algorithms

MDPI

Review

Reinforcement Learning Applied to AI Bots in First-Person Shooters: A Systematic Review

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

[1] 2AI, School of Technology, Polytechnic Institute of Cávado and Ave, 4750 Barcelos, Portugal; vcarvalho@ipca.pt
[2] LASI—Associate Laboratory of Intelligent Systems, 4800 Guimarães, Portugal
* Correspondence: a17564@alunos.ipca.pt (P.A.); asimoes@ipca.pt (A.S.)

Abstract: Reinforcement Learning is one of the many machine learning paradigms. With no labelled data, it is concerned with balancing the exploration and exploitation of an environment with one or more agents present in it. Recently, many breakthroughs have been made in the creation of these agents for video game machine learning development, especially in first-person shooters with platforms such as ViZDoom, DeepMind Lab, and Unity's ML-Agents. In this paper, we review the state-of-the-art of creation of Reinforcement Learning agents for use in multiplayer deathmatch first-person shooters. We selected various platforms, frameworks, and training architectures from various papers and examined each of them, analysing their uses. We compared each platform and training architecture, and then concluded whether machine learning agents can now face off against humans and whether they make for better gameplay than traditional Artificial Intelligence. In the end, we thought about future research and what researchers should keep in mind when exploring and testing this area.

Keywords: reinforcement learning; deep learning; first-person shooter; bot; artificial intelligence

Citation: Almeida, P.; Carvalho, V.; Simões, A. Reinforcement Learning Applied to AI Bots in First-Person Shooters: A Systematic Review. *Algorithms* **2023**, *16*, 323. https://doi.org/10.3390/a16070323

Academic Editors: Frank Werner and Mircea-Bogdan Radac

Received: 20 April 2023
Revised: 16 June 2023
Accepted: 25 June 2023
Published: 30 June 2023

1. Introduction

Many recent breakthroughs have been made in the creation of Artificial Intelligence (AI) agents powered by Reinforced Learning (RL). New platforms have made RL more accessible, new algorithms have made trained agents more capable, and new training architectures have 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 [1], Chess [2], and Shogi [2]. RL has also proved to be developed enough to also play 2D arcade games such as Breakout, Pong, and Space Invaders at a human level [3], but it is still being developed when facing complex 3D environments. However, it has been proven that it is possible to apply RL to a simplified first-person shooter (FPS) game [4].

FPS are one of the more popular genres for developing RL agents in these 3D environments due to their 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.

1.1. First-Person Shooter

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

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 (1993), Half-life (1998), Halo (2001), and Call of Duty (2003), and this genre many times contains a multiplayer element where various human players can play against each other or AI bots.

1.2. Purpose of the Review

The purpose of this review is to investigate the current state of the art of how RL is used in the creation of FPS AI bots and look at the various platforms and methods that have been used in recent years to create RL AI. Previously, there have been reviews conducted in RL agents for video games in general [5], but not specifically on FPS. As there have been multiple FPS papers in past years, it seems appropriate to focus only on them instead of generalizing, as these games have specific mechanics.

In this review, four questions are proposed to be explored and answered:

- What platforms exist and what methods are being used to create RL AI bots in FPS games?
- Which methods are currently the most effective at creating AI bots for better experiences?
- Can these AI bots rival and beat human players?
- Can RL-based AI bots yield better gameplay and be more enjoyable than traditional AI bots?

The first question will be explored and answered during Section 4, where we will expose the various platforms and methods that have been used in the past five years to create RL AI bots. The other three questions will be answered in Section 5, where we will cover each one and draw conclusions from the gathered information.

1.3. Document Organization

The sections will be structured as follows: Section 2 will describe the background information necessary to understand the rest of the document. Section 3 will present the research methods. Section 4 will expose the various frameworks, algorithms, training architectures, and platforms in the form of the research results. Section 5 is the discussion chapter where we compare results and answer the proposed questions. Finally, Section 6 is where we conclude the review and look into future research and discuss potential ethical issues in the development of ML for FPS games.

2. Background

In this section, the theoretical considerations required to understand the main paper concepts are presented.

2.1. Machine Learning

The field of machine learning (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 input or output without the need for manual updates.

A good example of how ML thrives is in problems that are too complex for traditional methods, such as the spam filter [6]. An ML program analyses words in emails flagged as spam, finding patterns, and learning 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 its patterns.

There are various types of ML paradigms, separated into categories by whether they require human supervision or not, whether they can learn on the fly, and whether they work by comparing new data points to previous data points, or by detecting patterns in the training data to build a predictive model [6].

2.2. Reinforcement Learning

RL is a subfield of machine learning that focuses on teaching an agent to make sequential decisions in an environment to maximize its long-term rewards. It is inspired by how humans and animals learn through interaction 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 maximize rewards based on its performed actions [7].

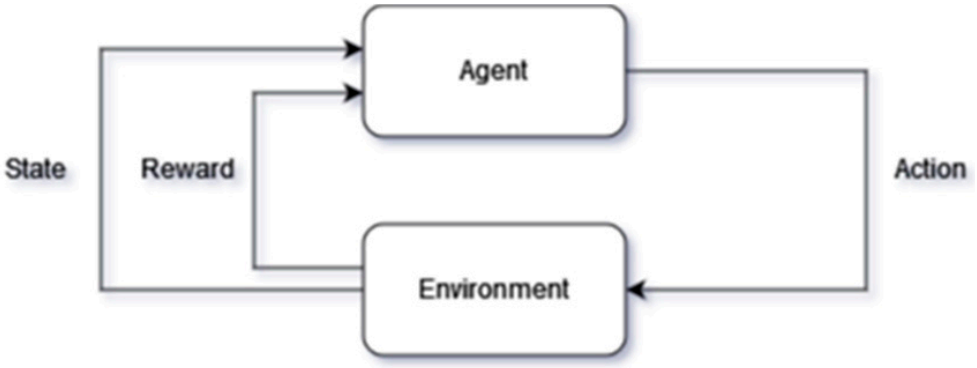

Figure 1. How an agent interacts with the environment in RL.

RL is different from both supervised and unsupervised learning, as it may not receive any pre-labelled data and it also is not trying to find a hidden structure, instead working its way to maximizing the reward value [8].

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.

2.3. Deep Reinforcement Learning

Deep Reinforcement Learning (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. In Figure 2, we can see how the added deep neural network works with RL.

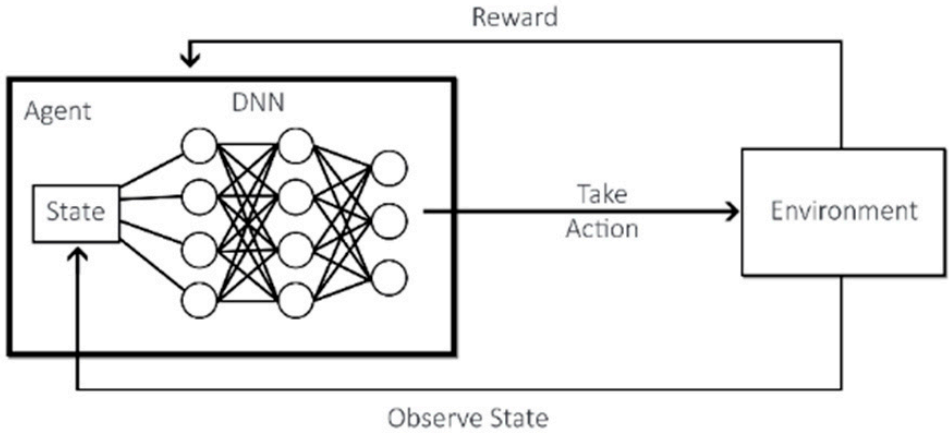

Figure 2. How the DRL agent interacts with the environment.

2.3.1. RL Agent

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

2.3.2. 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.

2.3.3. Actions

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.

2.3.4. 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. 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 maximizes a notion of cumulative reward signal or value over time.

2.3.5. 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 maximize this value as much as possible over time, during training [8].

3. Methods

In this section, we will look at the methods employed for searching for the literature used for analysing the RL state of art, starting with the query that was used. We end it with the metrics that are found and used in this review.

3.1. Literature Search

The query used for searching the papers was: "Reinforcement Learning" OR "DQN" OR "Deep Learning" AND "First Person Shooter" AND "bot", considering results only from 2018 to 2022.

The searches were recorded using the PRISMA format [9] (see Figure 3) and identified 300 records (after duplicates were removed) for screening, with 32 records read in full text and 13 retained for this review. During this process, two relevant results were locked out of access. The searches were conducted using the Publish or Perish software ver. 8.6.4198.8332 on Google Scholar and Scopus databases.

For this review, only papers focusing exclusively on RL applied specifically to Deathmatch FPS were considered. This means that, even if the work is about an FPS, it does not qualify if, for example, it is about making an agent that plays the campaign component of that game or playing a scenario where no enemies to destroy are found. The ultimate objective should be to create a "bot" that can be added to a deathmatch mode for human players to play against.

Papers that use older platforms such as POGAMUT have been excluded because these platforms lack critical features such as visual data and are thus considered outdated.

From the various papers, a diversity of platforms, algorithms, and methods were ascertained, and different surveys were extracted. Two of the thirteen papers contained surveys, although each one focused on different matters. Seven used ViZDoom, three used Unity's ML-Agents, one used DeepMind Lab, and two used just Unity with custom scripts. In Figure 4, we can see the distribution of platforms in a pie chart form.

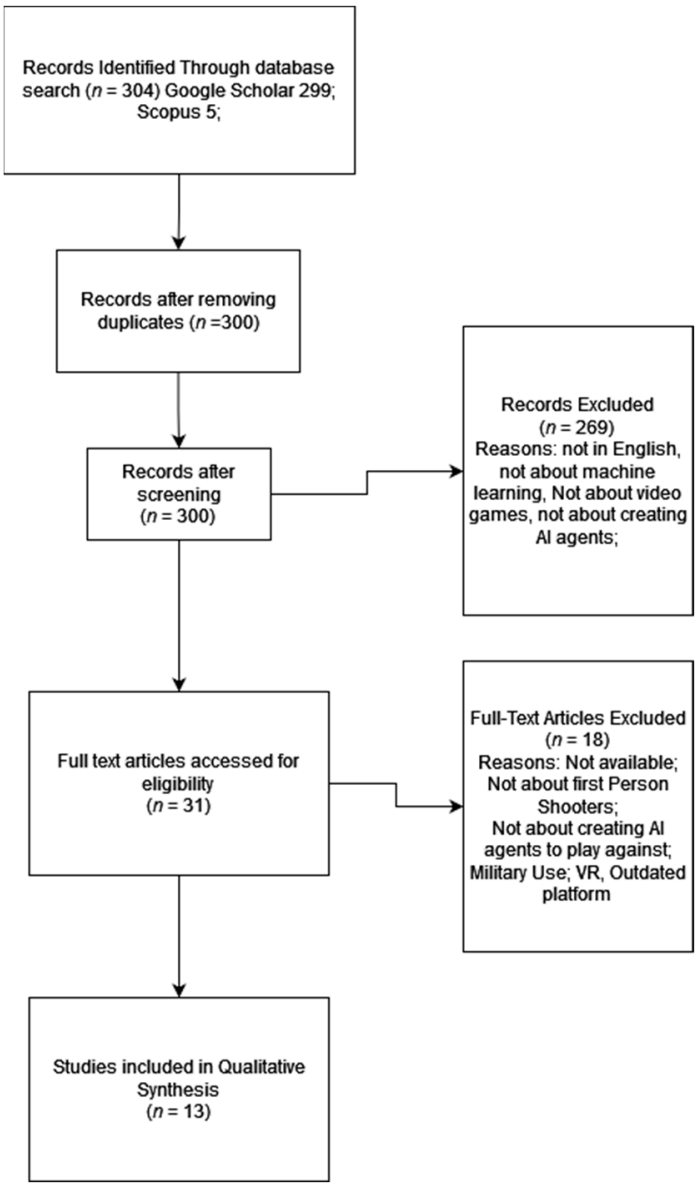

Figure 3. Prisma flow diagram.

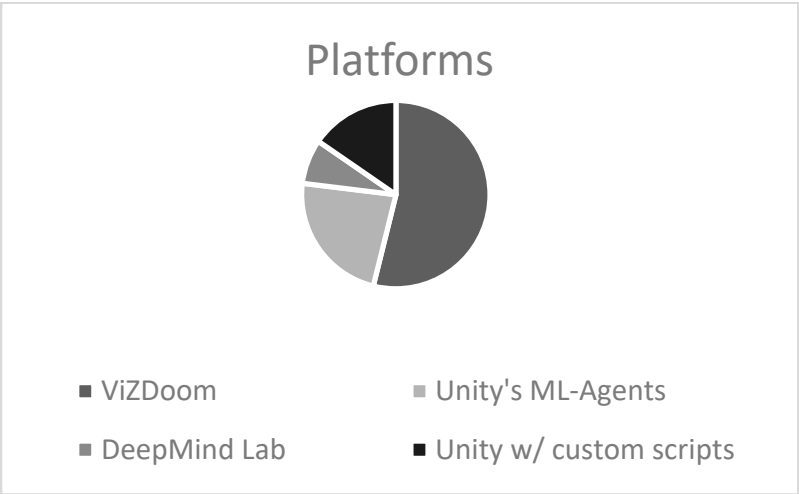

Figure 4. Distribution of used platforms.

During this research, various other sources were used to supplement research and better argue some positions.

3.2. Metrics

We now look at the metrics used for evaluating the performance of agents created for FPS games. Looking through the various papers, we found various metrics being used for comparations:

- Reward

The most common method of quantitative comparison is to use the obtained reward to compare two or more trained agents; we found find this used in [10–16]. It is indeed a reliable metric to use in a local environment; however, it leads to results that cannot be compared to anything outside of that paper as the reward will differ from project to project.

- Time

Time was also used, with [11] comparing the agent's time to train on top of the reward, stating that agents that train faster are superior.

- Game-based metrics

We believe that data such as the number of points, kills, deaths, and shots hit and missed are the most reliable method of metrics to train better agents, as this is the true purpose of the agents we train: to play a game. We found this used in [12,16–18].

4. Results

In this section, we will answer the first proposed question and talk about the various frameworks, algorithms, training architectures, and platforms that were identified in recent papers, stating facts and comparing items inside each category.

4.1. A Survey of Frameworks

In this section, we will expose the various frameworks and libraries that have been extracted from the collected data.

4.1.1. TensorFlow

TensorFlow is an open-source ML framework and library created by the Google Brain team and was initially released in 2015 [19]. TensorFlow contains neural network generating and training algorithms such as [20]:

- Convolutional Neural Networks (CNN), used in image classification and image recognition;
- Recurrent Neural Networks (RNN), used in natural language processing and time series;
- Generative Adversarial Networks (GAN), used in image generation;
- Transformer Networks, used in machine translation and text generation;
- Autoencoders, used in dimensionality reduction and anomaly detection [19].

TensorFlow was made to be flexible, efficient, extensible, and portable, able to be used on any device. Its main characteristic is the use of data flow graphs: it uses the computational model of a directed graph, where the nodes are functions and edges are numbers, matrices, or tensors. By splitting up calculations into smaller pieces, TensorFlow can distribute the load throughout multiple CPUs, GPUs, and other computational devices [21].

4.1.2. PyTorch

PyTorch is an open-source ML framework and library developed by Facebook, Inc. It was ported to Python from the Lua Torch library and provides seamless use of GPUs by utilizing PyTorch's CUDA backend and its DistributedDataParallel module to distribute the training process across multiple machines; it also provides a platform for deep learning that provides flexibility and speed [22]. PyTorch offers automatically generated computational

graphs with its autograd package, meaning a client can transform them during runtime, and allows the engineer to better control the memory use of the neural network as it develops [23]. However, PyTorch's most important feature is automatic differentiation. Automatic differentiation calculates the gradients of the inputs given a computational graph and can be performed in two modes: forward and reverse. Forward mode means that it calculates the gradients along with the result of the function, while reverse mode requires the evaluation of the function first, and then the calculation of the gradients starting from the output. This was what set PyTorch apart from TensorFlow and inspired changes in TensorFlow 2.0 [24]. Although not as popular as TensorFlow, PyTorch is still used in RL for FPS games, such as is the case in [17], where one of the contestants using PyTorch obtained second place in the 2017 competition, beating many of the TensorFlow users.

4.2. Algorithms

In this section, we will explain the various training algorithms that have been found in the selected papers and will look over their use cases.

4.2.1. Q-Learning

Introduced in 1989, Q-learning is an outdated Model-Free RL made to choose the best action for the given observation. Each possible action for each observation has its Q-value, which stands for the quality of the given act [10]. The Q-learning algorithm updates the Q-function at each time step using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big(r + \gamma * \max_{\alpha} Q(s_{t+1}, a) - Q(s_t, a_t)\Big) \quad (1)$$

where s is the current state, a is the chosen action, r is the observed reward, s_t is the next state, and a_t is the next action. α is the learning rate that controls how much the Q-value is updated and γ is the discount factor that discounts the future rewards. The algorithm continues until the Q-function converges to the optimal Q-function.

Q-learning works well when the state space is small or has a low dimensionality, as it can learn and update Q-values for each state–action pair. In addition, when the environment is simple and its dynamics are known, it can learn optimal policies when the transition probabilities and rewards are well-defined.

4.2.2. Deep Q-Network

The Deep Q-Network (DQN) is a variation of the Q-learning algorithm that uses a neural network to approximate the Q-function, instead of employing a value-based approach [11]. The training process involves updating the weights of the neural network based on the difference between the predicted and actual rewards obtained by taking certain actions. This allows the agent to improve its decision-making over time and learn optimal strategies for achieving its goals. The key feature of DQN is its use of an experience replay, which stores past experiences in memory and randomly selects them to train the neural network. This prevents it from becoming stuck in local optima and improves sample efficiency.

Unlike Q-learning, DQN is effective when the state space is large or high-dimensional, such as images or raw sensory inputs, and is well-suited for complex environments with unknown dynamics. However, DQNs can be sample inefficient, requiring extensive interactions with the environment to learn effectively. DQN also suffers from the curse of dimensionality, where the volume of the space increases so fast that the available data become sparse, meaning that it becomes challenging to obtain sufficient samples to accurately represent the underlying distribution of the data.

In [25], results showed that a Q-learning algorithm with a naïve Bayes approach improved the AI bot's "believability" against a Greedy-Like Behaviour algorithm. Instead of selecting actions deterministically based on Q-values, a Q-learning algorithm with a naive

Bayes approach can use the estimated probabilities to perform probabilistic action selection. This allows for a more diverse exploration of actions, reducing repetitive behaviours.

4.2.3. Double Deep Q-Network

The Double Deep Q-Network (DDQN) uses two different neural networks to train the model, with one network used for selecting actions and the other network for evaluating the value of those actions. This approach helps to reduce overestimation bias, which can occur when using a standard Q-learning algorithm.

The DDQN was compared to a DQN in [11], where it outperformed in all of ViZDoom's stock scenarios except for the "Defend the Centre" scenario, where both achieved the same average reward, but with DDQN reaching it 90 min faster.

4.2.4. Advantage Actor-Critic

The Advantage Actor-Critic (A2C) is a policy-based algorithm and, much like DDQN, uses two neural networks: the actor network and the critic network. The actor network takes the current state as input and outputs a probability distribution over possible actions. It is trained using policy gradient methods, such as the advantage function, to maximize the expected cumulative reward. By exploring the environment and receiving feedback, the actor network adjusts its parameters to improve the policy over time. The critic network in A2C learns the value function, which estimates the expected cumulative reward given a state or state–action pair. The critic network provides a measure of how good or valuable a particular state or state–action pair is. It helps the actor network by providing feedback on the quality of its actions. The critic network is trained using temporal difference learning, where it updates its parameters to minimize the difference between its estimated values and the actual observed rewards. The actor and critic networks are trained simultaneously to improve the learning efficiency and stability.

The A2C objective function can be defined as:

$$\begin{aligned} R'_t &= \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V_{\theta_v}(s_{t+k}), \\ A_{\theta,\theta_v}(s_t, a_t) &= R'_t - V_{\theta_v}(s_t), \\ J(\theta) &= \mathbb{E}_\pi[\sum_{t=0}^{\infty} A_{\theta,\theta_v}(s_t, a_t) \log \pi_\theta(a_t|s_t) + \beta H_\theta(\pi(s_t))]. \end{aligned} \tag{2}$$

θ_v are the parameters of the value network and $H_\theta(\pi(s_t))$ is an entropy term used to encourage exploration during the training process [12].

The A2C and the DQN algorithms were compared in [10] by using ViZDoom, with results showing that while DQN is better at exploiting the enemy's weaknesses, A2C is better at exploring different play strategies.

4.2.5. Actor-Critic Using Kronecker-Factored Trust Region

The Actor-Critic using Kronecker-Factored Trust Region (ACKTR) is an extension of the Advantage Actor-Critic (A2C) algorithm that uses the Kronecker-Factored Trust Region (KFTR) algorithm to optimize both the actor and critic neural networks. By optimizing both the actor and critic using K-FAC, ACKTR can better handle high-dimensional state spaces and complex action spaces. For the actor, the following policy distribution is used to define the Fisher matrix:

$$F = \mathbb{E}_{p(r)}\left[\nabla \log \pi(a_t|s_t) \nabla \log \pi(a_t|s_t)^T\right] \tag{3}$$

For the critic, the output of the critic v is defined as a Gaussian distribution and is defined as the Fisher matrix concerning this Gaussian output distribution.

$$p(v|s_t) \sim \mathcal{N}\left(v; V(s_t), \sigma^2\right) \tag{4}$$

When actor and critic share lower-layer representations, the joint distribution is defined as $p(a, v|s)$, and K-FAC is applied to approximate the Fisher matrix to perform updates simultaneously.

$$\begin{gathered} p(a, v|s) = \pi(a|s)p(v|s), \\ F = \mathbb{E}_{p(\tau)}\left[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^T\right] \end{gathered} \tag{5}$$

Furthermore, ACKTR adopts the trust region formulation of K-FAC, choosing the effective step size η. The natural gradient is performed with the following updates.

$$\begin{gathered} \eta = \min\left(\eta_{max}, \sqrt{\frac{2\delta}{\Delta\theta^T \hat{F} \Delta\theta}}\right), \\ \theta \leftarrow \theta - \eta F^{-1} \nabla_\theta L \end{gathered} \tag{6}$$

where η_{max} is the learning rate and δ is the trust region radius [12].

The ACKTR algorithm and the A2C algorithm in a Single-Agent RL environment were compared by [12], with results showing that ACKTR agents outperformed the A2C ones by a significant margin, while also being more stable.

4.2.6. Asynchronous Advantage Actor-Critic

The Asynchronous Advantage Actor-Critic (A3C) algorithm is a policy gradient RL algorithm that combines the benefits of both actor-critic and asynchronous methods to learn policies for decision-making in sequential decision-making problems. It operates in the forward view and uses a mix of n-step returns to update both the policy and the value function. The policy and the value function are updated after every t_{max} action or when a terminal state is reached. The update performed by the algorithm can be seen as $\nabla_{\theta'} \log \pi(a_t|s_{t+k}; \theta_v) A(s_t, a_t; \theta, \theta_v)$, where $A(s_t, a_t; \theta, \theta_v)$ is an estimate of the advantage function given by:

$$\sum_{i=0}^{k=1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{7}$$

where k can vary from state to state and is upper bounded by t_{max} [26].

In A3C, the critics learn the value function while multiple actors run in parallel, with each interacting with a separate copy of the environment, collecting experience, and updating a shared neural network model. The agent threads are asynchronous, which means that they do not wait for the other threads to finish their updates before making their own updates. This leads to faster training and better exploration of the state space.

In [17], the A3C algorithm was among the many algorithms used during the competition in 2017, and the participants who used it were able to achieve greater results than their opponents, with an A3C + behaviour cloning achieving first place.

The A3C algorithm was tested against a A3C-Anticipator network in [13] but it was concluded that there was not a significant difference between the two.

4.2.7. Proximal Policy Optimization

The Proximal Policy Optimization (PPO) is a model-free RL algorithm that 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 optimizes 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 [27]. PPO tends to have better generalization properties compared to TRPO. TRPO is designed specifically for policy optimization 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 also offers improved sample complexity compared to TRPO. While TRPO requires multiple iterations of policy optimization, which can be sample-intensive, PPO employs multiple epochs of mini-batch updates on collected data, which makes more efficient use of the collected samples and leads to better sample efficiency.

The surrogate objective approximates the expected reward based on the collected trajectories. Stochastic gradient ascent optimization is used to maximize 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 (8) shows the PPO's objective function that is not typically found in other algorithms:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \quad (8)$$

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, $_t$ 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:

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 ≤ NT
  $\theta_{old} \leftarrow \theta$
end for

The PPO algorithm is one of the algorithms that is usable by Unity's ML-Agents toolkit and sees much use in papers that use this toolkit, such as [14,15]. Unfortunately, we do not have any data that compare the PPO algorithm to other algorithms in the ambit of training agents for FPS games.

4.3. Training Architectures

In this section, we will talk about the various training architectures that were found in the data.

4.3.1. Single-Agent Reinforcement Learning

Single-Agent RL is a branch of machine learning 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 [16], as is the case in [3], where the agent learns to play various Atari arcade games.

4.3.2. Self-Play

Self-play is a technique often used in RL that involves having RL agents play against themselves to improve performance. As seen in Figure 5, a single agent acts as all players, learning from the outcomes of its own actions. Self-play has been successfully applied in [2], where researchers used this method to develop their Chess- and Shogi-playing AI.

4.3.3. Multi-Agent Reinforcement Learning

Multi-Agent RL focuses on scenarios where multiple agents learn and interact in a shared environment. As shown in Figure 6, 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 it cooperative with each other, competitive between each other in a 1-vs.-1 scenario, or a team-vs.-team scenario [16].

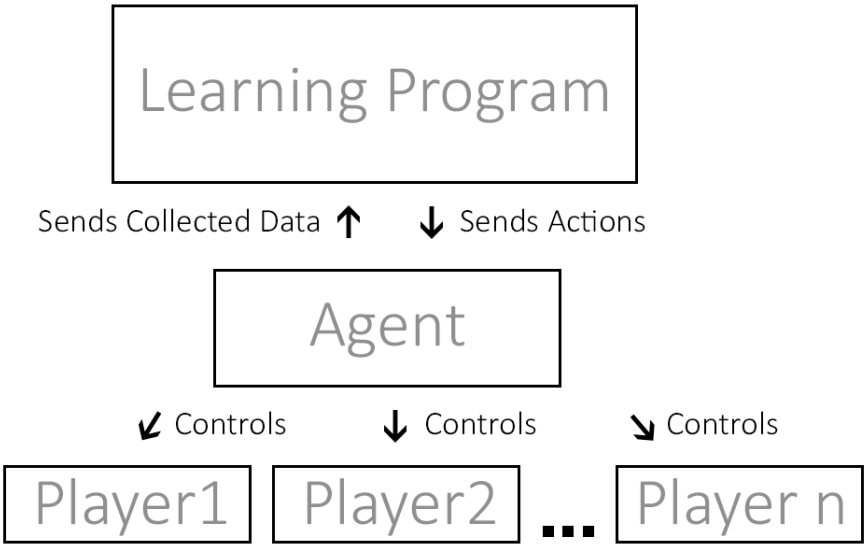

Figure 5. How self-play puts an agent in control of various players.

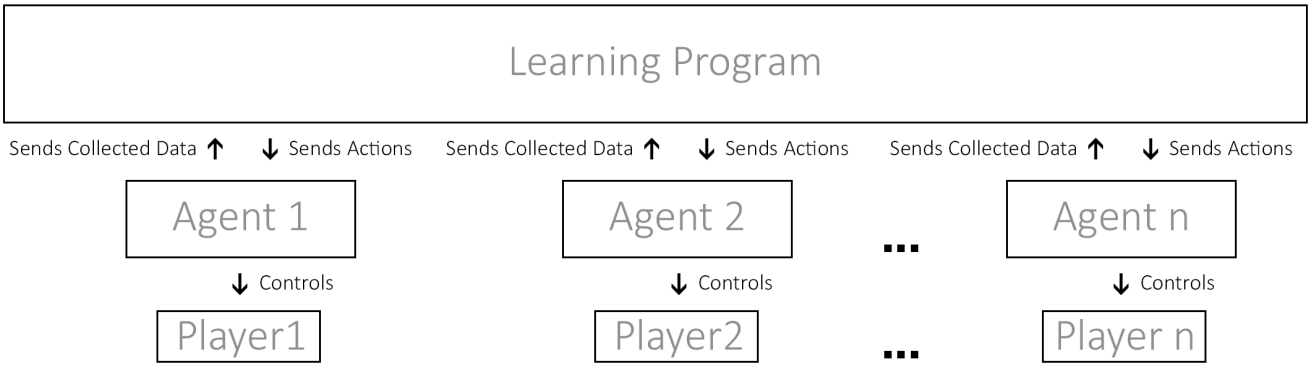

Figure 6. How Multi-Agent RL has multiple agents controlling multiple players, each acting independently from each other, but still contributing to the same policy.

4.3.4. 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 [28]. In video games RL, this usually means having a human player play in the constructed environment, where their actions are recorded and then used to train the agent's policy. The more diverse the recording data are, the better.

4.3.5. Curriculum Training

Just like neural networks are inspired by the human brain, the Curriculum Training 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 the task that the agent is required to perform [29].

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 to become taller, as shown in Figure 7. 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 [14].

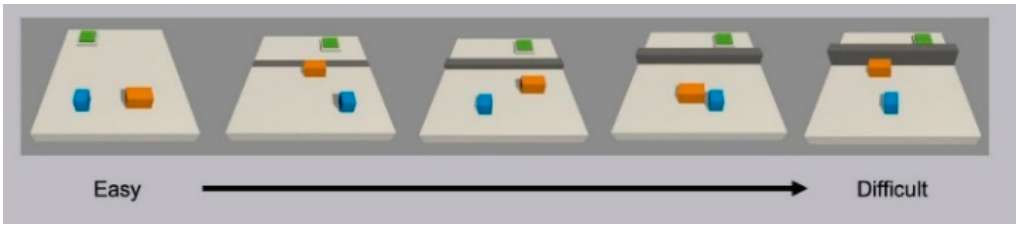

Figure 7. A Curriculum Learning environment where the agent must jump over a progressively higher wall, where the blue and orange objects are 2 agents, the grey object is a wall that increases in size and the green object is the agents' target objective where they have to arrive at [30].

4.4. A Survey of Platforms

There exist various platforms for developing and investigating ML-Agents for video games. Some are exclusively made for research purposes, but engines such as Unity [31] allow both research and application of research to commercial products all in the same environment.

From the gathered data, we have extracted three platforms: DeepMind Lab, ViZDoom, and Unity. We will go over each one of them, explain them, and expose the use cases that were found.

4.4.1. DeepMind Lab

The DeepMind Lab is a platform tailored for ML created by the DeepMind company and released in 2016. It is built on top of the IoQuake3 engine, which by itself is based on Quake III arena code, features assets made for visual learning, and allows for the easy creation of randomly generated levels with text levels. Rendering of the engine uses OpenGL and can use either a GPU or a software renderer [32]. However, due to it being based on such software, its physics and rendering engine can be considered quite dated.

- Use Cases of the DeepMind Lab platform

Google's DeepMind Company used DeepMind Lab in [33] to train RL agents in a multi-agent scenario that required teamwork. They trained their agents for 450 k games with their inhouse developed For the Win (FTW) algorithm, and the initial results from this research showed that their agents started winning against average human opponents after 50 k games. Later, at the 200 k games mark, they began to beat strong human opponents, proving that it was possible to create and train agents to play as a team and even win against human opponents. This research also showed the benefits of using agent-vs.-agent scenarios and how well the multi-agent training architecture works.

4.4.2. ViZDoom

Based on the open-source ZDoom source port, which itself is based on the Doom engine, ViZDoom was created as a platform for researching ML in 3D environments as, at the time, the most widely used platforms were for Atari games [34]. ViZDoom works by giving the agent low-level information with the use of Visual Recognition, although it is also possible to input game variables to the agent [11]. The ViZDoom platform is very lightweight, although this is because its visual and physical complexity is extremely limited by today's standards.

- Use Cases of the ViZDoom Platform

ViZDoom comes with eight different scenarios created for different training purposes, with each one focusing on a different aspect that an RL agent must learn, such as shooting, surviving, identifying, dodging, etc. [35].

ViZDoom was picked by [10] to train agents and test RL agents with a DQN and an Advantage Actor-Critic (A2C) algorithm, because it was lightweight and had native Python support. The experiment's results show that DQN takes less experimental steps and focuses more on exploiting its Q_table, while the A2C algorithm focuses more on updating it. This

in turn meant that DQN was better for exploiting the enemy's weakness, while A2C was better for exploration.

A novel method called combo-action was developed by [18], being macro actions built on a set of primitive actions. The model was trained as a High-level Controller and when tested against other agents, the combo-action agent performed much better, proving the efficiency of the method.

Single-agent-vs.-multi-agent training was tested by [16], pitting RL agents against each other in a 1-vs.-1 scenario and 1 RL agent against a traditional AI agent. The results from these tests showed that agents trained against other agents perform much better, beating the agents that trained against traditional AI agents. This experiment showed that training agents by using multi-agent training scenarios is superior to single-agent training ones, and that future works should always strive to implement multiple agents in their training scenarios.

The Advantage Actor Kronecker-Factored Trust Region (ACKTR) algorithm was compared to the Advantage Actor-Critic (A2C) algorithm by [12], in a single-agent scenario. They found that the ACKTR agents significantly outperformed the A2C agents in all metrics by a significant margin.

An Asynchronous Actor-Critic Agent (A3C) algorithm with an anticipator network was pitted against a regular A3C algorithm [13]. This anticipator predicts the next frame according to the current frame and inputs it to the A3C algorithm. The results show that there is useful information in the predicted frames and that it improves the agent's performance.

In [11], various algorithms were compared: Deep Q-Learning Network (DQN), Double Deep Q-Learning Network (DDQN), Deep Recurrent Q-Network (DRQN), and Double Deep Recurrent Q-Learning Network (DDRQN), while using the TensorFlow library for supervised learning. Results showed that DDQN performed better than DQN in most scenarios except Defend the Centre where they both performed similarly. DRQN and DDRQN also performed better in Defend the Centre (combat scenario). Results also showed that giving the agent access to game variables, such as the agent's health, increased training time but significantly increased performance.

A competition in [17] involved various participants competing in creating the best RL agent player: each one using different algorithms, frameworks, and techniques. Frameworks used included TensorFlow and PyTorch and the algorithms were Direct Future Prediction (DFP), A3C, DRQN, and others. Training architectures included behaviour cloning and curriculum learning. In 2016, victory went to a participant who used TensorFlow with a Direct Future Prediction (DFP) algorithm, while in 2017, the victor also used TensorFlow but this time with a A3C algorithm and behaviour cloning. From this paper, one of the main conclusions is that using behaviour cloning and curriculum learning is superior to not using any, as the contestants who used these training architectures achieved better results than many others who did not. In terms of algorithms, the victor proved to be the A3C algorithms, placing second in 2016 and first in 2017.

4.4.3. Unity

The Unity engine is a cross-platform multimedia platform made for creating 3D and 2D games, simulations, and other experiences [31]. Unlike other previously mentioned platforms, Unity is a standalone general platform, meaning that users can freely create their environments with many more customized parameters than the alternatives. Unity contains its 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 [36]. Furthermore, Unity's in-engine editor is easy and fast to use, allowing for quick prototyping and development of environments [37].

- Use Cases of the Unity Engine Platform

Unity was connected to TensorFlow via an API in [15], where they used a PPO algorithm to train RL agents to play a team-based survival shooter, with the map being a randomly generated maze. The objective was to train agents that could navigate the

randomly generated maze while dealing with enemy agents. Results showed that continuous action spaces with no visual observations and no recurring neural network were the best option.

A Q-learning algorithm with a naïve Bayes (QNBB) approach was used to train an RL state machine in playing as a simple enemy in a custom-made game [25]. This agent could learn from previous games, called "stages". They then conducted a survey in which 20 participants played 5 games each for 4 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 improved with more experience.

As referred to, 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 [36]. The toolkit features support for training single agents, multi-agent cooperative, and multi-agent competitive scenarios with the use of several RL algorithms, such as PPO and Soft Actor-Critic (SAC) [36].

Unity with ML-Agents was used by [38] to create ML-Agents using RL that can participate in competitive FPS matches by employing behaviour cloning. To obtain results, they conducted a survey with 18 participants who were asked to identify the ML-Agent in a multiplayer match. Of the 18 participants, 61% correctly guessed the agent, but many noted that it was hard to identify it.

A multi-agent scenario was used by [39] to train agents in an attacker–defender scenario, by using Unity with ML-Agents.

In [14], the ML-Agents toolkit is used to compare one agent trained with curriculum training to another agent trained without curriculum training. The curriculum agent was not only 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.

5. Discussion

5.1. Platforms and Methods

Our second proposed question was about which methods currently being used were the most effective for creating AI bots for better experiences.

5.2. Algorithms

Of the algorithms that we discussed in Section 4, the data say that the A3C algorithm was the best compared to DQN, DDQN, and A2C, but since these comparations, algorithms such as PPO and SAC have appeared. PPO has the advantage of being a more stable algorithm than A3C, which can be sensitive to hyperparameter settings and requires careful tuning but, unfortunately, there have not been any comparisons between A3C and these in FPS games and, as such, it is difficult to say which is the better one.

5.2.1. Methods

After looking at the data on the various training architectures, we can say that training agents with the self-play or multi-agent training architecture proved much more efficient than using single-agent environments. This is supported not only by [16], who compared the two architectures, but also by [33], who used multi-agent scenarios right from the start, obtaining results much quicker.

However, we believe that before undergoing multi-agent training, training designers should first incorporate curriculum training or behaviour cloning into their training regime. As proven by [14], by using curriculum learning, agents more easily grasp the basics of how to perform the required tasks, much like humans must learn how to walk before running.

An alternative to curriculum learning, behaviour cloning can be used to quickly train an agent to acceptable levels, although [39] reported that agents will start training by not deviating from what was demonstrated, meaning that there is less room for experimentation.

5.2.2. Platforms

In terms of AI research and development platforms, we have identified three (Table 1). Although Unity allows for a more flexible, fast, and easy-to-use environment creation editor, it is still very heavy compared to ViZDoom and DeepMind Lab, which are very lightweight due to being based on older engines.

Table 1. Table with the various identified ML research and development platforms.

Platform Name	Release Date	Base Domain	Computer Vision?	Load Amount	Software Mode?	Other Features
DeepMind Lab	2016	IoQuake3	Yes	Light	Yes	-
ViZDoom	2016	ZDoom	Yes	Very Light	Yes	Flexible training code
Unity's ML-Agents	2017	None	Yes	Heavy	No	Easy and fast to use, advanced physics, flexible environment

ViZDoom can be considered the most flexible in terms of code, as shown by [14], as it can use a wide array of algorithms and frameworks, instead of being chained to what comes out of the box, such as in Unity's ML-Agents' case.

Unlike Unity, ViZDoom and DeepMind Lab also have software rendering [32,34], meaning that rendering of graphics can be performed in the CPU, leaving the GPU free to compute the ML part of the projects.

In terms of physics, due to being based on such old engines, ViZDoom and DeepMind Lab have limited simulations, something that Unity compensates for with its Nvidia PhysX integration.

Although ViZDoom is more lightweight than Unity, we still feel that researchers should consider using Unity or an equivalent due to its flexible environment construction and advanced physics. Both ViZDoom and DeepMind Lab are based on very old engines that are considered more than outdated in today's time, and as such should only be considered in cases where unity or any newer alternative cannot be used in any way.

In conclusion, if what we are looking for is a flexible and easy-to-use training environment that can produce agents usable in commercial projects, we may want to consider using Unity's ML-Agents. However, if our objective is to research algorithms and frameworks in a resource-light platform, then ViZDoom or even DeepMind Lab might be more appropriate.

5.3. Human vs. AI

The third proposed question asks if RL AI bots can rival human players. In [33], we see that at approximately 50 k games, it started beating what was considered an "average" player, while after approximately 200 k games, it started beating "strong" players. Although we only have this data to rely on, we can say that RL AI bots can beat human players if they receive enough training.

Looking at examples outside of FPS games, such as [40], the OpenAI company trained an RL AI with self-play to play the e-sports game of Dota 2. In doing so, the AI managed to beat the world champion in 2019 in an astonishing 2–0 match in favour of the AI team.

Another non-FPS case to look at is [41], where the DeepMind team trained their Alphastar RL AI to play the game of Starcraft 2 and was then made to play the game on the official servers, against human players, ranking up the game's competitive ladder. Alphastar was able to achieve grandmaster rank (the highest rank in the game) on all 3 races available to play and was ranked above 99.8% of players [41].

These papers help supplement the idea that RL AI can indeed be much better than traditional AI, and can even reach the top ranks in terms of skill when given enough training time, although there needs to be more research conducted in the vein of FPS games.

5.4. Better Gameplay, Better Enjoyability

The final proposed question was about evaluating gameplay between RL AI bots and traditional AI bots, to see which one leads to better gameplay and more enjoyability on the part of the player. To answer this question, we turn to the surveys found.

From the collected surveys in [25], we obtained a total of 38 participants who had played against both an RL agent and a traditional AI agent and were in some way asked to compare the two of them. The survey found in [38] is about identifying the RL agents and thus cannot be used here.

The survey in [25] asked participants to evaluate playability from -2 to $+2$ (very bad to very good) for each agent. This study then gives the averages of scores for each agent. After the RL agent had played various games and gained experience, against 20 participants, the average "level of playability" for it was slightly less than 1, while for the traditional agent, it was slightly less than 0.

With these results, our conclusion is that RL agents are just above traditional AI in terms of enjoyability, but we believe that there is insufficient data.

6. Future Work

In this next section, we will talk about future work possibilities, starting with some suggestions, then some potential directions for future work, and how researchers could benefit from using formal methods in their work. Finally, we will discuss potential ethical issues with using ML in FPS.

For future work, one thing to note is that both researchers and developers should not train just one agent (or two if they are comparing something). Multiple agents should always be trained, then either use every trained agent (in cases where teams can be formed) or pick the best ones. One way to test this is to make the agents play the environment and see who achieves better results, such as score, total kills, or fewer deaths. Trained brains do not always perform the same. In [16], one of the trained agents scored erratically and did not perform like its peers; on top of that, all the trained agents had different albeit similar results, yet this proves that problems can occur and it is best to prevent them. Some of the papers analysed during this review only trained one agent for each of their purposes, which raises the question of the credibility of the results. RL includes many randomized factors, meaning that no two agents will ever perform the same even when the environment and training methods are the same.

When comparing training architectures, results should be drawn by pitching multiple teams of bots that were trained in different environments against each other over many games. The team that has the most wins should be considered the best. Many times, researchers just compare the total reward amount, which works for comparing algorithms, but not so much for other parts of the RL training frame. We believe that putting the bots in practical situations and comparing the results should be the go-to method for evaluating performance. Next, we will present two potential directions for future research, Transfer Learning and Multi-Agent RL, and then we will present something that future researchers should keep in mind when building their ML projects: formal methods.

6.1. Transfer Learning

The first research subject is transfer learning, which allows bots to leverage prior knowledge and experience to enhance their performance in a new environment. This could be useful in FPS games where the bots need to learn multiple tasks and adapt to different environments.

6.2. Multi-Agent Reinforcement Learning

Just as [32,39] have achieved, one way to improve bots is to make them work together in teams, by training multiple agents to work together towards a common goal. In FPS games, Multi-Agent RL could lead to more intelligent and coordinated teamplay, which is critical for success in team-based game modes.

6.3. Formal Methods

The complexity and non-deterministic nature of AI algorithms makes it challenging to guarantee the correctness and reliability through traditional testing and validation methods alone. As such, we should turn to formal approaches for AI-based technique verification. Formal methods are mathematical approaches used to specify, build, and verify software and hardware systems [42].

There exist various formal methods that can be used, such as Abstract Interpretation, Semantic Static Analysis, Model Checking, and many others [43]. These methods can be applied to the process of creating RL agents for FPS games in various ways:

- Specification and Verification

Formal methods can help in specifying the desired behaviour and properties of the agent and verifying whether the agent satisfies those properties [44]. This ensures that the agent behaves correctly and meets the specified requirements. Techniques such as model checking, theorem proving, and formal specifications can be used for this purpose.

- Model Construction

Formal methods can aid in constructing formal models of the game environment and the AI agent. These models capture the game dynamics, rules, and interactions. By formalizing the environment and agent behaviours, it becomes possible to reason about their properties and make assertions about the agent's actions and decisions.

- Debugging and Error Detection

Formal methods can assist in debugging and error detection during the development of agents. By analysing the formal models and properties, it becomes possible to identify potential flaws, logic errors, or unintended consequences in the agent's behaviour. This helps in improving the agent's performance, stability, and overall quality.

6.4. Ethical Issues in Training AI for FPS

The rise of ML technology has given rise to many ethical issues related to its development; we believe that it is important to discuss the ethical issues raised by training AI for FPS games.

- Issues with Learning

RL algorithms learn through trial and error, often by maximizing rewards or minimizing penalties. If the reward system is not carefully designed, there is a risk that the algorithm may learn undesirable behaviours. For example, it might prioritize achieving high kill counts at the expense of other objectives, such as teamwork or following ethical rules within the game.

- Transparency

ML algorithms can be complex and difficult to interpret, and this lack of transparency raises concerns about how the AI behaves and makes decisions within the game. Players may question the fairness of the AI opponents' actions if they cannot understand how they work.

Developers must ensure transparency in how their ML algorithms work and what data are used to train them so that the players, consumers, and others not involved in development can better understand how it works [45].

- Data Privacy

ML often requires substantial amounts of data to train effectively. This can raise concerns about privacy if the algorithms collect and process personal player data without proper consent or transparency. Players should have control over the data collected about them and understand how it is used [45].

7. Conclusions

In this study, we delved into the various applications of RL to the creation of AI "bots", or agents for the purposes of using them as opponents in FPS games. Through an extensive exploration of the existing literature, we have gained valuable insights into the current state of the field and identified several key findings.

Firstly, we identified the various algorithms, training architectures, and platforms that researchers are using to create and train their agents. We then discussed these findings, answering each one of the proposed questions one by one, by comparing each one of the found algorithms, each one of the platforms, and finally each one of the methods, and drawing conclusions. We then discussed how these agents perform against humans and how they can bring better gameplay to FPS games, and found that indeed RL agents can beat human opponents. However, it is still hard to say if they bring better gameplay.

In the end, we talked about the future, by firstly considering what researchers should keep in mind in future research, and what new techniques could be explored. We then finished this systematic review by talking about the ethical issues in training AI for FPS games and how developers and researchers alike can avoid bad-use cases.

RL is currently still in its "infancy" and it is still a hardware-demanding task to train an agent, let alone various agents to test against each other. However, the next step in research is to keep increasing the scale with more agents, more inputs, more training steps, and new techniques.

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, A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was partially funded by national funds, through the FCT/MCTES of the projects UIDB/05549/2020 and UIDP/05549/2020 and project GreenHealth, NORTE-01-0145-FEDER-000045, supported by Norte Portugal Regional Operational Program (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

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

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

References

1. 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]
2. 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]
3. 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]
4. McPartland, M.; Gallagher, M. Reinforcement Learning in First Person Shooter Games. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 43–56. [CrossRef]
5. ElDahshan, K.; Farouk, H.; Mofreh, E. Deep Reinforcement Learning based Video Games: A Review. In Proceedings of the 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 8–9 May 2022; pp. 302–309. [CrossRef]

6. 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.
7. Mitchel, T. *Machine Learning*; McGraw-Hill: New York, NY, USA, 1997.
8. Sutton, R.; Barto, A. *Reinforcement Learning—An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018.
9. Page, M.; McKenzie, J.; Bossuyt, P.; Boutron, I.; Hoffmann, T.; Mulrow, C.; Shamseer, L.; Tetzlaff, J.; Akl, E.; Brennan, S.; et al. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *PLoS Med.* **2021**, *18*, e1003583. [CrossRef] [PubMed]
10. Bhojwani, S.; Ganiga, S.; Jain, H.; Jain, S.; Karia, R.; Sayed, N.; Hajgude, J. AI Soldier using Reinforcement Learning. *Int. J. Innov. Sci. Res. Technol.* **2019**, *4*, 1122–1125.
11. Ulrich, C.; Salameh, H.; Wu, M. Training a Game AI with Machine Learning. Bachelor's Thesis, IT University of Copenhagen, Copenhagen, Denmark, 2020.
12. Shao, K.; Zhao, D.; Zhu, Y. Learning Battles in ViZDoom via deep reinforcement Learning. In Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, The Netherlands, 14–17 August 2018; pp. 1–4. [CrossRef]
13. Sun, Y.; Khan, A.; Yand, K.; Fend, J.; Liu, S. Playing First-Person-Shooter Games with A3C-Anticipator Network Based Agents Using Reinforcement Learning. In Proceedings of the International Conference on Artificial Intelligence and Security, Berlin, Germany, 17–19 September 2019; pp. 463–475. [CrossRef]
14. 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.
15. 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 2019 IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; pp. 1–8. [CrossRef]
16. 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 2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Foz Do Iguaçu, Brazil, 29 October–1 November 2018; pp. 117–11709. [CrossRef]
17. Wydmuch, M.; Kempka, M.; Jasjiwski, W. ViZDoom Competitions Playing Doom from Pixels. *arXiv* **2018**, arXiv:1809.03470. [CrossRef]
18. Huang, S.; Su, H.; Zhu, J.; Chen, T. Combo-Action Training Agent for FPS Game with Auxiliary Tasks. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19), Honolulu, HI, USA, 27 January–1 February 2019; pp. 954–961. [CrossRef]
19. Tensorflow. TensorFlow's Official Website. Available online: https://www.tensorflow.org/about (accessed on 20 June 2023).
20. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
21. Abrahams, S.; Hafner, D.; Erwitt, E.; Scarpinelli, A. *TensorFlow for Machine Intelligence*; Bleeding Edge Press: Santa Rosa, CA, USA, 2016.
22. Ketkar, N.; Moolayil, J. *Deep Learning with Python*, 2nd ed.; Apress: New York, NY, USA, 2021.
23. Imambi, S.; Prakash, K.; Kanagachidambaresan, G.R. PyTorch. In *Programming with TensorFlow*; Prakash, K.B., Kanagachidambaresan, G.R., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; pp. 87–104.
24. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. In Proceedings of the 31st Conference on Neural Information Processing System, Long Beach, CA, USA, 4–9 December 2017.
25. 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]
26. Mnih, V.; Badia, A.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1602.01783. [CrossRef]
27. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
28. 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.
29. Soviany, P.; Ionescu, R.; Rota, P.; Sebe, N. Curriculum Learning: A Survey. *arXiv* **2022**, arXiv:2101.10382. [CrossRef]
30. 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).
31. Unity Team. Unity Engine's Official Site. Available online: https://unity.com/ (accessed on 20 June 2023).
32. Beattie, C.; Leibo, J.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. DeepMind Lab. *arXiv* **2016**, arXiv:1612.03801. [CrossRef]
33. 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]

34. Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; Jaskowski, W. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. In Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini, Greece, 20–23 September 2016; pp. 1–8. [CrossRef]
35. Farama Foundation. ViZDoom's Github Scenario Page. Available online: https://github.com/Farama-Foundation/ViZDoom/tree/master/scenarios (accessed on 20 June 2023).
36. Unity Team. The ML-Agent's Github Page. Available online: https://github.com/Unity-Technologies/ml-agents (accessed on 20 June 2023).
37. 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]
38. 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]
39. Akansha, M.; Khan, M. Creating Intelligent Agents in Game Using Reinforcement Learning. *Int. Res. J. Eng. Technol. (IRJET)* **2020**, *7*, 5703–5709.
40. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv* **2021**, arXiv:1912.06680. [CrossRef]
41. Vinyals, O.; Babuschkin, I.; Czarnecki, W.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef] [PubMed]
42. Raman, R.; Gupta, N.; Jeppu, Y. Framework for Formal Verification of Machine Learning Based Complex System-of-System. In Proceedings of the 2011, INCOSE International Symposium, Denver, CO, USA, 20–23 June 2011; Volume 31, pp. 310–326. [CrossRef]
43. Krichen, M.; Mihoub, A.; Alzahrani, M.; Adoni, W.; Nahhal, T. Are Formal Methods Applicable To Machine Learning And Artificial Intelligence? In Proceedings of the 2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH), Riyadh, Saudi Arabia, 9–11 May 2022; pp. 48–53. [CrossRef]
44. Clarke, E.; Wing, J. Formal methods: State of the art and future directions. *ACM Comput. Surv.* **1996**, *28*, 626–643. [CrossRef]
45. Jobin, A.; Ienca, M.; Vayena, E. The global landscape of AI ethics guidelines. *Nat. Mach. Intell.* **2019**, *1*, 389–399. [CrossRef]