12030034

Received: 16 January 2024
Revised: 25 February 2024
Accepted: 28 February 2024
Published: 5 March 2024

1. Introduction

Recently, many breakthroughs have been made in the creation of Artificial Intelligence (AI) agents powered by Reinforced Machine Learning [1]. New platforms have made Reinforcement Learning (RL) more accessible [2], new algorithms trained agents more capably, and new training architectures created new ways to train those agents.

Video games have always been used as a benchmark for testing new AI techniques, and as the years pass, more and more games are solved by AI algorithms, as is the case with the classic games of Go [3], Chess [4], and Shogi [4]. RL has proved to be developed enough to play 2D arcade games such as Breakout, Pong, and Space Invaders at a human level from scratch [5], but it is still being developed for facing complex 3D environments. However, it has been proven that it is possible to apply RL to more simple First-Person Shooter (FPS) games [1,6].

FPS games are one of the more popular genres for developing RL agents in these 3D environments due to its nature of placing the camera on the agent's perspective, thus facilitating the use of visual recognition, and more closely mimicking what a real-life robot would require when functioning in a real environment [1].

From a systematic review conducted before this work [7], it was identified that no previous research had been performed comparing the Curriculum Learning and Behaviour Cloning training architectures for the purpose of training agents for FPS games. It is unknown if the two can achieve significantly better results when placed together. We saw this as a research opportunity and decided to compare and combine these training

architectures. The aim of this research is to further research RL applied to FPS games; with that in mind and the previously stated problem, we have proposed the following research questions:

- Can Behaviour Cloning yield trained agents with better performance than Curriculum Learning in FPS games?
- Does combining Behaviour Cloning and Curriculum Learning bring better performance in the design of bots for FPS games?

The main objective is to compare the Curriculum Learning and Behaviour Cloning training architectures and see not only which one brings better results, but if combining the two is feasible and can bring better results than just using one of them. We are using Unity and the ML-agents toolkit as many other articles mostly use other platforms such as ViZDoom, PAGOMUT, or Deepmind Lab. With Unity being one of the most largely used platforms, this research becomes more easily reproducible and comparable to future ones.

The structure that is followed in this paper can be described as follows: Section 2 describes the background; Section 3 presents the state of the art of RL applied to FPS games; Section 4 presents the implementation of the work; the results are presented in Section 5; the discussion is presented in Section 6; and finally, the conclusion and future work are shown in Section 7.

2. Background

In this section, we will cover relevant topics that the reader must consider in order to understand the work, namely, from the genre of FPS to various Machine Learning (ML) topics as well as to the Unity engine and the ML-agents Toolkit.

2.1. First-Person Shooters (FPS)

FPS games are a sub-genre of action games that are played from the first-person point of view that usually involve one or more ranged weapons and allow the player to fully navigate a 3D environment. The major focus of games like these are usually combat, although they can also have narrative and puzzle elements to them. They allow the player to freely control their character's movement, aim, and shooting, often in fast-paced and intense scenarios [8].

Many of the games in this genre have a multiplayer component, where players can play against each other or against AI-controlled opponents in various formats such as duels, free-for-all, or team-based modes.

Games in this genre include Doom (Id Software, 1993), Counter-Strike (Valve, 2000), Halo (Bungie, 2001), and Call of Duty (Infinity Ward, 2003).

2.2. Machine Learning (ML)

The field of ML focuses on developing programs that learn how to perform a task, as opposed to the traditional approach of developing programs with hardcoded rules on how to perform a task. With ML techniques, a program can adapt to changes in its environment without needing manual changes [9].

A good example of how ML thrives is in problems that are too complex for traditional methods, such as the spam filter [10]. An ML program analyses words in emails flagged as spam, finding patterns and learning by itself how to identify future spam mail. If the spam filter was run through the traditional programming approach, the designers would have to update the program each time the spam mail changed patterns.

2.2.1. Neural Networks

A neural network's purpose is to simulate the mechanism of learning in biological organisms [11]. Nervous systems contain cells referred to as "Neurons", which are connected to one another through axons and dendrites, and these connections are referred to as synapses. The strength of the synapses changes in response to external stimulation, and these changes are how learning takes place in living organisms.

This process is simulated in artificial neural networks, which also contain "neurons", in the form of computation units [12]. These neurons are organised into three main types of layers: input, hidden, and output layers. Data are fed into the network through the input layer, and they propagate through the hidden layers where computations occur. The output layer then produces the network's predictions or results [13].

In modern times, neural networks are becoming more and more popular in multiple areas and many organisations are investing in them to solve their problems. Neural networks can be found in a variety of places, which include computing, science, engineering, medicine, environmental, agriculture, mining, technology, climate, business, arts, and nanotechnology, among others [11].

2.2.2. Deep Learning

A subfield of ML, Deep Learning refers to the use of artificial neural networks with multiple layers in their networks, which can better process high levels of raw inputs. These Deep Learning neural networks can be commonly found being used in modern uses of neural networks such as image processing programs like face recognition and image generation, smart assistants such as Siri/Alexa, suggestion algorithms, and many more. Most of these state-of-the-art programs require inputting large amounts of data into the neural network, and as such, they are classified as Deep Learning [14].

2.3. Reinforcement Learning (RL)

RL is a subfield of ML that focuses on teaching an agent to make sequential decisions in an environment to maximise its long-term rewards. It is inspired by how humans and animals learn through interactions with the world. RL places an agent in an environment, carrying sensors to check its state, and gives it a set of actions that it can perform, as seen in Figure 1. The agent then tries out those actions by trial-and-error, so that it can develop its control policy and maximise rewards based on its performed actions [15].

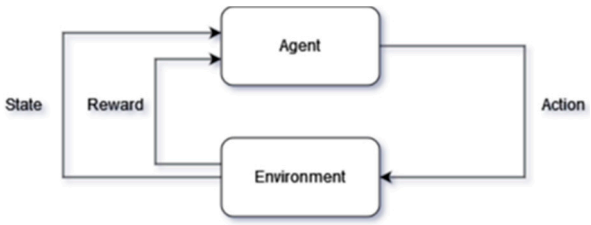

Figure 1. Agent interaction with the environment in RL [7].

RL is different from both Supervised and Unsupervised Learning, as it does not receive any pre-labelled data and it also is not trying to find a hidden structure, but instead working its way to maximising the reward value [16].

RL is made up of several components such as the agent, the environment, the actions, the policy, and the reward signal.

There is also a deep version of RL, called Deep Reinforcement Learning (DRL) [17].

Reinforcement Learning Components

When constructing an RL scenario, there are many components that one should keep in mind [16]:

- RL Agent

The agent is the entity that is being trained on the environment, with various training agents contributing to designing and refining the control policy. The agent monitors the state of the environment and performs actions.

- RL Environment

The environment is the space that the agent is in and can interact with and changes according to the agent's actions. It sends back feedback to the agent in the form of a reward signal.

- Actions of the RL Agent

The actions are the choices available to the agent—the agent selects actions based on the control policy, which influences the environment and generates a reward signal.

- RL Policy

The control policy is a map of the agent's action selection—it represents the behaviour or strategy of an agent in an environment. Moreover, it defines the mapping between states and actions, indicating what action the agent should take when it is in a particular state. The goal of RL is to find an optimal policy that maximises a notion of cumulative reward signal or value over time.

- RL Reward Signal

The reward signal is a numeric value that defines the goal for the agent. When the agent performs certain actions, reaches goals, or makes mistakes, the environment sends the agent a reward value, which can be positive, negative, or zero. The sole objective of the agent is, therefore, to maximise this value as much as possible over time during training.

2.4. Deep Reinforcement Learning (DRL)

DRL is achieved by combining Deep Learning techniques with RL. While RL considers the problem of an agent learning to make decisions by trial-and-error, DRL incorporates Deep Learning into the solution, which allows the input of large quantities of data, such as all the pixels in a frame, and still manages to decide which action to perform [17]. In Figure 2, we can see how the added Deep Neural Network (DNN) works with RL.

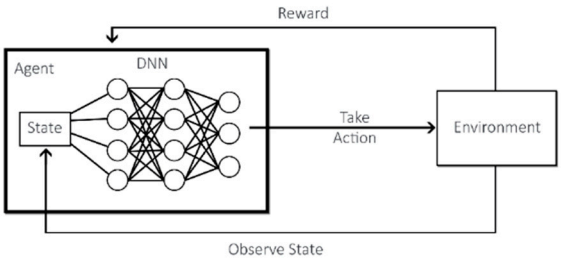

Figure 2. DRL agent interaction with the environment [7].

2.5. Training Architectures

A training architecture is how the designer trains their agents; agents can be trained alone versus traditional AI, against themselves, or even against or with other agents. They can also be trained using Curriculum Learning [18] and with Behaviour Cloning [19]. We will explain the various training architectures that are relevant for this work.

2.5.1. Single-Agent Reinforcement Learning

Single-Agent RL is a branch of ML that focuses on the interaction between an agent and its environment. In single-agent RL environments, there is one agent learning by interacting with either just the environment without AI or against traditional AI agents [20], as is the case in [5], where the agent learns to play many Atari arcade games.

2.5.2. Multi-Agent Reinforcement Learning

Multi-agent RL focuses on scenarios where numerous agents learn and interact in a shared environment. As shown in Figure 3, each agent is an autonomous entity that observes the environment, takes actions, and receives rewards based on its own actions and the actions of other agents. It can take the form of multiple scenarios, be cooperative with

each other or competitive with each other in a one-vs.−one scenario or a team-vs.-team scenario [20].

Figure 3. How multi-agent RL has multiple agents each controlling one player, acting independently from each other but still contributing to the same policy [7].

2.5.3. Self-Play

Self-play is a technique often used in RL that involves having RL agents playing against themselves to improve performance. As seen in Figure 4, a single agent acts as all players, learning from the outcomes of its own actions. Self-play has been successfully applied in [4], where researchers used this method to develop their Chess- and Shogi-playing AI.

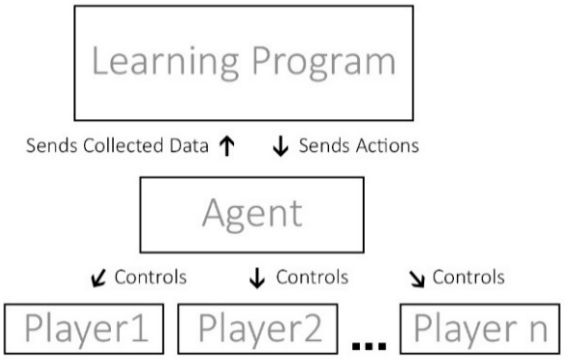

Figure 4. How self-play puts an agent in control of various players [7].

2.5.4. Behaviour Cloning

A form of imitation learning, Behaviour Cloning involves capturing the actions of a human performer and inputting them into a learning program. The program will then output rules that help agents reproduce the actions of the performer [19]. In video games, this usually means having a human player play in the designed environment, where their actions are recorded and then used to train the agent's policy. The more diverse the recording data, the better.

2.5.5. Curriculum Learning

Curriculum Learning architecture mimics human training by gradually increasing training difficulty. In Supervised Learning, this means increasing the complexity of training datasets; while in RL, it means increasing the complexity of the environment and task that the agent is required to perform [18].

In practical terms, this means that, for example, if one is training an agent on how to jump over a wall, they might want to start by having a wall with no height, and as the agent accumulates reward, the wall starts getting taller, as shown in Figure 5 [7,21,22]. At the beginning of the training, the agent has no prior knowledge of the task, so it starts exploring the environment to learn a policy and randomly tries out things. It will eventually reach

the goal, understand its objective, and progressively improve at jumping over higher and higher walls [21].

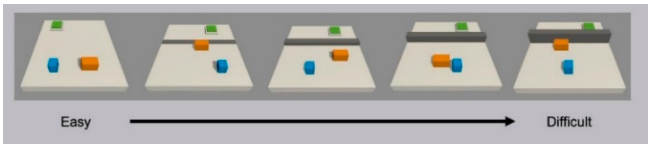

Figure 5. A Curriculum Learning environment where the agent must jump over a progressively higher wall; blue and orange objects are 2 agents, the grey object is the wall, and in green is the agents' target [7].

2.6. Unity

The Unity engine is a cross-platform multimedia platform made for creating 3D and 2D games, simulations, and other experiences [23,24]. Unlike other previously mentioned platforms, Unity is a standalone general platform, meaning that users can freely create their own environments with many more customised parameters than the alternatives. Unity contains its own physics engine and dedicated tools to create commercial 3D and 2D games, as well as a tool to create RL agents—the ML-agents toolkit [2]. Furthermore, the Unity's in-engine editor is easy and fast to use, allowing for quick prototyping and development of environments [24].

2.6.1. Unity's Features

Nvidia PhysX engine integration—Unity comes out of the box integrated with the PhysX physics software created by Nvidia allowing users to simulate complex state-of-the-art physics, mimicking real world interactions [24].

Simple to use, yet flexible—compared to alternative AI research platforms such as ViZDoom and DeepMind Lab [24], Unity's interface is simpler and easier to use. As seen in Figure 6, it allows the user to control all the environment's aspects either through its menus or programmatically. As Unity is a standalone engine meant for game development instead of a modified open-source engine such as ViZDoom, it allows much better control of its game environment [24].

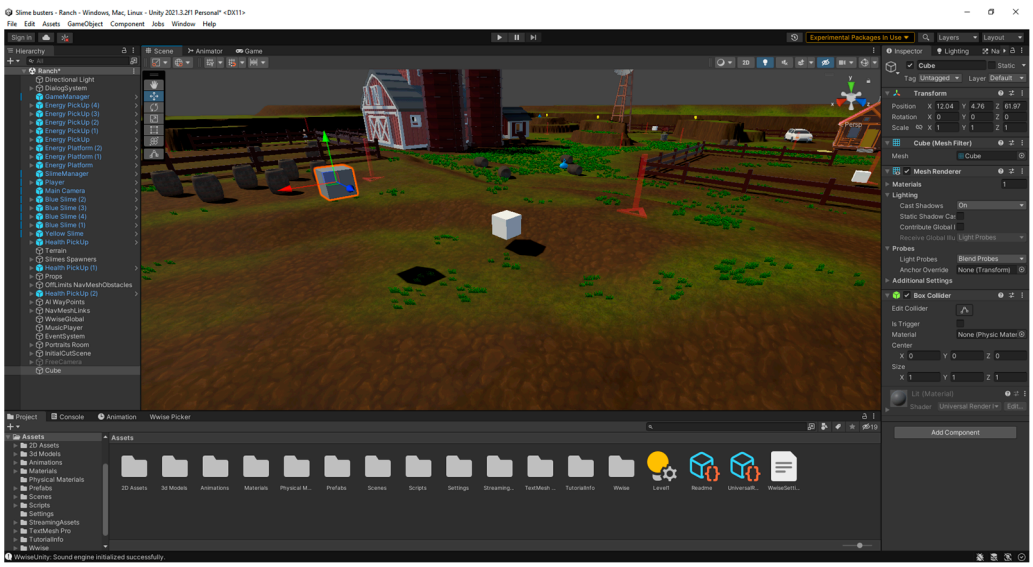

Figure 6. The Unity engine's interface, with a scene being worked on. On the right is the selected object's properties, on the left, the list of objects in the scene, and below, the list of assets in the whole project.

2.6.2. ML-Agents Toolkit

The ML-agents toolkit is an open-source project that allows researchers and developers to use environments created in the Unity engine as training grounds for ML agents by connecting via a Python API [2]. The toolkit features support training single-agent, multi-agent cooperative, and multi-agent competitive scenarios with the use of several RL algorithms such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) [2].

2.7. Proximal Policy Optimization Algorithm

The PPO is a model-free RL algorithm developed by OpenAI. It is used to train a policy function that maps the state of the environment to an action. It alternates between sampling data through environment interaction and optimises a surrogate objective function using stochastic gradient ascent. The PPO algorithm has some of the benefits of the Trust Region Policy Optimization (TRPO) algorithm but is much simpler to implement, more general, and has a better sample complexity [25].

PPO tends to have better generalisation properties compared to TRPO. TRPO is designed specifically for policy optimisation with continuous actions and imposes constraints on the policy update based on a trust region. PPO, on the other hand, is more flexible and can handle both continuous and discrete action spaces. PPO offers improved sample complexity compared to TRPO [25]. The surrogate objective approximates the expected reward based on the collected trajectories. Stochastic gradient ascent optimization is used to maximise this surrogate objective. The surrogate objective in PPO is a combination of two terms: a policy ratio term and a clipping term. The policy ratio compares the probabilities of the actions under the old and updated policies. It measures how much the new policy deviates from the old policy. Equation (1) shows the PPO's objective function that is not typically found in other algorithms [25]:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \quad (1)$$

where θ is the policy parameter, $_t$ denotes the empirical expectation over timesteps, r_t is the ratio of the probability under the new and old policies, respectively, $\hat{A}_t$ is the estimated advantage at time t, and ε is a hyperparameter, usually 0.1 or 0.2. In Algorithm 1, we see how a PPO algorithm that uses fixed-length trajectory segments works in pseudocode [25]:

Algorithm 1 PPO, Actor-Critic Style.

for iteration = 1, 2, . . . do
  for actor = 1, 2, . . ., N do
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L *wrt* θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

With each iteration of the algorithm, each of the N parallel actors collects T timesteps of data, then we design the surrogate loss on those NT timesteps of data and optimise it with minibatch SGD, for K epochs.

The PPO algorithm is one of the algorithms that can be used by Unity's ML-agents toolkit and sees much use in papers that use this toolkit such as [21,26].

3. State of the Art

Before starting this work, we first conducted a systematic review of RL applied to AI bots in FPS games [7], where we looked at numerous related articles. In this systematic review, we evaluated multiple algorithms, training methods, and platforms. In the end,

we thought about how AI performs against humans and how AI bots are bringing better gameplay and enjoyability to games and talked about future work possibilities.

In the systematic review, we concluded that training agents with the self-play or multi-agent training architecture proved much more efficient than using single-agent environments and that training designers should first incorporate Curriculum Learning or Behaviour Cloning into their training regime.

Furthermore, we can now look at older works and compare them to newer ones. We start with [1], which in 2011 applied a Sarsa (λ) RL algorithm to a purposely built FPS game. They used RL to make agents learn the tasks of navigation, item collection, and combat with results showing "that the RL algorithm was able to learn a satisfactory strategy for navigation control, but not to the quality of the industry standard pathfinding algorithm".

Again in 2011, Tastan, B. [27] examined the problem of teaching agents to play in a human-like manner in FPS games. They use pre-recorded data to make agents learn attack, exploration, and targeting policies in unreal tournament 2004 with POGAMUT.

Later, in 2017, Lample, G. [28] presented a model that combines both visual information input and game variable information input to make RL agents play FPS games to tackle 3D environments. By using ViZDoom with agents trained using a Deep Recurrent Q-Learning (DRQN) algorithm, they achieved agents that could win against humans by directly inputting information on whether an enemy was on screen or not.

We can already see from these three previous examples that older papers focus more on comparing and trying out algorithms, as opposed to more modern papers that focus more on training techniques.

One recent important paper that we found contains a competition in the ViZDoom platform with various participants competing with the goal of creating the best RL agent player [29]. They held two editions, one in 2016 and one in 2017: in 2016, victory went to a participant that used TensorFlow with a Direct Future Prediction (DFP) algorithm; in 2017, the victor also used TensorFlow but this time with a A3C algorithm and Behaviour Cloning.

Another great example are the experiments conducted by DeepMind in the DeepMind lab platform [30], where they used a multi-agent scenario and trained their agents for 450 k games with their inhouse developed For the Win (FTW) algorithm. The results from this research showed that their agents could beat strong human opponents at around the 200 k games mark.

A Q-Learning algorithm with a Naïve Bayes (QNBB) approach was used to train a RL state machine in playing as a simple enemy in a custom-made game [31]. This agent could learn from previous games, called "stages". They then conducted a survey where twenty participants played five games each for four stages of the agents, where the enemy is randomly the Q-Learning agent or a Greedy-Like Behaviour (GLB) algorithm agent. Participants were asked to rate both agents per stage in believability, overall game difficulty, and level of playability. Results showed that while the first stage of the QNBB was rated worse than the GLB one, subsequent stages were always the opposite and that the QNBB algorithm always got better with more experience.

Looking now at works performed using Unity, we start with one that used TensorFlow connected to Unity via an API [26]. They used a PPO algorithm to train RL agents to play a team-based survival shooter, where the map was a randomly generated maze. The objective was to train agents that could navigate the maze while dealing with enemy agents. The final results showed that continuous action spaces with no visual observations and no recuring neural network was the best option. The agents that incorporated visual observations had problems recognising their enemies and performed erratically.

Finally, we have two cases that use the ML-agents toolkit in Unity. The first case in [32] used the ML-agents toolkit with the objective of creating RL agents that can participate in competitive FPS matches by training them with Behaviour Cloning. To get results, they conducted a survey with eighteen participants who were asked to identify the ML agent in a multiplayer match. Out of the eighteen participants, 61% correctly guessed the agent and many of those noted that the agent had problems with navigation.

The second case is [21], where the ML-agents toolkit is used to compare one agent trained with Curriculum Learning to another agent trained without Curriculum Training. The curriculum agent not only was able to achieve higher rewards, but also achieve high rewards faster, with the final results showing that agents trained with Curriculum Learning have a significantly better performance than ones trained without it.

When comparing the older papers with the new ones, we can see a shift in research objectives. While the older papers focused more on comparing algorithms, the newer papers focus more on comparing training architectures.

One thing to note is that even in more modern papers, the resulting trained agents still do not perform much better than traditional agents. Such is the case in [31], where the ML agents did not score much higher than more traditional AI in terms of believability, difficulty, and playability. ML is still in an early phase, and the quality of the AI seems to correlate with the number of resources available to the creators; this means that as rule of thumb, we should expect most AI created by individuals to not perform as well as AI created by large teams.

4. Implementation

In this section, we will explain to the reader this work's implementation and the various design choices that were made.

By using the Unity engine, we hoped to create agents that receive game information such as the agent's health, agent's position, rotation, and their weapon's fire readiness state for comparison and then combine the Curriculum Learning and Behaviour Cloning training architectures. We used a simple environment with four obstacles to train each agent for one hundred million steps and in the end made the agents fight each other to draw results.

Development, training, and testing of this project were conducted over the course of a period of 4 months. We first conducted a testing period, where we tried to find values that we thought made sense to use in terms of training steps and hyperparameters, and then moved on to training. Because of the limited time we had to conduct this research, we could only spend 1 month in development, 1 month in testing, and 1 month in training and result taking.

4.1. Why Unity?

The Unity engine was considered because it allows for efficient environment creation. It allows us to design a much better Curriculum Learning environment for the agents to be trained on. It is also the platform with which we have the most experience working with, meaning that we were able to attain better results than if we were to work with other platforms, such as, for example, ViZDoom or DeepMind Lab.

For the ML solution within Unity, we used ML agents with the PPO algorithm, because it allows us to easily set up and train agents with much more speed than if we were to create our own ML scripts.

4.2. Why Use Game Information without Visual Information?

As many other works such as [20,28,30] have used visual information, we decided to try and use game information only, in order to create more lightweight agents. By only using game information, we can input much fewer values to the neural networks, making it much smaller and much lighter computationally. The use of these lighter networks allows us to have multiple agents in the environment at the same time on slower hardware even on Unity, which is an intensive game engine resource when compared to others such as ViZDoom and DeepMind lab.

4.3. Hardware Used for Training

The training and testing process was conducted on a personal laptop with the following specs:

- CPU: Intel core i7-7700HQ @ 2.80 GHz, Intel (Santa Clara, CA, USA);
- RAM: 16 GB;
- GPU: NVIDIA GeForce 1050 @ 4 GB VRAM, Nvidia (Santa Clara, CA, USA).

4.4. Training and Testing

To test which training architecture achieved the best results, we trained several agents to play a simple deathmatch FPS game, where the agents faced each other in square shaped one-vs.−one arenas with four obstacles. We trained four teams of three agents: one team of three agents trained only with Curriculum learning; one team of three agents trained with only Behaviour Cloning; and two other teams of three agents trained with a combination of Curriculum Learning and Behaviour Cloning—each using different methods of combination. Each agent was trained for one hundred million steps, a value that we found was feasible, as it translated into 2 days of training and was the minimum time required to produce agents that could reliably complete their task. We can see a resume of the agents in Table 1. When training of all the agents was completed, we pitched them against each other, making each one fight a match against the nine agents of the other teams five times each.

Table 1. Details of each training type.

Training Type	Nr. of Agents	Nr. of Steps	Description
Curriculum Learning	3	One hundred million	Used an 8-stage Curriculum Learning to teach the agent how to move, aim, and shoot.
Behaviour Cloning	3	One hundred million	Inputted previously recorded data into the agent and made them fight themselves (using Generative Adversarial Imitation Learning—GAIL [33]).
Combination Type 1	3	One hundred million	Inputted the recorded data and made the agents play the Curriculum Learning phases.
Combination type 2	3	One hundred million	Inputted different recorded data and made the agents play through the Curriculum Learning phases.

4.5. Agents

Agents receive input in the form of variable input only. We developed a rudimentary way to simulate sound and sight, so that the agents know the opponent's location from simulated sound or by rotating to face them.

The agents navigate the environment with the help of the sensors provided by the Unity ML-agents toolkit; these sensors raycast in multiple directions, telling the agent where obstacles are.

4.5.1. Agent Sound

While moving, spawning, and with every shooting action, the location of the agent or target will be transmitted to the other agents in the environment. The agents are inputted with the latest received "sound"; there is no limit on distance and these data are transmitted to the others regardless of position.

4.5.2. Agent Sight

Agents will raycast each step to the opponent; if the raycast succeeds, then the angle of the impact vector will be compared to the agent's forward vector. If the angle is less than 45°, then the enemy is being seen by the agent, giving it a set reward. If the angle is less than 15°, the reward is increased.

4.5.3. Weapon

The agent's weapon is a hit scan weapon with infinite ammo that can shoot once every five hundred milliseconds, meaning that after shooting, the player must wait five hundred

milliseconds before shooting it again. Each shot does twenty-five damage, meaning that the targets and agents, who all have one hundred health, are destroyed in four shots.

4.5.4. Inputs

The agents have a total of seven observation inputs, totalling at twelve float values:

- Agent Position (3 floats)—the agent's current 3D coordinates;
- Agent's Rotation in degrees (1 float)—the agent's current y Euler angle;
- Last seen Enemy Position (3 floats)—the 3D coordinates of the last seen enemy;
- Last non-self-made sound Position (3 floats)—the last sound heard that was not produced by this agent;
- Can shoot (1 float)—if weapon is ready to fire;
- Current Health difference to max health (1 float).

In addition to these, the agent receives inputs from the 3D ray perception sensors, which are sensors included in Unity's ML-agents toolkits—the agent's sensors have the following set settings, as seen in Table 2.

Table 2. Agent's sensor's settings.

Description	Value
Rays per direction	6
Max ray degrees	170
Sphere cast radius	0.5
Ray length (unity units)	20
Stacked raycasts	1
Start vertical offset	0
End vertical offset	0

We reached these values from the default ones after testing by adjusting the values until we found something that the agents could use to reliably complete their tasks.

4.5.5. Navigation

To improve the agent's learning of navigation, we added a sphere collider (Figure 7) to each agent's body that only collided with the obstacles and walls. When this collider collided with any of them, the agent would start receiving a negative reward. This method somewhat improved the agent's path finding, but they still had the tendency of getting stuck in the obstacles or walls.

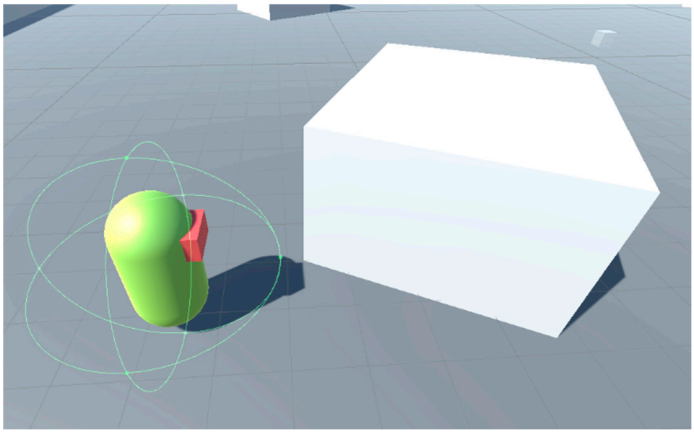

Figure 7. The collider (around the agent) (radius of 0.5 units) that tells the agent (in green) if they are near an obstacle (in white).

4.6. Environment

The environment in which the agents were trained is a very simple square environment surrounded by walls, with four cubes serving as obstacles, as seen in Figure 8. The environment was purposely made simple to shorten training time, as a more complex one requires the brain to learn more complex navigation, which would imply more steps.

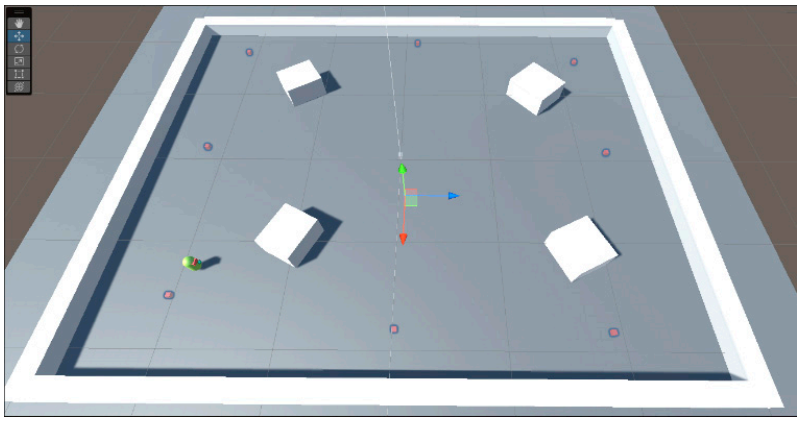

Figure 8. Environment in which the agents were trained on and the eight spawn points scatted around the arena.

During the Curriculum Learning program, the environment changes to suit the needs of the various phases. With each episode or game, the first agent is randomly placed in one of the eight "spawn point" (as seen in Figure 8), and then the enemy is placed in the one furthest away. These spawn points were purposely placed, so that the two participants will always spawn at the edge of the arena and most of the time with something hiding them from the opponent.

4.7. Parallel Training

To make use of each training "step" more efficiently, we have the option to train agents in parallel, this means having multiple environments running side by side with agents performing the same tasks, as seen in Figure 9. Each environment's agents contribute and draw decisions from the same brain. We used six environments in parallel training, meaning that with self-play, there were eight agents in the scene at each time while in the last phase of Curriculum Learning or during Behaviour Cloning. Due to hardware limitations, we could not increase the number of environments, as this is something that vastly increases the required processing power.

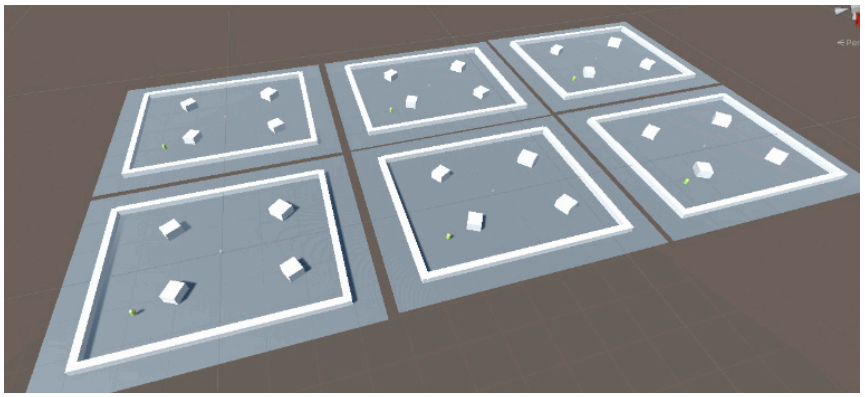

Figure 9. Example of parallel training with six copies of the same environment; each copy needs to be the exact same.

4.8. Training

The following sub-sections describe the multiple types of training that were conducted during this research.

4.8.1. Curriculum Learning Training

Curriculum Learning was split into eight phases. We distributed the tasks between them, making each phase harder than the previous; the objective was to teach the agents the various things they needed to learn such as enemy detection, aiming, and navigation.

During the eight phases, the agents find three types of targets, Immobile, Wandering, and Mobile, all of them having one hundred health. In the final phase of the curriculum, the agent fights against itself. The wandering target wanders around its spawn position, while the mobile target moves around the arena with a system of waypoints.

During phase 1 through 4, the arena does not have the obstacles shown in Figure 8; the eight phases of the curriculum are as follows:

- Phase one—Destroy the target directly in front:

In this phase, the agent spawns on the centre of the arena, the immobile target then placed right in front of them. The agent's objective is to destroy the target by shooting it. The objective of this phase is simply to teach the agent to look and shoot at the enemy.

- Phase two—Destroy the target that is slightly off-centred at the front of the agent:

During this phase, the agent spawns on the centre of the arena; the target is placed in front of them, but off-centred in a circle. The agent's objective is to destroy the target by shooting it. The objective of this phase to teach the agent to aim at an enemy in front of them.

- Phase three—Destroy the target that spawns randomly around the agent:

In this phase, the agent spawns in the centre of the arena. The target is placed randomly in one of the thirty-five preplaced points around the agent. The agent needs to look around for the target and then destroy it.

- Phase four—Destroy the target that spawns randomly around the agent and moves around:

During this phase, the agent will once again spawn at the centre of the arena. The target is placed randomly in one of the many preplaced points around the agent just like in phase three; but this time, the target is the wandering target, meaning it has a wandering behaviour that makes it move around. The objective of this phase is to teach the agent to aim at a moving enemy.

- Phase five—Find and destroy target in arena with obstacles while spawning in the middle:

The arena now has four cube obstacles as seen previously in Figure 8. The agent spawns in the centre of the arena, while a wandering target is placed randomly in one of the eight spawn points placed at the edge of the arena. This phase is the first step in teaching navigation to the agent by forcing it to move and search for the enemy.

- Phase six—Find and destroy target in the arena:

During this phase, the agent now spawns in one of the eight spawn points; after that, a wandering target is placed in the furthest spawn point. This phase requires the agent to destroy the target twice, respawning the target in a random spawn point after it is destroyed. This phase is the second step in teaching navigation to the agent, as well as extending the time that it stays in battle, forcing it to train how to search for an enemy that has respawned.

- Phase seven—Find and destroy a target that moves in the arena:

This phase is the exact same as phase six, except the target now has a moving behaviour, meaning that instead of wandering around, it moves towards the many AI waypoints placed around the arena. When the target reaches its destination, it calculates the new furthest waypoint and start moving again. The agent's objective is still to navigate the area, find the target, and destroy it by shooting it.

- Phase eight (final)—Agent fights against itself:

In this final phase, the agent spawns at random in one of the eight spawn points scattered around the edge of the arena; after that, another agent is spawned in the furthest available spawn point. The objective of these agents is to find and destroy each other by shooting. Each agent has one hundred health and must be destroyed twice for the episode to end. When an agent is destroyed, they are respawned in a randomly picked spawn point from the available eight. This phase's arena has its obstacles enabled as seen in Figure 8.

4.8.2. Behaviour Cloning Training

To train the agents using Behaviour Cloning, we first recorded a demonstration file by playing games until they achieved one hundred victories in Curriculum Learning's phase eight's environment. This file was then inputted into each one of the new Behaviour Cloning agents, which then fought against themselves, also in curriculum learning's phase eight's environment.

To train the agents with Behaviour Cloning, we used GAIL, one of the types of imitation learning available in ML agents.

The training setup of the Behaviour Cloning agents is the exact same as of phase eight of the Curriculum Learning training. The agents started with just the knowledge they developed from the demo, and each one of the three agents trained against their own clone for one hundred million steps each.

4.8.3. Combination of Curriculum Learning and Behaviour Cloning

To train the agents using a combination of Curriculum Learning and Behaviour Cloning, the agents were inputted the previously recorded demo file via configuration file and then started training from phase one of Curriculum Learning.

Two sets of teams of three agents were created as we tried to find an effective way to combine the two architectures. We called these teams the "Duel-type Combination" and "Basics-type Combination".

4.8.4. Duel-Type Combination

In this combination type, the demo file used in the Behaviour Cloning method was inputted to the agents via the configuration hyperparameters. A team of three duel-type agents were trained for one hundred million steps each.

4.8.5. Basics-Type Combination

For this attempt, we recorded a new demo file, this time with fifty episodes throughout the Curriculum Learning method's stage one through four. We then inputted this file to the agents via the configuration hyperparameters.

For both types, after the demo files were input, they started training from phase one of the Curriculum Learning, trying to clear each stage to reach the final eight stage. Each one of the three agents trained for one hundred million steps.

4.9. The Testing System

To test the agents against each other we made each one of them battle against all the different types of agents, meaning that each agent must fight the nine agents from the three opposing teams. To do this, we devised a matchmaking system that picks an agent, then makes them battle each agent of the opposing type until they have accumulated either five finished battles or ten time-outs against that agent. Note that two agents of same type will

never battle each other (example: agent trained in Curriculum Learning vs. agent trained in Curriculum Learning), as we want to compare the different training architectures.

The battles use the same exact mechanics as phase eight of the of the curriculum training program. Each participant spawns opposite each other and then must destroy their opponent two times to achieve victory. A finished battle is declared when one of the agents destroys its opponent twice.

Because the agents are not perfect players, we implemented a time-out system for each battle. After a battle starts, the agents have two minutes to destroy their opponent before the battle resets. After an agent destroys their opponent, the timer resets, and they once again have two minutes to continue the match before a time-out happens. This system is in place to prevent cases where the agents get stuck and cannot find their opponents during large amounts of time.

To get the results, we saved the following parameters from the battles per agent such as the total number of battles won, total number of battles lost, total number of successful battles, total number of battles that timed-out, total number of battles, total number of shots missed, total number of shots hit, total number of enemies destroyed, and total number of times destroyed.

We ran this testing environment twice, once with obstacles enabled and once without because we wanted to evaluate with and without the need for map navigation and see how the results differed. In the results section, we will present the results from these tests.

4.10. Metrics and Performance Measurement

Performance is measured through the parameters obtained in battle by the agents. Each parameter is a number that we use to measure the agent's performance, with wins, successes, hits, and kills being better the more there are and losses, time-outs, total games, misses, and deaths being better the less there are.

When we mention an agent having better or worse performance, we are talking about these metrics and numbers being better or worse when compared to their peers.

We picked these metrics as they are the metrics most found in commercial Deathmatch player-vs.-player FPS games to rank and measure player performance. While in team-based games, there is also the use of "score" due to a more varied range of actions, our research focuses specifically in creating agents for deathmatch games, and as such, we believe that using kill-to-death ratios and win percentages makes the most sense, as our goal is to compare the performance in deathmatch games of agents created with different training methods.

5. Results

In this section, we present the results obtained from the testing system.

5.1. Tables of Results with Obstacles

In Table 3, we find the recorded stats of the performance of the multiple agents during the testing with obstacles. In this test, the obstacles that we see in Figure 8 are enabled, meaning that the agents must find paths around them. The table is ordered by the agent's ID.

As we can see, the curriculum agents were the best performers, with the three of them achieving over 95% wins. The cloning agents won against the combination ones but still lost against the curriculum. Meanwhile the combination agents failed to ever achieve one win and ended up timing out when playing against each other. The kill-to-death ratio shows that the curriculum agents outperform others by far.

Table 3. Raw results of tests ran with obstacles enabled.

ID	Training Type	Team	Wins	Losses	Successes	Time Outs	Total Games	Misses	Hits	Kills	Deaths
1	Curriculum1	1	42	3	45	17	62	6946	394	93	9
2	Curriculum2	1	42	3	45	20	65	5752	366	89	9
3	Curriculum3	1	45	0	45	18	63	5786	370	90	7
4	Cloning1	2	25	10	35	78	113	4349	380	73	26
5	Cloning2	2	17	14	31	96	127	5594	339	58	32
6	Cloning3	2	9	15	24	102	126	4308	189	26	34
7	Combination1	3	0	26	26	84	110	9380	84	7	63
8	Combination2	3	0	20	20	90	110	9298	31	0	50
9	Combination3	3	0	25	25	88	113	8431	57	2	57
10	Phase1Combination1	4	0	19	19	98	117	5184	23	1	48
11	Phase1Combination2	4	0	25	25	101	126	354	1	0	54
12	Phase1Combination3	4	0	20	20	100	120	4101	13	0	50

5.2. Results without Obstacles

Now, we show in the following Table 4 the results of the tests when ran without obstacles, meaning that the agents did not need to find paths around the four objects placed in the arena.

Table 4. Raw results of tests ran without obstacles.

ID	Training Type	Team	Wins	Losses	Successes	Time Outs	Total Games	Misses	Hits	Kills	Deaths
1	Curriculum1	1	40	5	45	7	52	4789	371	88	16
2	Curriculum2	1	42	3	45	4	49	2768	348	86	15
3	Curriculum3	1	44	1	45	2	47	3465	360	88	12
4	Cloning1	2	31	11	42	45	87	3684	393	84	29
5	Cloning2	2	25	13	38	52	90	4211	369	67	30
6	Cloning3	2	29	13	42	69	111	4484	436	79	29
7	Combination1	3	1	24	25	74	99	9964	107	9	56
8	Combination2	3	0	22	22	95	117	11,263	65	3	50
9	Combination3	3	1	30	31	82	113	10,797	131	12	73
10	Phase1Combination1	4	0	30	30	69	99	5806	19	0	67
11	Phase1Combination2	4	0	31	31	56	87	689	12	0	71
12	Phase1Combination3	4	0	30	30	73	103	2848	6	0	68

The results without obstacles are more balanced—we see that the cloning agents performed better but still lost to the curriculum ones. There were less timed-out battles and more deaths in general, meaning that the agents were able to find their opponents more easily.

Overall, the performance is better, but the results remain the same, with the Curriculum Learning agents being the best, followed by the Behaviour Cloning ones, and then the combination ones.

5.3. Observations

In this section, we will describe the observations made during training and testing.

5.3.1. Curriculum Learning Agents

The curriculum agents were all able to complete the whole curriculum. One thing to note is that as they learned to be constantly shooting, which lead to them having many missed shots. Other than that, there were no notable problems or anomalies.

5.3.2. Behaviour Cloning Agents

The Behaviour Cloning agents did not constantly shoot like the Curriculum Learning team. These agents were much less aggressive than the curriculum ones and did not explore the map as much. As a result, they timed-out due to not being able to find their opponents multiple times.

5.3.3. Duel-Type Combination

The first combination team was never able to progress beyond phase one of the curriculum. They performed very poorly in both training and in battle when compared to the Curriculum Learning and Behaviour Cloning agents.

5.3.4. Basics-Type Combination

The second combination team was able to reach phase four of the curriculum but was unable to progress any further. Just like the first combination team, they performed very poorly in battle.

5.3.5. Overall

In general, all the teams had issues with pathfinding and often found themselves getting stuck on the obstacles. Another issue was that they were not aggressive enough and many times would just stay doing circles in the area where they spawned, trying to find the enemy.

The combination teams when pitched against each other had a very hard time finishing the match, as they either could not find each other or could not hit each other—this meant that most of their matches ended in time-outs.

6. Discussion

In this section, we will discuss the results and answer the research questions that were proposed.

6.1. Comparing Curriculum Learning and Behaviour Cloning

The first proposed research question was to compare the Curriculum Learning and Behaviour Cloning training architectures and see which one yielded the best results. To investigate this, we used as metrics the number of wins, kills, deaths and shots hit after we made all agents battle each other.

The agents trained with the Curriculum Learning method obtained the best results with and without obstacles: with a max of 45 obstacle arena wins and a max of 44 wins in the arena without obstacles, least time-outs, and an average kill-to-death ratio score in the obstacle arena of 11.02—they were the ones most capable of finding and destroying their opponents.

Meanwhile, the agents trained with Behaviour Cloning, although able to beat the combination agents, could not beat the Curriculum Learning ones, as the Curriculum Learning agents barely have any loses, maxing out at 5 for curriculum1 in the arena without obstacles. They also exhibit many time-outs, with a minimum of 45 for Cloning1 in the area without obstacles, meaning that they were not aggressive enough and failed to find their opponent.

The only place where we can say that the Behaviour Cloning agents are on par with the Curriculum Learning ones is the accuracy, most likely because unlike the Curriculum Learning agents which were constantly shooting, the Behaviour Cloning agents would try to shoot only when the enemy was in their field of vision thanks to the data from the demo.

Even though without obstacles, the agents could more easily find their enemies, this still did not change the outcome where the Curriculum Learning agents come out on top.

With these results, we reach the conclusion that the Curriculum Learning method that we developed achieves much better results than using Behaviour Cloning. We believe this is because the Behaviour Cloning agents become so focused on the provided demo data that they cannot develop new strategies during training; this led them to lose almost all battles against the Curriculum Learning agents who learned from zero on how to achieve the best results.

6.2. Combining Curriculum Learning and Behaviour Cloning

The second proposed question was about combining the training architectures of the Curriculum Learning and Behaviour Cloning and see whether this combination was viable and created a significant improvement in the agent's performance. Once again, the metrics are the exact same as in the previous question. For this, we tried two methods of combination.

The first method never managed to go beyond the first curriculum stage, failing to ever destroy the very first immobile target, and while the second method was looking promising during training, it stalled and failed to go beyond the fourth stage.

Both teams failed to achieve any meaningful results, having both failed to even reach the end of the Curriculum Learning program during training. Even with GAIL, the agents relied too much on the provided demos and failed to adapt to any difference in the environment.

During testing, they were repeatedly destroyed by their opponents, and when facing each other, they almost always ended the battle in a time-out. Looking at the concrete results, we see that these agents were a complete failure, with the second method not achieving one single win while the second method only managed to win three matches. Almost all their success matches ended in defeat, and even their hit percentages are abysmal compared to the Curriculum Learning and Behaviour Cloning agents, with the max hit percentage being by Phase1Combination2 in the arena without obstacles with 1.71% of all shots hit, while the minimum we see from the non-combination was 4.2% from the Cloning3 in the arena with obstacles.

With these results, we conclude that Curriculum Learning and Behaviour Cloning are incompatible, as the demonstration data interfere with the curriculum's progression system.

The explanation for these results seems to be that the agents are so focused on the data created by the demo that they fail to adapt to any environment that is not the exact same as the one from the demo. The second method saw some success in reaching the fourth stage of the curriculum because the environment was only very slightly different from the provided demo, but the moment the target started moving, the agent could no longer aim at it and failed to progress any further.

7. Conclusions and Future Work

In this study, we aimed to compare two training architectures of Machine Learning, Curriculum Learning and Behaviour Cloning, and then to see if it was viable to combine them. To do this, we used the Unity platform with the ML-agents toolkit and trained various agents to get results through testing in the form of battles between them. We started by creating a curriculum that the agents could progress through and then created teams of three agents to represent Curriculum Learning, Behaviour Cloning, and two methods of combining the two training architectures. The agents of each team trained for one hundred million steps each, and after all training was conducted, we set them up to matchmake and

fight against other agents of the other teams until each possible pairing had achieved five successful battles or ten time-outs against each possible opponent.

In the end, the program output two record files with various values that we could use to draw results from: one for battles with obstacles enabled and one for battles without obstacles enabled. These results showed us that the Curriculum Learning training architecture creates agents that perform much better than the Behaviour Cloning ones, achieving better win percentage, reliability, and kill-to-death ratio.

However, we also arrived at the conclusion that the two training architectures are not compatible, as the agents that were created from them struggled to even perform the Curriculum Learning course and had many problems performing in battle against the single architecture agents.

Considering further developments and just like we had previously stated in [7], we believe that a good path to take would be to incorporate Transfer Learning, multi-agent RL, and formal methods in the making of AI for videogames.

With the emergence of new AI technologies, new research possibilities also open. We believe that technologies such as generative AI can be well implemented in videogames to complement many old technologies, not just in the area of agent AI but also in the area of procedural generation.

Of course, in Reinforcement Learning, there are still many research opportunities. While we conducted this research without visual data, it is also worth conducting more testing on visual data vs. no visual data. Another path that researchers can take is to compare the Proximal Policy Optimization and Soft Actor-Critic algorithms, something that had not been extensively researched in terms of creating agents for FPS games. Furthermore, we plan to extend the study to other types of games. As a limitation to Reinforcement Learning approaches, considering its computational effort, it might limit the scalability to larger and more complex game environments and/or real-world scenarios.

However, we must never forget the ethical issues in conducting AI research; just as we had previously stated in [7], we should always ensure transparency in how our Machine Learning algorithms work, should always take care of our data handling to ensure data privacy, and, most important of all, should work towards creating AI that does not harm others, not just in the literal sense, but also in a way that AI is used to aid humans and not replace them.

Author Contributions: Conceptualization, P.A.; methodology, P.A.; software, P.A.; validation, P.A., V.C. and A.S.; formal analysis, P.A.; investigation, P.A.; resources, P.A.; data curation, P.A.; writing—original draft preparation, P.A.; writing—review and editing, P.A., V.C. and A.S.; visualization, P.A.; supervision, V.C. and A.S.; project administration, P.A.; funding acquisition, V.C. and A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was funded by national funds (PIDDAC), through the FCT—*Fundação para a Ciência e a Tecnologia* and FCT/MCTES under the scope of the projects UIDB/05549/2020 and UIDP/05549/2020.

Data Availability Statement: No new data were created or analysed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. McPartland, M.; Gallagher, M. Reinforcement Learning in First Person Shooter Games. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 43–56. [CrossRef]
2. Unity Team. The ML-Agent's Github Page. Available online: https://github.com/Unity-Technologies/ml-agents (accessed on 20 June 2023).
3. Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]

4. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
5. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
6. Glavin, F.; Madden, M. Learning to Shoot in First Person Shooter Games by Stabilizing Actions and Clustering Rewards for Reinforcement Learning. In Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG), Tainan, Taiwan, 31 August–2 September 2015; pp. 344–351. [CrossRef]
7. Almeida, P.; Carvalho, V.; Simões, A. Reinforcement Learning Applied to AI Bots in First-Person Shooters: A Systematic Review. *Algorithms* **2023**, *16*, 323. [CrossRef]
8. Elias, H. First person shooter: The subjective cyberspace. In Proceedings of the ISEA2008, Singapore, 25 July–3 August 2008.
9. Marsland, S. *Machine Learning: An Algorithmic Perspective*, 2nd ed.; CRC Press: Boca Raton, FL, USA, 2014.
10. Géron, A. *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2017.
11. Abiodun, O.; Jantan, A.; Omolara, A.; Dada, K.; Mohamed, N.; Arshad, H. State-of-the-art in artificial neural network applications: A survey. *Heliyon* **2018**, *4*, e00938. [CrossRef] [PubMed]
12. Aggarwal, C. *Neural Networks and Deep Learning: A Textbook*, 1st ed.; Springer: Cham, Switzerland, 2018. [CrossRef]
13. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
14. Ketkar, N.; Moolayil, J. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, 2nd ed.; Apress: Berkeley, CA, USA, 2021. [CrossRef]
15. Mitchel, T. *Machine Learning*; McGraw-Hill: New York, NY, USA, 1997.
16. Sutton, R.; Barto, A. *Reinforcement Learning—An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018.
17. François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M.; Pineau, J. An introduction to deep reinforcement learning. *Found. Trends®Mach. Learn.* **2018**, *11*, 219–354. [CrossRef]
18. Soviany, P.; Ionescu, R.; Rota, P.; Sebe, N. Curriculum Learning: A Survey. *arXiv* **2022**, arXiv:2101.10382. [CrossRef]
19. Sammut, C. Behavioral Cloning. In *Encyclopedia of Machine Learning and Data Mining*, 2nd ed.; Sammut, C., Webb, G., Eds.; Springer: Boston, MA, USA, 2017; pp. 120–124.
20. Serafim, P.; Nogueira, Y.; Vidal, C.; Neto, J. Evaluating competition in training of Deep Reinforcement Learning agents in First-Person Shooter games. In Proceedings of the 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Foz do Iguaçu, Brazil, 29 October–1 November 2018; pp. 117–11709. [CrossRef]
21. Adamsson, M. Curriculum Learning for Increasing the Performance of a Reinforcement Learning Agent in a Static First-Person Shooter Game. Master's Thesis, KTH University, Stockholm, Sweden, 2018.
22. Juliani, A. Introducing ML-Agents Toolkit v0.2: Curriculum Learning, New Environments, and More—Unity blog. 8 December 2018. Available online: https://blog.unity.com/community/introducing-ml-agents-v0-2-curriculum-learning-new-environments-and-more (accessed on 20 June 2023).
23. Unity Team. Unity Engine's Official Site. Available online: https://unity.com/ (accessed on 20 June 2023).
24. Juliani, A.; Berges, V.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. Unity: A General Platform for Intelligent Agents. *arXiv* **2020**, arXiv:1809.02627. [CrossRef]
25. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
26. Piergigli, D.; Ripamonti, L.; Maggiorini, D.; Gadia, D. Deep Reinforcement Learning to train agents in a multiplayer First Person Shooter some preliminary results. In Proceedings of the IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; pp. 1–8.
27. Tastan, B.; Sukthankar, G. Learning Policies for First Person Shooter Games Using Inverse Reinforcement Learning. *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.* **2011**, *7*, 85–90. [CrossRef]
28. Lample, G.; Chaplot, S. Playing FPS Games with Deep Reinforcement Learning. *Proc. AAAI Conf. Artif. Intell.* **2017**, *31*, 10827. [CrossRef]
29. Wydmuch, M.; Kempka, M.; Jasjiwski, W. ViZDoom Competitions Playing Doom from Pixels. *arXiv* **2018**, arXiv:1809.03470. [CrossRef]
30. Jaderberg, M.; Czarnecki, W.; Dunning, I.; Marris, L.; Lever, G.; Castaneda, A.; Beattie, C.; Rabinowitz, N.; Morcos, A.; Ruderman, A.; et al. Human-level performance in first-person multiplayer. *arXiv* **2018**, arXiv:1807.01281. [CrossRef]
31. Yilmaz, O.; Celikcan, U. Q-learning with Naïve Bayes Approach Towards More Engaging Game Agents. In Proceedings of the 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), Malatya, Turkey, 28–30 September 2018; pp. 1–6. [CrossRef]
32. Hagen, J. Agent Participation in First Person Shooter Games Using Reinforcement Learning and Behaviour Cloning. Master's Thesis, Breda University, Breda, The Netherlands, 2022. [CrossRef]
33. Ho, J.; Ermon, S. Generative Adversarial Imitation Learning. *arXiv* **2016**, arXiv:1606.03476. [CrossRef]