An Application of Reinforcement Learning in Rocket League

Josh Albom
Queen's University
josh.albom@queensu.ca

Justin Badua
Queen's University
justin.badua@queensu.ca

Chase Colby
Queen's University
22PLN@queensu.ca

Ethan Stassen
Queen's University
23JLV2@queensu.ca

Aayan Kader
Queen's University
aayan.a@queensu.ca

Nicolas Raco
Queen's University
nicolas.raco@queensu.ca

***Abstract*—This paper presents the development of a reinforcement learning (RL) agent for Rocket League, aiming to achieve competitive, human-like gameplay. Building upon existing RL frameworks and Proximal Policy Optimization (PPO), we address limitations of prior agents by implementing a refined reward structure that balances offensive and defensive strategies, discrete action spaces for improved control precision, and enhanced observation processing for better spatial awareness. We utilize RLGym and RLBot frameworks for training and interaction, respectively. Our agent demonstrates superior performance against human players, achieving significant score disparities in controlled matches, showcasing advanced ball control, strategic decision-making, and effective execution of ground-based maneuvers. We discuss the agent's architecture, training methodology, and performance metrics, highlighting its strengths in dribbling, flicking, and kickoffs. Limitations, such as the lack of opponent diversity during training and challenges with advanced aerial maneuvers, are also addressed. Future work focuses on enhancing reward functions, exploring alternative learning architectures, and optimizing environment interaction to further improve the agent's competitive performance and strategic adaptability.**

I. Introduction

A. Motivation

Reinforcement Learning (RL) is instrumental in the field of Artificial Intelligence (AI), particularly in complex decision making environments. Rocket League serves as an ideal environment to test and refine RL algorithms, due to its dynamics and physics-based gameplay. Notably, the development of Lucky-SKG, a RL agent for Rocket League, has demonstrated superior performance, outperforming top ranking bots like Necto and Nexto, while establishing new benchmarks in the game [1].

Despite advances in RL, academic and professional awareness remains limited. Highlighting successful applications of RL in popular games such as Rocket League can serve as an engaging way to educate and inform the public about the potential of RL. This application showcases the practical uses of RL, which additionally inspires further research and development in the field.

B. Related Works

Several reinforcement learning (RL) agents are developed for Rocket League, with notable contributions from Necto and Nexto. These agents utilize the RLGym and RLBot frameworks alongside Proximal Policy Optimization (PPO) to train competitive bots. Their methodologies primarily focus on reward-driven policy optimization, allowing them to outperform traditional scripted bots. However, they struggle with fine motor control, adaptability to dynamic scenarios, and complex maneuvers such as aerials and flip resets.

More recently, Lucky-SKG demonstrated superior performance by improving policy optimization and reward structure. Its methodology integrates a more refined balance between offensive and defensive strategies while leveraging self-play for continuous improvement. Despite its advancements, challenges remain in improving strategic decision-making, especially in unpredictable in-game scenarios where human intuition often outperforms AI.

Our work builds upon these efforts by addressing these aspects of training. Our refined reward function, unlike previous approaches, often prioritizes goal-scoring at the expense of broader game play strategy. We introduce a balanced reward structure that accounts for positioning, boost management, and defensive plays. Enhanced Action Parsing: While prior agents rely on continuous action spaces, we implement discretization techniques to improve control precision, reducing erratic behavior. Using advanced observation processing, by altering how the agent perceives game state information, we improve spatial awareness and strategic adaptability. These enhancements contribute to a more competitive and adaptable AI agent, capable of executing high-level plays with greater efficiency. Future work explores further optimization strategies and alternative learning architectures to push the boundaries of RL performance in Rocket League.

C. Problem Definition

The problem we aim to solve is the development of a reinforcement learning (RL) agent capable of playing Rocket League at a competitive level while exhibiting human-level gameplay. Traditional scripted bots, as well as earlier RL-based agents, demonstrate strong decision-making in controlled environments but struggle with complex maneuvers, adaptability, and strategic decision-making in dynamic game states. Additionally, imitation learning approaches, such as TensorBot and Levi, attempt to learn from human replays but face challenges due to data loss, inconsistency, and action dependencies. Formally, given a sequence of game states St, the goal is to determine an optimal policy (at — St) that maximizes the probability of winning the game while ensuring smooth and human-like gameplay. The optimization objective is given by:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right] \tag{1}$$

Where represents the policy parameters, is a trajectory sampled from the policy, is the discount factor, and rt is the reward function at time step *t*.

Balancing exploration and exploitation ensures the agent explores novel strategies while optimizing for performance. Handling continuous and discrete action spaces in Rocket League requires fine-grained control over movements, making action space discretization essential for improved stability. Adapting to dynamic opponents, unlike scripted AI, human players and self-improving bots present unpredictable behaviors, requiring real-time adaptability. By leveraging Inverse Dynamics Models (IDM) for improved action inference and refining reward structures to balance offensive and defensive playstyles, we aim to develop a robust, competitive RL agent that surpasses the performance of existing models while incorporating human-like strategic elements.

II. Methodology

A. Frameworks

To train the agent, we use the RLGym framework [2], allowing the agent to train in Rocket League as if it were an OpenAI Gym environment. The RLGym framework uses a plugin for Rocket League called Bakkesmod [3], increasing game speeds for training purposes. Furthermore, RLGym creates multiple instances of the game to run allowing for multiple agents to train simultaneously.

For the agent to interact with the game, we implement the RLBot framework [4]. The framework allows for the RL agent to connect to the game by providing an API. The framework provides the agent its action space and observation space, which are the actions the agent can take, and the respective information about the current state at the given time step. This enables the framework to let individuals play against the bot in Rocket League.

B. Optimizing Policy

To have the agent play at a competitive level, it requires a policy that maximizes the probability of winning the game. The policy is optimized using Proximal Policy Optimization (PPO)[cite]. The main idea of PPO is that the next policy is similar to the previous policy. As well, PPO uses generalized advantage estimation (GAE) to balance variance and bias. Furthermore, PPO uses an entropy coefficient to balance between exploration and exploitation.

PPO updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a \sim \pi_{\theta_k}}{\mathbb{E}} \left[L(s, a, \theta_k, \theta) \right] \tag{2}$$

Where $L(s, a, \theta_k, \theta)$ is

$$\begin{aligned} L(s, a, \theta_k, \theta) = \min \Big(& \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \\ & g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \Big) \end{aligned} \tag{3}$$

The hyperparameters the agent uses to train are displayed in Table I. One hyperparameter is changed during training, the learning rate, which is reduced as training continues.

Hyperparameter	**Value**
PPO Batch Size	50,000
Timesteps per Iteration	50,000
Experience Buffer Size	150,000
PPO Minibatch Size	50,000
PPO Entropy Coefficient	0.01
PPO Epochs	2
Standardize Returns	True
Standardize Observations	False
Save Every Timesteps	100,000
Timestep Limit	10^{20}
Policy Learning Rate	2×10^{-4}
Critic Learning Rate	2×10^{-4}
Policy Layer Sizes	(1024, 1024, 1024, 1024, 512)
Critic Layer Sizes	(2048, 1024, 1024, 1024, 512)

TABLE I
Hyperparameter Settings

C. Observation Space

At each time step, the agent receives 89 inputs about the game that contain physical information about the player and the ball, as well as the information about boost pads. RLBot and RLGym provide these observations and scale them from [-1, 1] before feeding this information into the neural network.

The game's information passes through a series of arrays of data fed from the game's memory. This allows the agent to know the environment information at each timestep and enables the computational costs to be lower than feeding images into the agent.

D. Action Space

The action space in Rocket League is made up of 5 continuous actions ranging in the interval [-1, 1], and 3 boolean actions (Table II). However, continuous actions in RL make it hard for the agent to control the car. Therefore, we turn the continuous actions into discrete actions by making bins, whereby the agent can select a discrete value to throttle, steer, and roll, allowing the agent to have better control over the car.

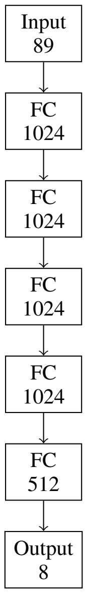
E. Neural Network Architecture

The agent uses a feedforward network that is trained using PPO in an actor-critic framework. The network receives the 89 inputs from the observation space, which are preprocessed and converted into PyTorch tensors. The tensors are then

Action	Domain
Throttle	{-1, 0, 1}
Steer	{-1, 0, 1}
Pitch	{-1, 0, 1}
Yaw	{-1, 0, 1}
Roll	{-1, 0, 1}
Jump	{0, 1}
Boost	{0, 1}
Handbrake	{0, 1}

TABLE II
ACTION SPACE FOR LOOKUPACTION

fed through four fully connected layers with ReLU activation functions. The final layer of the network outputs discrete logits corresponding to different control actions (such as throttle, steer, jump, boost, etc.). These logits are then adjusted for uniformity, where padding is applied as needed, ensuring that all action dimensions are compatible.



F. Reward Weights

For the agent to learn an optimal policy, reward functions are used to speed up learning. Table III shows the rewards that the agent can earn with their associated weights. The rewards given to the agent are either sparse or dense, with sparse rewards being received only when a certain condition is met, whereas dense rewards are given at each timestep. Each time step, the reward is multiplied by its associated weight to obtain the total reward for the agent at that time step.

G. Exploration

We allow the agent to explore the environment in two ways. The first being the entropy coefficient, during the entire training process, the coefficient is set to 0.01, allowing the bot to occasionally explore new gameplay options. In addition, the agent's state is randomized to increase the amount of scenarios the agent is in. Instead of always being placed in the default kickoff position, eighty percent of the time the agent and ball are put in a random spot on the field with a random velocity.

Reward Name	Weight
Goal	300
Demo	5
VelocityBallToGoalReward	1.1
FaceBallReward	0.08
TouchVelReward	1
JumpTouchReward	0.3
SpeedTowardBallReward	0.2
AlignBallGoal	0.4
LiuDistanceBallToGoalReward	0.4
LiuDistancePlayerToBallReward	0.3
FlipResetReward	100

TABLE III
REWARD WEIGHTS FOR DIFFERENT EVENTS

III. RESULTS

A. Milestones

During the initial training phase, the agent was set up to learn how to jump and touch the ball, both key parts of playing Rocket League. This laid the groundwork to later perform more complex actions. Once the agent became proficient in jumping and ball touches, the training shifted from jumping and ball touches, to pushing the ball towards the net and accurate goals. As the agent became more accurate, it was encouraged to then put more force into the ball to enhance offensive capabilities. Building on these foundations, advanced techniques could now be introduced such as flicks and dribbles. Flicking allowed the agent to redirect the ball in mid air to perform advanced shots. Dribbling gives the agent the ability to quickly carry the ball into the net. Each of these milestones greatly contributed to the agent's overall skill set, paving the way for complex and strategic behavior in the field.

B. Strategy

The agent demonstrates an impressive range of strategies, particularly excelling in dribbling and executing flicks at close range to enhance its goal-scoring ability. Without direct input, the agent effectively implements these techniques, showcasing advanced ball control and offensive decision-making. Furthermore, the agent consistently performs well in kickoffs, mirroring the approach of professional players in one-on-one scenarios. While it does not master aerial attacks, it efficiently executes flips, controlled shooting, and ball receptions from high touches to transition into dribbles and scoring opportunities. On defense, the agent exhibits patience by waiting for shots and strategically positioning itself, making it highly effective in maintaining control over matches. During testing, the agent remained undefeated against all human challengers, demonstrating its ability to outmaneuver opponents through refined strategy and adaptability.

The agent also displays a fundamental understanding of boosting and flipping into the ball, effectively using boost to return to defensive positions or advance offensively while maintaining ball control. Its well-trained kickoff approach remains consistent regardless of positioning, leading to a high success rate in gaining possession and frequently converting kickoffs into goals.

C. Scores

During our conference showcase, we conducted a series of eight matches between a human player and our reinforcement learning-based Rocket League agent. The recorded scores, presented as human score vs. bot score, are as follows:

- Match 1: 1–29
- Match 2: 1–17
- Match 3: 1–20
- Match 4: 2–20
- Match 5: 0–24
- Match 6: 1–18
- Match 7: 0–7
- Match 8: 0–37

A detailed examination of these results reveals a significant performance disparity between the human player and the bot. In most matches, the human goal tally is extremely low compared to that of the bot. For example, in Match 1, the human scored only one goal versus the 29 goals of the bot. This yields a human-to-bot score ratio of approximately 0.0345, or 3.45%, when calculated as:

$$\text{Ratio} = \frac{\text{Human Score}}{\text{Bot Score}} = \frac{1}{29} \approx 0.0345$$

Similarly, Match 4 represents the best relative performance for the human, where a score of 2 against 20 results in a ratio of 0.1 or 10%. In contrast, Matches 5, 7, and 8 recorded no goals for the human, corresponding to a 0% ratio.

When summing the performance across all matches, the total goals scored were 6 for the human and 172 for the bot. This cumulative data gives an overall human scoring percentage calculated by:

$$\text{Overall Human Percentage} = \frac{6}{6+172} \approx \frac{6}{178} \approx 3.37\%$$

These numbers highlight that, on average, the human contributed only about 3.37% of the total goals across all matches.

D. Limitations

Several limitations highlight areas for further improvement. One significant limitation is the lack of opponent diversity during training. During training, the agent was only trained against the current version of itself, this may have restricted its ability to generalize effectively against more advanced strategies. Computational constraints also presented a challenge, limiting the extent of experimentation with different reward functions, hyperparameters, and training durations. Another notable limitation is the agent's inability to execute advanced aerial maneuvers such as aerial shots, flip resets, and ceiling plays. Although the agent demonstrated strong ground-based mechanics, including dribbling, flicks, and powerful shots, it struggled to take advantage of airborne plays.

IV. Conclusion

The investigation into integrating artificial intelligence into Rocket League entailed developing and training an AI agent using reinforcement learning techniques. Our work involved utilizing a simulation environment (RLGym) that replicates key elements of Rocket League gameplay, implementing a reward function tailored to promote desirable agent behaviours, and fine-tuning the agent using PPO to achieve optimal performance. Through iterative experimentation with unique rewards, a baseline from which the agent could learn fundamental actions was established. These actions include tracking the ball, jumping, using boost, striking the ball, and most importantly, getting goals.

Looking forward, several avenues for further development emerge. Enhancing the reward function allows the agent to handle more advanced strategies and make more intelligent in-game decisions to create a more sophisticated play style. Experimenting with more additional sub-rewards, such as positioning, defense, and goaltending, would help make the agent play more strategically.

Increasing the training process by adding the number of parallel environments and challenging the agent against its previous versions, can allow stronger adaptability and more efficient development. Additionally, by having agents play against each other (themselves) and retaining the winning strategies, this will allow the agent to improve continuously. This gradual progression from simpler scenarios to more complex challenges may be beneficial for achieving a higher level of competitive performance.

Another direction for further development is optimizing how the agent perceives and interacts with the environment. Developing custom observation parsers that extract richer, more detailed state information, such as spatial-temporal features and complex game dynamics, could enable a deeper understanding of the agent's surroundings. Additionally, exploring different action parsers that provide a more continuous and flexible control mechanism may allow the execution of more precise maneuvers, such as pinches and flip resets.

Ultimately, future research should aim to identify new reinforcement learning approaches that enhance both learning efficiency and practical in-game performance, with the understanding that these refinements may yield unexpected insights beyond the scope of the current study.

References

[1] V. Moschopoulos, P. Kyriakidis, A. Lazaridis, and I. Vlahavas, "Lucy-skg: Learning to play rocket league efficiently using deep reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2305.15801

[2] R. Developers, "Rlgym: A reinforcement learning environment for rocket league," 2025, accessed: 2025-03-15. [Online]. Available: https://rlgym.org/

[3] B. Developers, "Bakkesmod: The all-in-one mod for rocket league," 2025, accessed: 2025-03-15. [Online]. Available: https://bakkesmod.com/index.php

[4] RLBot, "Rlbot: Reinforcement learning for rocket league," 2025, accessed: 2025-03-15. [Online]. Available: https://rlbot.org/