Enhancing Rocket League AI Performance Using Proximal Policy Optimization with Principled Reward Structures

Mauro Filomeno
mauro.filomeno@estudiantat.upc.edu

Josep Vidal and Margarita Cabrera-Bean
{josep.vidal, marga.cabrera}@upc.edu

Abstract

AI systems have consistently surpassed human performance in various games, marking significant milestones in artificial intelligence development. From IBM's Deep Blue defeating chess grandmaster Garry Kasparov to Google's AlphaGo mastering the ancient game of Go, these achievements highlight AI's tactical prowess. Today, state-of-the-art AI systems outperform these benchmarks, exceeding human prowess in both board and video games.

Rocket League, a popular multiplayer sports video game blending soccer with vehicular action, has emerged as a dominant force in esports since its 2015 debut. Its dynamic gameplay, which emphasizes teamwork and mechanical skill, presents a unique challenge. In this study, we demonstrate how a reinforcement learning algorithm, initialized from scratch, rapidly surpasses players with extensive gameplay experience. Furthermore, by crafting a concise set of shaping rewards based on the game's core principles, we illustrate how the AI can autonomously acquire advanced game mechanics through self-play.

By leveraging these techniques, our research sheds light on the adaptive and learning capabilities of AI systems in mastering complex real-time environments like Rocket League.

1. Introduction

The fundamental objective of Artificial Intelligence is to simulate human-like intelligence in machines, enabling them to perform complex tasks and make decisions autonomously. Just like in the learning process of a person, games have served as stepping stones for Artificial Intelligence. Having achieved groundbreaking performances in games like Backgammon (1992) [18], Chess (1997) [8] or Atari (2013) [11] through reinforcement learning, research has gone a step further to tackle complex video games.

Rocket League is a multiplayer vehicular soccer video game released by Psyonix in July 2015. The game has averaged 10 000 to 100 000 concurrent players[1] between 2015 and 2024 and has had an active esports scene since 2016, with professional players competing for a prize-pool of over \$4 300 000[2]. With highly-mechanical play, team-driven decisions and consistency being at the core of the highest competitive levels, bots have been banned from online games due to its potential to surpass human gameplay, especially in terms of consistency and mechanics. However, Psyonix's default bots, released with the game in 2015, are easy to defeat even by beginners and there is yet to be produced a bot that can defeat the best teams in the world. This gap between the potential of bot gameplay and the level that the best bots have actually achieved relies in the complex nature of the video game, which presents challenges to reinforcement learning due to big action spaces, sparse game-defined rewards and the interaction with a real-time physics engine.

We attack those challenges by adding a set of shaping rewards based on the core principles of the game. With them the bot is able to learn emergent behaviours and mechanics that don't have a direct reward. Therefore, the bot is able to continuously learn and improve with that set of rewards, surpassing the expected performance. Potentially, this can lead to a cyclic learning process, where the human-defined rewards lead to strong gameplay and the discovery of new mechanics that can surpass human gameplay.

We show how this principle, matched with a good choice of the hyper-parameters of the Proximal Policy Optimization (PPO) algorithm, our choice for the reinforcement learning algorithm, can produce a high-level bot. In particular, with optimized training, the bot is able to surpass the level of the best hard-coded bots, and win games against gold level players, a level which is estimated to take 300-500 hours to master[3], in 8 days of training in a single computing machine.

In section 2, we describe Rocket League in detail, including its challenges. In section 3, we take a look at the

[1]For real-time statistics, visit https://steamcharts.com/app/252950#7d.

[2]https://www.rocketleague.com/en/news/the-rlcs-returns-for-2024

[3]https://t.ly/zz4Ka

state-of-the-art bots. In section 4, we explore the impact of shaping rewards and the emergent behaviours. In section 5, we talk about the algorithm and the learning settings. In section 6, we show the resulting performance of the training of the bot. In section 7, we talk about the next steps for future projects. Finally, in section 8, we talk about the conclusions and the take-aways of our project.

2. Rocket League

2.1. Game rules

The primary objective in Rocket League is to score more goals than the opposing team within the match time, 5 minutes. A goal is scored by hitting the ball into the opposing team's net using the players' cars.

Matches are played with two teams, each having one to four players (the same amount for both teams). Each player controls one car equipped with rocket boosters. The field is a closed rectangle with smoothed edges to allow cars to climb walls and reach the ceiling, with a hole in each short side that is the goal. In order to score, the players must push the ball into the opposing team's goal, and the entire ball must cross the goal line.

For that, cars have the ability to:

1. Throttle
2. Steer
3. Yaw
4. Pitch
5. Roll
6. Jump
7. Boost
8. Handbrake

Each of these is a separate action that can be triggered contemporaneously.

2.2. Challenges

Rocket League entails a set of challenges for reinforcement learning algorithms that have gatekept bots from surpassing top human gameplay. The main challenges are:

- Custom physics engine. The physics engine is a core component that defines the unique gameplay mechanics and feel of the game. The engine is unique and controls the behaviour of the ball, the cars, the aerial dynamics, the collisions, the energy and the momentum through thousands of lines of code. Interacting with it to read each bounce and movement is a big challenge for humans and AI alike.
- Big action space. In an 8-dimensional action space with 6 out of the 8 actions being continuous (1-6), the possible inputs at each second are infinite. Even after discretizing the actions and significantly reducing the complexity of the problem, at each timestep there are 1944 possible actions for the bot to choose from.
- Sparse rewards. The only game-defined reward is scoring. With that, no reinforcement learning bot would learn anything other than random policies. We attack this by introducing shaping rewards in Section 4, at the cost of having to design and test the learning process of the bot in an empty canvas.

In this study, we train an agent for 1v1 matches. In this setting, we are able to meticulously study the design process necessary for surpassing top human play. The extension of the agent for 2v2 and 3v3 matches is left as a direction for future work.

3. State of the Art

The incorporation of machine learning into the Rocket League bot community has revolutionized bot development, surpassing the capabilities of early hard-coded bots. Techniques such as behavioral cloning and reinforcement learning have significantly raised the skill ceiling for bots. An active and dedicated community has been pivotal in driving these advancements, continuously working to enhance bot performance [1]. Notable projects include the development of sophisticated bots like Nexto [7] and ReliefBot [5], which demonstrate advanced gameplay mechanics and strategic decision-making.

Despite these substantial efforts, the creation of a bot that can consistently outperform professional human players remains an unachieved milestone. Professional Rocket League players accumulate over 10 000 hours of gameplay experience and have honed their skills over nine years of competitive tournaments, setting a high bar for AI to surpass.

Our research aims to lay the groundwork for surpassing top human performance in Rocket League. To achieve this, we adhere to the principle that an agent trained from scratch, without emulating human behavior, holds greater potential. Learning the game independently, free from the constraints of human norms, allows for the development of a unique, creative, and dynamic playing style. This approach has proven successful in recent years with games such as chess [15], Go [16], and Dota 2 [13]. These cases illustrate the potential of reinforcement learning and self-play to achieve superhuman performance, suggesting a promising direction for future Rocket League bot development.

4. Shaping Rewards

Given the sparse rewards defined by the game, it is necessary to implement *shaping rewards* to reduce the learning time. Relying solely on scoring as the reward would result in prohibitively long training periods for the bot to learn effective strategies.

To address this, we employ potential-based shaping functions [12]. We design rewards (or penalties) for the agent based on a set of actions derived from principled human play. This approach guides the agent's learning process more effectively, facilitating faster and more efficient skill acquisition.

While each player develops a unique playstyle, the progression from lower to higher skill levels in the game follows a natural evolution of learned skills. Initially, simpler skills suffice against less skilled opponents but prove inadequate as opponents become more proficient. In contrast, advanced skills require extensive practice and are seldom mastered due to their complexity and marginal gains in performance. Mastery of these skills is typically achieved at the highest competitive levels of the game. Attempting to learn advanced skills without first mastering core gameplay mechanics not only requires an impractical amount of time but also offers limited situational improvement. Therefore, focusing on foundational skills before advancing to more complex techniques is essential for substantial and sustainable improvement in gameplay performance.

4.1. Reward engineering

To guide our agent through the learning process towards achieving high-level gameplay, we employ reward engineering. Reward engineering, also known as reward shaping, involves modifying the environment so that reward maximization leads to the desired behavior [9]. Specifically, we utilize a potential-based shaping function as defined in [12], which ensures that the optimal policy in the modified environment remains optimal in the original game environment, where rewards are determined solely by scoring goals. Consequently, our reward is defined as a function

$$\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}^+ \to \mathbb{R} \tag{1}$$

using the notation from Sutton and Barto [17]. Here, $\mathcal{S}$ represents the set of all non-terminal states, $\mathcal{S}^+$ includes the terminal state, and $\mathcal{A}(s)$ denotes the set of all actions available in state s. Since the set of possible actions is the same for any state, we denote $\mathcal{A} := \mathcal{A}(s)\ \forall s \in \mathcal{S}$. Therefore, $\mathcal{R}(s, a, s')$ is the reward received upon taking action a in state s and transitioning to state s'.

In particular, we define the rewards $\mathcal{R}$ as linear combinations of the scoring reward and a set of potential-based shaping functions $(\mathcal{F}_1, \dots, \mathcal{F}_M)$, each of which rewards a different mechanic or aspect of the game:

$$\mathcal{R} = R_S + \sum_{i=1}^{M} w_i \mathcal{F}_i \tag{2}$$

where R_S is the scoring reward defined by the game, and $w_i \in \mathbb{R}$.

This approach allows us to fine-tune the agent's behavior by emphasizing specific elements of gameplay.

4.2. Curriculum learning

A promising approach for designing shaping rewards involves curriculum learning, originally proposed by Bengio *et al.* [6]. Curriculum learning entails a methodical progression where systems begin with simpler concepts and gradually advance to more complex ones over time.

In our experiments, we employ an annealing function to define a sequence of rewards $(\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_N)$ along with corresponding transition points $(t_1 = 0, t_2, \dots, t_N)$. Starting from timestep 0, the system initially receives rewards defined as $\mathcal{R}_1$. These rewards then transition incrementally to $\mathcal{R}_2$ by timestep t_2, and so forth, until reaching timestep t_N, where the rewards are defined as $\mathcal{R}_N$.

This approach mirrors the natural progression of skill development from lower to higher ranks in gameplay. For example, $\mathcal{R}_1$ might include rewards for basic actions such as hitting the ball, positioning near the ball, or attempting shots on the opponents' net. In contrast, $\mathcal{R}_N$ could encompass more advanced mechanics like flip resets, advanced flicks, or executing double taps. By structuring rewards in this manner, the bot systematically builds proficiency from fundamental skills to sophisticated techniques.

However, as intuitive as this approach is, it presents notable downsides which make it inadequate for our objective:

- Defining the values of t_i is impractical and prone to error. To set specific values, we must clearly identify the agent's training progress. Any changes in the algorithm's hyperparameters would alter the learning pace, necessitating adjustments to the t_i values. This increases the number of experiments needed to train the agent, which is costly in terms of time and computation.
- A large number of rewards $\mathcal{R}_i$ must be defined. Reward engineering is a challenging task, and there is no straightforward answer to which rewards are best for our task. Experimenting with different values and functions requires a significant investment of time and computation.
- Curriculum learning constrains the learning path of the agent. To surpass human gameplay by learning from scratch, we want the bot to learn as freely as possible.

4.3. Principled play

Instead, we opt for a simpler definition of the rewards that aims to capture the essential principles of the game. Rather than having the system learn the intricate mechanics of each rank, we reward it for actions that align with the core principles leading to victory. This approach facilitates a more flexible and unrestricted development path.

We build a single reward function $\mathcal{R}$ – as defined in 2 – to be used throughout the entire training process. In doing so, we encounter two primary challenges: the definition of the individual reward components $\mathcal{F}_i$ and the assignment of appropriate weights w_i to each component. Then, in constructing our reward function $\mathcal{R}$, we incorporate a zero-sum framework and normalization of the reward components $\mathcal{F}_i$.

Shaping functions. We find that the reward components $\{\mathcal{F}_i\}_i$ are best defined through human expertise. Based on extensive experience in playing the game, we identify a set of core principles that a player needs to master to progress from the lowest to the highest ranks. These mechanics encompass fundamental skills and strategies crucial for success in the game. The core principles include boost management, pressure, momentum and ball control.

These principles are distinct from the specific mechanics a player must learn to master them. Instead of directly rewarding the mastery of these mechanics, we allow the agent an unconstrained path to discover the most optimal mechanics for each stage of its learning path. For instance, momentum, which is the product of an object's mass and its velocity, is a crucial principle at every level of the game. Players learn various mechanics to maximize momentum, such as boosting, flipping for speed, drifting, recoveries, wave dashes, and half flips. While we could explicitly reward the agent for learning each of these mechanics progressively, our findings indicate that by directly rewarding the principle of momentum, the agent autonomously learns the associated mechanics without specific rewards for each one. This approach not only simplifies the reward structure but also promotes more natural and adaptive learning.

Importantly, this method enables the agent to develop emergent behaviors through mastering complex mechanics. By focusing on fundamental principles, the agent can independently discover and perfect advanced techniques that are typically mastered only at the champion and grand champion levels, where players often accumulate between 1000 and 2000 hours of playtime[4]. This emergent learning leads to sophisticated and highly effective gameplay strategies, demonstrating the agent's ability to achieve high-level performance through the understanding of the game's core dynamics.

In Table 1, we provide a summary of the primary shaping functions we define. As indicated in Equation 2, each shaping function is assigned a specific weight to balance the overall reward structure. These weights are crucial for ensuring that the agent learns effectively across various aspects of gameplay, as discussed in the following paragraph. For clarity, the table excludes shaping functions intended for accelerated learning in the initial stages, which have minimal relative weight, such as facing the ball or landing on four wheels.

[4]https://t.ly/zz4Ka

Reward	**Scale**	**Principles**
Low distance between opponent's goal and ball	Exponential	Control, pressure
Low distance between car and ball	Exponential	Control, pressure
Ball's velocity vector facing opponent's goal	Linear	Pressure
Touching the ball when it is high	Squared	Pressure
Touches resulting in an increase in ball speed	Linear	Pressure
Scoring goals with high ball speed	Linear	Shooting
Scoring goals from far away	Square root	Shooting
Demolishing rival	Binary	Control, pressure
Ground dribbling	Binary	Control
Boost management	Squared	Control
High kinetic and potential energies	Linear	Momentum
High car velocity	Linear	Momentum
High car velocity in the direction of the ball	Linear	Momentum, pressure
Consecutive aerial touches	Exponential	Aerial control, pressure

Table 1. Summary of shaping functions

Weights. To determine the weights $\{w_i\}_i$, we conduct extensive testing with different values to identify those that lead to the desired behavior. Proper fine-tuning of the weights is critical for the system to learn the intended principles effectively. If a weight w_i is set too high, it may cause reward hacking, where the agent learns a sub-optimal strategy that maximizes the high-weight reward at the expense of overall performance. Conversely, if a weight w_i is set too low, the associated reward $\mathcal{F}_i$ becomes negligible, and the agent might ignore it due to the minimal impact on the overall reward $\mathcal{R}$.

To streamline the tuning process, we observe that minor adjustments in the weights do not significantly alter the agent's behavior. It is only when the relative weight of a reward changes substantially that the learning path of the agent is noticeably affected. This insight allows us to make more precise and confident adjustments to the weights during the training process.

Normalization. Normalization of the reward components $\mathcal{F}_i$ ensures consistency and comparability across various components. It involves scaling the reward values to the range $[0, 1]$. This prevents any single reward component from dominating the learning process due to disproportionately large values.

Zero-sum rewards. A zero-sum reward structure ensures that the total amount of reward available to the agent is constant, preventing the inflation of rewards and promoting competitive behavior. In this context, zero-sum rewards imply that the gain of a reward by one agent corresponds to a loss for the opponent. This mechanism is particularly useful in a competitive environment like Rocket League, where the agent's success comes at the expense of the opponent's.

Implementing zero-sum rewards involves balancing positive and negative rewards in such a way that the sum of all rewards across different actions and agents equals zero. For example, if scoring a goal yields a positive reward, the same event can be framed to include a negative reward for the opposing player, maintaining a zero-sum balance.

5. Experimental setup

Reinforcement learning is the key to allowing a bot to learn from scratch and surpass the performance of existing bots that rely on pre-programmed behaviors or imitation learning. In this study, we utilized the Proximal Policy Optimization (PPO) algorithm, a state-of-the-art method in reinforcement learning known for its robustness and efficiency in handling high-dimensional action spaces and continuous action controls [14]. Our approach leverages self-play, where the bot learns by playing against itself, iteratively improving its strategies and decision-making processes over time.

In order to interact with the game, we use a gymnasium-like environment [19]. In [1], Rocket League Gym is designed for that purpose, and does so by interacting with the game in real time. However, to train an agent for a very large number of timesteps, having to execute and speed up an instance of the game is sub-optimal. Instead, we take advantage of Allen's Rocket League Gym Sim [2], which is a wrapper around RocketSim, a C++ library for simulating Rocket League games at maximum efficiency. Moreover, it provides a vectorized implementation of the PPO algorithm that allows for parallel computation [3].

Configuration objects. Configuration objects allow us to define how the agent interacts with the game. In Figure 1, we show the flowchart of Rocket League Gym Sim. The objects are:

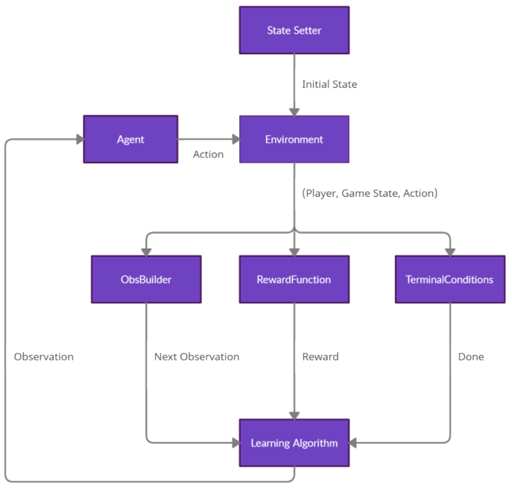

Figure 1. Rocket League Gym Sim flowchart.

- State setter: Definition of the initial state. We use a stochastic function so that 30% of the time, the initial state is the kick-off, as intended by the game. The other 70% is used for initial conditions that force a particular situation in a game, such as having to save a shot. With these, we aim to accelerate the learning process by defining custom training states.
- Reward function: As discussed in 4.
- Observation builder: The observation builder defines the state space by specifying the environment's observations. This includes the physics state of the ball and each player (*e.g.* position and velocity vectors), the status of boosts, timers for boosts and demolitions, and the remaining game time. Figure 2 illustrates a sample state.
- Terminal conditions: Other than scoring a goal, we set as terminal conditions surpassing the 5-minute time limit of a game or having the ball not be touched for 30 seconds. The latter is particularly designed for the early stages of training, so that the agent can focus on learning to interact with the ball.
- Learning algorithm: As mentioned before, we use the vectorized implementation of PPO for Rocket League Gym Sim. We discuss the learning algorithm in the next section.

Proximal policy optimization and hyperparameter selection. Proximal Policy Optimization (PPO) is a policy gradient method used for training an agent's policy network. It leverages an actor-critic architecture where the actor proposes actions based on the current policy, and the critic evaluates these actions to provide feedback on their quality. PPO offers advantages in terms of stability and simplicity: it controls the size of policy updates to prevent drastic changes that could disrupt learning by using a clipping mechanism.

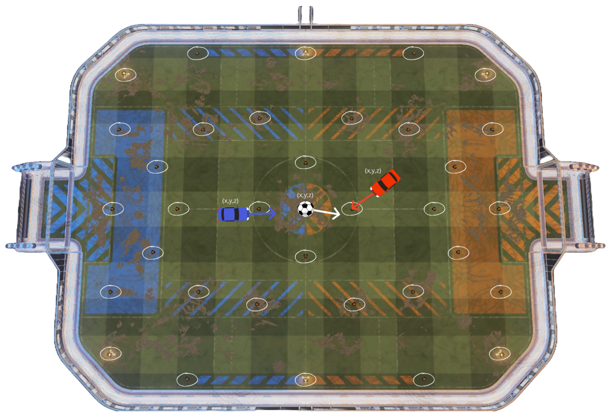

Figure 2. Sample state, excluding timers.

This approach not only enhances stability but also simplifies implementation, making PPO a robust choice for training agents in complex environments. Since 2018 PPO has seen a wide variety of successes, such as controlling robotic arms, beating professional players at Dota 2, and excelling in Atari games [10].

Choosing appropriate hyperparameters is critical for the performance of the PPO algorithm. Hyperparameters govern the training process and significantly influence the stability and efficiency of learning. For the agent to surpass top human gameplay, a long time horizon of training is required. In this setting, catastrophic forgetting can easily occur before a billion timesteps if the hyperparameters are not properly tuned.

We find a good starting point for hyperparameters in the PPO algorithm from previous works and empirical studies, such as [4].

We pay particular attention to the learning rate, which must ensure consistent learning over an extended training period. Setting the learning rate too high can result in catastrophic forgetting, while setting it too low can hinder progress. Therefore, we use the same learning rate for both the actor and the critic to maintain uniform learning dynamics. Another crucial hyperparameter is the PPO entropy coefficient, which controls the level of exploration. We select a value that sustains exploration throughout the entire learning process.

For the neural network architecture, we employ multilayer perceptron (MLP) networks for both the actor and the critic. Our experiments show that using three layers with an equal number of units is optimal for this task. Specifically, we use a larger network for the critic to accommodate the complexity of learning the value function, while a smaller actor network helps reduce the number of parameters without compromising performance.

The final parameters are reported in Table 2.

By carefully tuning these hyperparameters, we achieve a stable and efficient learning process that allows our agent to master the complex and dynamic environment of Rocket League.

Hyperparameter	**Value**
Learning Rate	1.0×10^{-4}
Entropy Coefficient	0.005
Discount Factor (γ)	0.99
Advantage Estimation (λ)	0.90
Clipping Parameter (ϵ)	0.25
Number of Epochs	10
Batch Size	1.0×10^5
Actor layers	(128,128,128)
Critic layers	(512,512,512)
Reward Normalization	Applied
Observation Normalization	Applied

Table 2. Summary of PPO Hyperparameters

Training infrastructure. We train the agent on a personal computer with the following specifications

- **Processor**: Intel Core i7-13700F
- **RAM**: 32GB DDR5
- **GPU**: NVIDIA GeForce RTX 4070 Ti
- **Operating System**: Windows 11

With the hyperparameters given in Table 2, we are able to run 12 500 steps per second on average.

6. Experimental results

We evaluate the performance of the agent after 10 billion timesteps. That is around 216 hours of training. You can view video clips showcasing our bot's performance on our GitHub repository: `https://github.com/maurofr/RL-principled-reward-structures`.

Performance evaluation. To evaluate the performance of our Rocket League bot, we conduct a series of benchmark tests against various levels of human players and other bots. The key metrics used for evaluation are win rate, average goals scored (AGS) and average goals conceded (AGC).

The bot's performance is assessed in the following scenarios:

- Comparison with other bots: We compare our bot with the best official bot, Psyonix Allstar, and the winner of 2018's 1v1 bot championship, Leaf.

- Comparison with human players: We compare our bot with human play. In particular, we test it against a gold player and a grand champion player. Gold is the average 1v1 skill level, while Grand Champion is only reached by less than 1% of the players [5]

We distinguish between human opponents and bots, noting that human players generally exhibit a higher skill level. Specifically, the gold-ranked player consistently defeated Leaf without any difficulty. Furthermore, to illustrate the superior level of the Grand Champion, he not only defeated the Gold player flawlessly but did so with such dominance that he conceded only a single goal, and that too as a result of a risky play. We do not expect our bot to compete against grand-champion-level players in 10 billion timesteps of training.

Our bot plays 10 games against each opponent. We set a limit of goals to the match, so that if a player scores 5 goals, they automatically win the game. The results are reported in Table 3

Opponent	**Wins**	**AGS**	**AGC**
Psyonix Allstar	10	3.4	1.7
Leaf	9	3.8	2.4
Gold	2	2.7	4.1
Grand Champion	0	0.3	5

Table 3. Performance of the bot against bot and human opponents

The evaluation results confirm that our bot surpasses the competitive level of the best hard-coded bots, Leaf and Psyonix Allstar. On the other hand, it is able to compete against a Gold level player, winning 2 games. However, as the player adapts to the bot's playstile, he is able to win more confidently the games. Finally, the Grand Champion is able to easily beat the bot.

Playstyle. Our analysis reveals that the agent adopts a distinctive playstyle that is not explicitly driven by shaping rewards.

- The agent adeptly recognizes the importance of winning *fifties* in 1v1 scenarios. A *fifty*, short for $50/50$, occurs when two players simultaneously contest the ball from opposite sides. Victory in a *fifty* grants a significant advantage, especially in 1v1 matches where no teammates are available to continue the contest. Our bot effectively leverages this and maintains high pressure on the opponent, keeping the ball on the opponent's half for the majority of the game. Professional players emphasize that mastering *fifties* is crucial for reaching top ranks without the need for complex mechanics. Our bot quickly identifies and utilizes this strategy to its benefit.
- The bot autonomously learns a set of mechanics that enhance its gameplay. Key among these are predictive movements, half flips, and wall shots. Remarkably, the bot acquires these skills without any explicit rewards for them, instead recognizing that these mechanics improve overall performance, leading to higher momentum, increased pressure, and more formidable threats, which in turn yield rewards.
- The bot's recoveries are particularly noteworthy. It consistently uses them to maintain high momentum and stay engaged in the play. Effective recoveries are crucial as players progress through the ranks, enabling faster gameplay. In 1v1 matches, poor recoveries can result in open nets for opponents. Our bot demonstrates a high level of proficiency for its competitive tier.

Learning curve. The agent demonstrates a progressive learning curve that effectively balances exploration and exploitation. Remarkably, it begins to exhibit reasonable gameplay proficiency after approximately 1 billion steps and continues to learn without reaching a plateau.

The potential for further enhancement remains significant with increased training time. Even after 10 billion steps, the agent shows inconsistencies in various aspects of gameplay, such as finishing techniques and boost management. Additionally, it has yet to optimize the exploitation of certain rewards, including strategic dribbling and the advantageous use of high balls.

Figure 3 illustrates the average car velocity and the average ball velocity for each episode during training, serving as crucial indicators of the Rocket League bot's performance. The graph clearly shows a continuous improvement trend, with no stagnation. Additionally, we track other metrics such as the player's velocity relative to the ball, total touches, total goals, and average ball height per episode. All these indicators exhibit a similar upward progression, reflecting the agent's ongoing development.

7. Future work

Based on our experimental results, future work should prioritize training with increased computational resources to elevate the bot's performance to a higher level. Leveraging more powerful hardware can significantly improve the agent's capabilities.

Additionally, a promising direction involves integrating more advanced deep learning architectures for the actor and critic networks. Exploring architectures such as

[5]Detailed rank distribution: `https://www.reddit.com/r/RocketLeague/comments/1bl30s4/season_13_rank_distribution/`

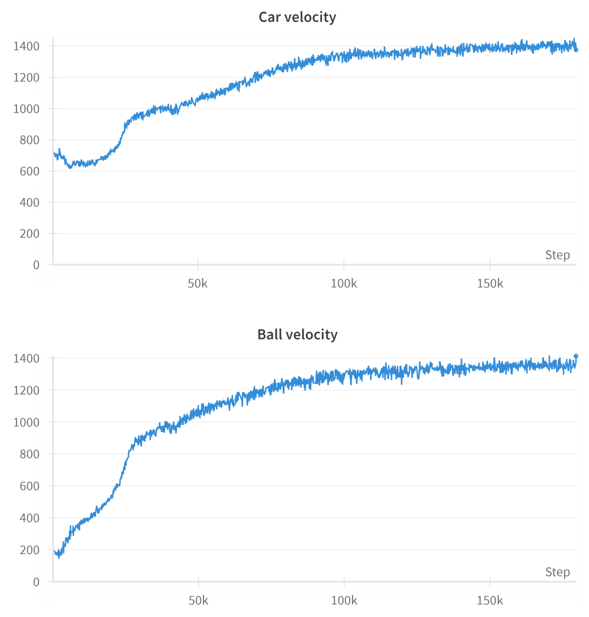

Figure 3. Car and ball velocities throughout the entire training process (10 billion timesteps). The x-axis represents gradient updates, performed every $50\,000$ timesteps.

Transformer-based models could potentially enhance the bot's decision-making and strategic planning.

Lastly, an intriguing avenue for future research is extending the bot's functionality to accommodate 2v2 and 3v3 game modes. This extension introduces the complexity of team dynamics and cooperation. A straightforward yet effective approach to address this is incorporating team spirit into the reward structure, as demonstrated in [13]. This modification can encourage collaborative strategies and improve overall team performance, aligning the agent's incentives with those of its teammates.

8. Conclusion

In this study, we demonstrate the effectiveness of using the Proximal Policy Optimization (PPO) algorithm to train an AI agent for Rocket League, focusing on a principled approach to reward function design. By leveraging human expertise to define core principles and employing a single, comprehensive reward function, we facilitate the development of an agent capable of discovering and mastering advanced gameplay mechanics autonomously. This method not only simplifies the reward structure but also promotes more natural and adaptive learning, leading to emergent behaviors and sophisticated strategies typically seen only at high levels of human play.

Careful tuning of hyperparameters is crucial in maintaining consistent learning over an extended training period and preventing catastrophic forgetting. We successfully train a bot that demonstrates these abilities in 10 billion steps, setting the foundation for prolonged training that can potentially achieve top-tier human gameplay performance.

Overall, this study underscores the potential of principled reinforcement learning methods to push the boundaries of AI performance in dynamic and challenging environments. Our findings highlight the importance of a well-designed reward function and precise hyperparameter tuning.

References

[1] Rlbot. `https://rlbot.org/`. Accessed: 2024-06-09.

[2] Matthew Allen. Rocket league gym sim. `https://github.com/AechPro/rocket-league-gym-sim`, 2023.

[3] Matthew Allen. Rocket league gym sim ppo. `https://github.com/AechPro/rlgym-ppo`, 2023.

[4] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020.

[5] Tyler Arehart. Reliefbot. `https://github.com/tarehart/ReliefBot`, 2017.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[7] Rolv-Arild Braaten. Nexto. `https://github.com/Rolv-Arild/Necto`, 2021.

[8] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57–83, 2002.

[9] Daniel Dewey. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*, 2014.

[10] Arxiv Insights. An introduction to policy gradient methods. `https://www.youtube.com/watch?v=5P7I-xPq8u8`, 2018.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[12] A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[13] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip

Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[16] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[18] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[19] Mark Towers, Jordan K Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Jin Shen Tan, and Omar G. Younis. Gymnasium.