Reinforcement Learning in Rocket League

Benjamin Pelham
Undergraduate Computer Science Capstone
Capstone Advisor: Dr. Michael J. Reale
SUNY Polytechnic Institute
Fall 2022

Introduction

The primary goal of this project was to use reinforcement learning to train a deep-learning neural network to interact with an agent in Rocket League. Rocket League is a video game in which each player controls a car, with the objective being to navigate the pitch and score more goals than the opponent, much like soccer. This is a difficult game for software to learn due to the complexities of controlling the car in addition to the substantial set of relevant information that must be considered for every decision.

Machine learning and neural networks are at the forefront of technological advancement and it is therefore important to understand how they are implemented and trained. This project served as a demonstration of abilities as well as an opportunity to gain specific knowledge regarding the use of deep learning.

Related work

The Rocket League bot-making community has paved the way for the creation of deep-learning agents with RLBot [1] and RLGym [2]. RLBot is a platform that allows for the creation and sharing of bots, as well as the ability to start offline matches in Rocket League playing with or against these bots. RLGym is a custom python API that implements the OpenAI Gym environment into Rocket League. This allows for direct interaction between the game and the agent in said environment, as well as accelerated training with multiple game instances.

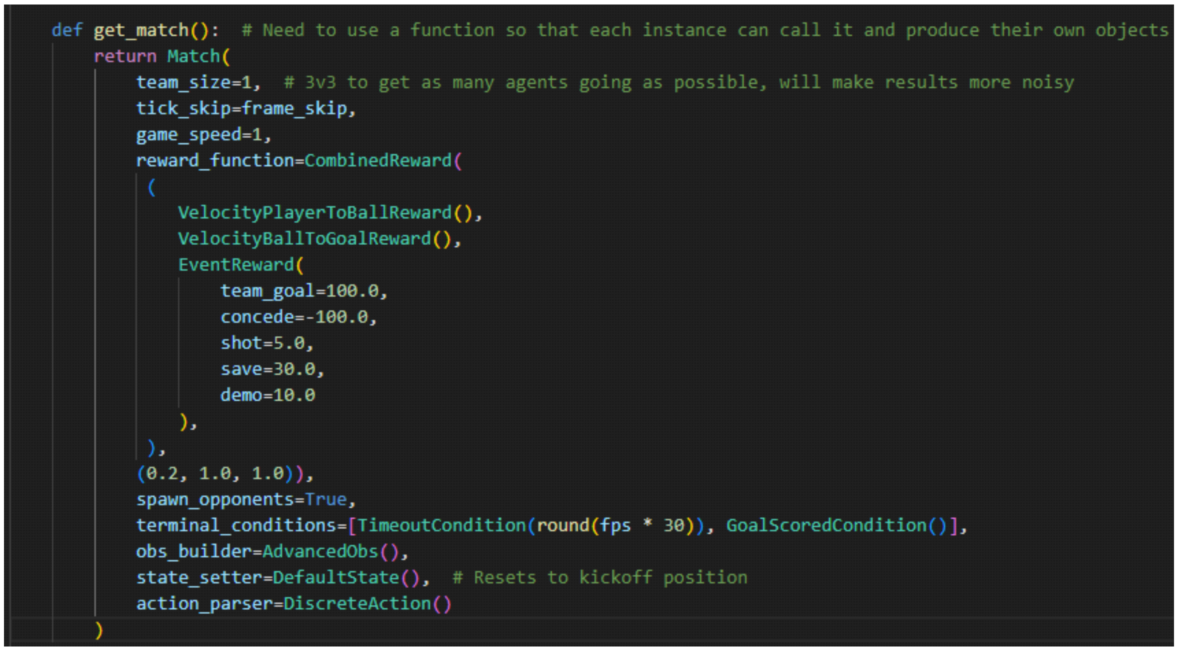
The Necto bot [3] series, which utilizes both RLBot and RLGym, is the most advanced Rocket League bot available. While Necto versions have years of total training and an advanced reward system, the overall structure and techniques used for training are very similar to this project.

Method

Much of the code for this project originated from a YouTube tutorial for reinforcement learning in Rocket League [4].

Main

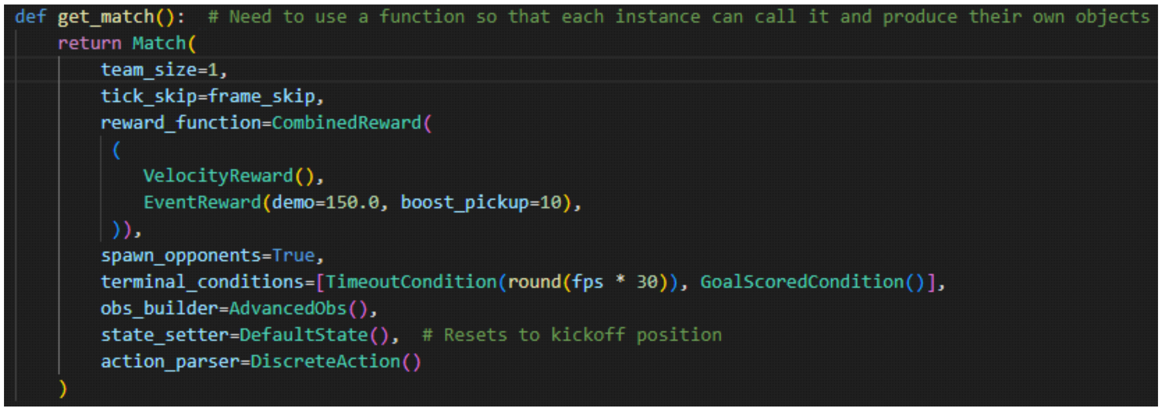
The exBot.py and demoBot.py files are the central location from which the regular and demo bots run, respectively. These files illustrate the framework for each bot, providing environment settings, defining the reward functions, setting network architecture, loading previous model saves, and dictating the logging mechanism and save frequency for training. The get_match() function is used to allow each training instance to create a match object that contains information such as team size, the reward function, terminal conditions, etc. (See figures 1 and 2)



```python
def get_match():  # Need to use a function so that each instance can call it and produce their own objects
    return Match(
        team_size=1,  # 3v3 to get as many agents going as possible, will make results more noisy
        tick_skip=frame_skip,
        game_speed=1,
        reward_function=CombinedReward(
        (
            VelocityPlayerToBallReward(),
            VelocityBallToGoalReward(),
            EventReward(
                team_goal=100.0,
                concede=-100.0,
                shot=5.0,
                save=30.0,
                demo=10.0
            ),
        ),
        (0.2, 1.0, 1.0)),
        spawn_opponents=True,
        terminal_conditions=[TimeoutCondition(round(fps * 30)), GoalScoredCondition()],
        obs_builder=AdvancedObs(),
        state_setter=DefaultState(),  # Resets to kickoff position
        action_parser=DiscreteAction()
    )
```

Figure 1: Shows the 'get_match()' function in exBot.py

In the above figure, the reward function is set to "CombinedReward", an RLGym class that allows for several different reward types to be combined. In figure 1, the reward function consists of VelocityPlayerToBallReward(), which provides more reward the faster the agent moves towards the ball, VelocityBallToGoalReward(), which provides more reward the faster the ball is moving towards the opponent's net, as well as rewards for scoring, shooting on target, getting saves, and negative reward for getting scored on. This combination of rewards is designed to incentivize the agent to perform beneficial actions such as hitting the ball towards the opponent's net, saving the ball, etc.

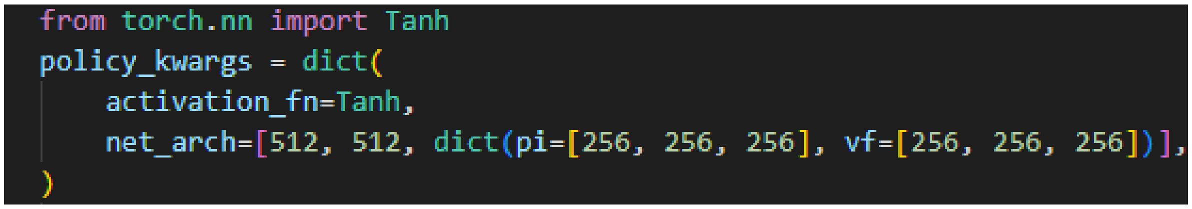


```python
def get_match():  # Need to use a function so that each instance can call it and produce their own objects
    return Match(
        team_size=1,
        tick_skip=frame_skip,
        reward_function=CombinedReward(
        (
            VelocityReward(),
            EventReward(demo=150.0, boost_pickup=10),
        )),
        spawn_opponents=True,
        terminal_conditions=[TimeoutCondition(round(fps * 30)), GoalScoredCondition()],
        obs_builder=AdvancedObs(),
        state_setter=DefaultState(),  # Resets to kickoff position
        action_parser=DiscreteAction()
    )
```

Figure 2: Shows the 'get_match()' function in demoBot.py

As shown in the above figure 2, the demoBot's reward function is different, as the demoBot's goal is to get demolitions on the opponent, rather than win the match (A demolition is achieved when one player hits an opponent at a direct angle while traveling at the maximum speed). The VelocityReward() provides more reward the faster the agent moves around the pitch, and the EventReward provides reward for getting demos and picking up boost pads. These rewards are designed to incentivize the agent to move quickly in order to get as many demolitions as possible.

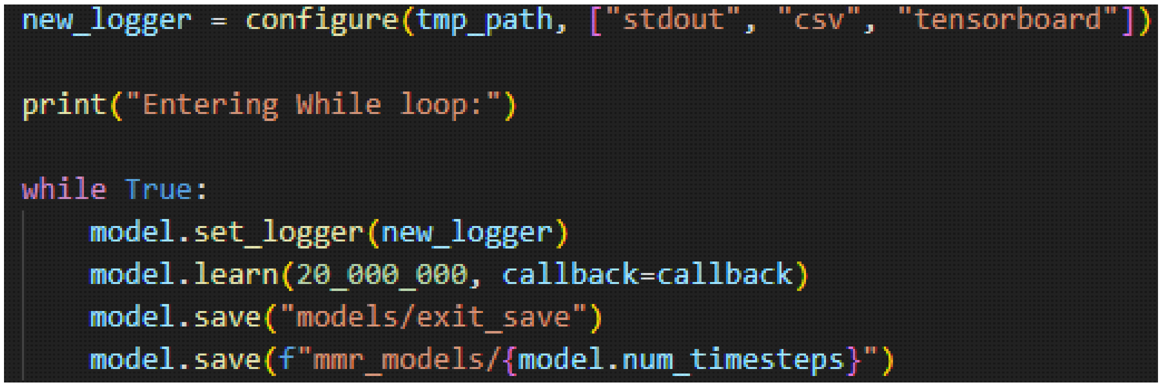
The network architecture is set using the net_arch parameter of the Stable Baselines 3 PPO (Proximal Policy Optimization) algorithm. As shown in figure 3, the network for this project has two shared layers of size 512, three non-shared value network layers of size 256, and three non-shared policy network layers of size 256 [5].



```
from torch.nn import Tanh
policy_kwargs = dict(
    activation_fn=Tanh,
    net_arch=[512, 512, dict(pi=[256, 256, 256], vf=[256, 256, 256])],
)
```

Figure 3: Defining the network architecture

Lastly, the exBot.py and demoBot.py files create a logger to document learning progress and then call model.learn in a while loop. This trains the models for a set number of steps, then saves the models and repeats until the program is ended. This is shown in figure 4 below.

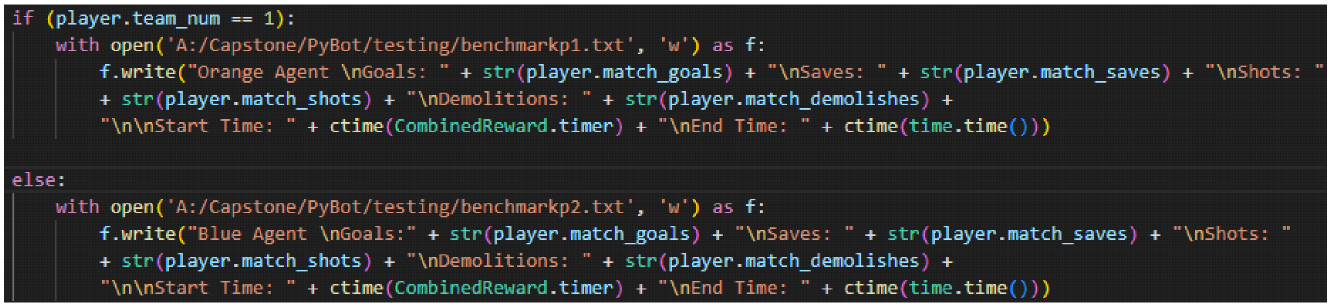


```
new_logger = configure(tmp_path, ["stdout", "csv", "tensorboard"])

print("Entering While loop:")

while True:
    model.set_logger(new_logger)
    model.learn(20_000_000, callback=callback)
    model.save("models/exit_save")
    model.save(f"mmr_models/{model.num_timesteps}")
```

Figure 4: The learning loop

Stat Logging

In order to get a more accurate assessment of training sessions, a custom logger was created (See figure 5). This logger is located within the RLGym combined_reward.py file and it prints the match stats to a text file whenever the reward function is called. The goals, shots, saves, and demolitions of each agent as well as the start and end time of the training session are recorded.



```python
if (player.team_num == 1):
    with open('A:/Capstone/PyBot/testing/benchmarkp1.txt', 'w') as f:
        f.write("Orange Agent \nGoals: " + str(player.match_goals) + "\nSaves: " + str(player.match_saves) + "\nShots: "
        + str(player.match_shots) + "\nDemolitions: " + str(player.match_demolishes) +
        "\n\nStart Time: " + ctime(CombinedReward.timer) + "\nEnd Time: " + ctime(time.time()))

else:
    with open('A:/Capstone/PyBot/testing/benchmarkp2.txt', 'w') as f:
        f.write("Blue Agent \nGoals:" + str(player.match_goals) + "\nSaves: " + str(player.match_saves) + "\nShots: "
        + str(player.match_shots) + "\nDemolitions: " + str(player.match_demolishes) +
        "\n\nStart Time: " + ctime(CombinedReward.timer) + "\nEnd Time: " + ctime(time.time()))
```

Figure 5: Code for logging game stats

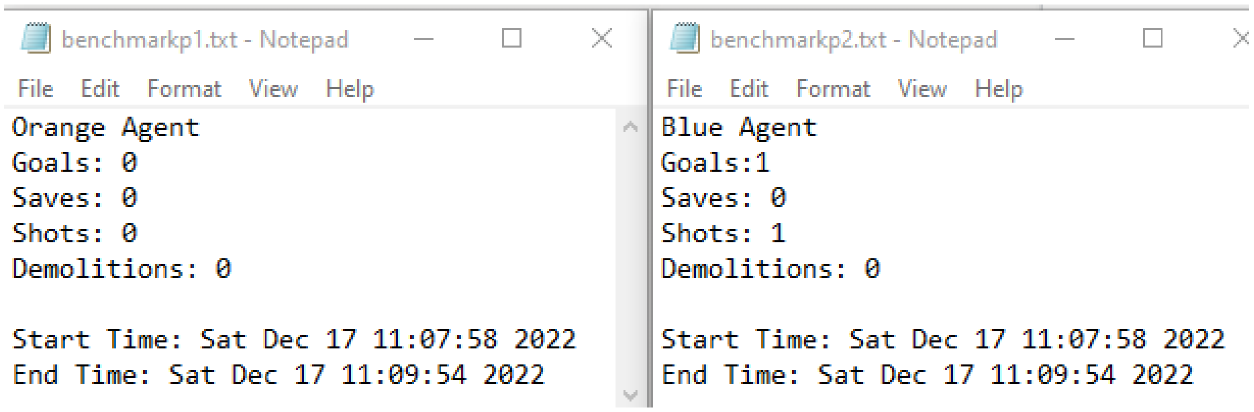
Results

The most important and certainly most time-consuming part of this project was the training. Both bots were essentially inputting random actions at the start of the training. After many hours, the bots began driving instead of randomly flipping around. Eventually, the agents began to perform the actions that their reward functions were designed to encourage. Using the custom stat logger, the progress of each bot can be tracked by completing a short test run using the saved state of the network from different stages of the training process. Each of the benchmark results in this section show the stats from each player (one instance of the bot on the blue team and one on the orange team) generated from 30 in-game minutes of playing.

Bot #1

The first bot, “exBot”, was created with the goal of becoming as proficient as possible at playing the game. This meant scoring, defending, etc. as well as possible. This agent trained for an approximate total of 400 hours. This equates to around 60,000 hours of in-game training because each training session consisted of 5 game instances, each of which the bot is playing against another instance of itself. This means that at any time during training, there were 10 instances of the bot learning. The RLGym training environment allowed for each game instance to be run at around 15 times the normal game speed, meaning in total, training was 150 times as fast as one agent training in real time.

This first benchmark was run on the save instance when the bot had about 50 hours of training (See figure 6). These results show that despite playing a 30 minute match, this early version of exBot was only able to score one goal. This is because the network had yet to “learn” how to play the game yet and was basically controlling the agent with random inputs.


benchmarkp1.txt - Notepad
File Edit Format View Help

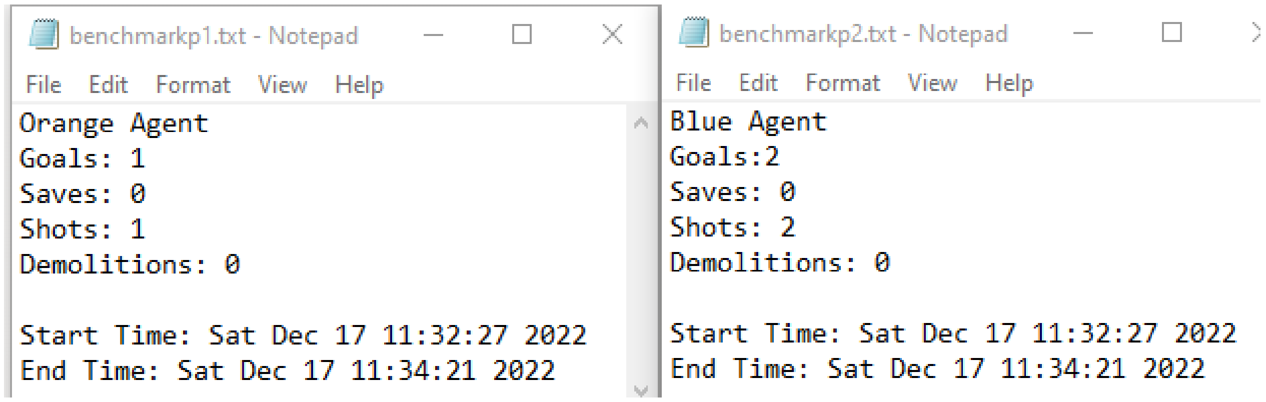
```
Orange Agent
Goals: 0
Saves: 0
Shots: 0
Demolitions: 0

Start Time: Sat Dec 17 11:07:58 2022
End Time: Sat Dec 17 11:09:54 2022
```

benchmarkp2.txt - Notepad
File Edit Format View Help

```
Blue Agent
Goals:1
Saves: 0
Shots: 1
Demolitions: 0

Start Time: Sat Dec 17 11:07:58 2022
End Time: Sat Dec 17 11:09:54 2022
```

Figure 6: Benchmark results for "exBot" with 50 hours of training

The second benchmark results shown in figure 7 below were generated from an exBot save state with approximately 100 hours of training. At this point. the agent still seemed to be somewhat random but possibly started to try and move towards the ball, which led to a few more goals being scored.


benchmarkp1.txt - Notepad
File Edit Format View Help

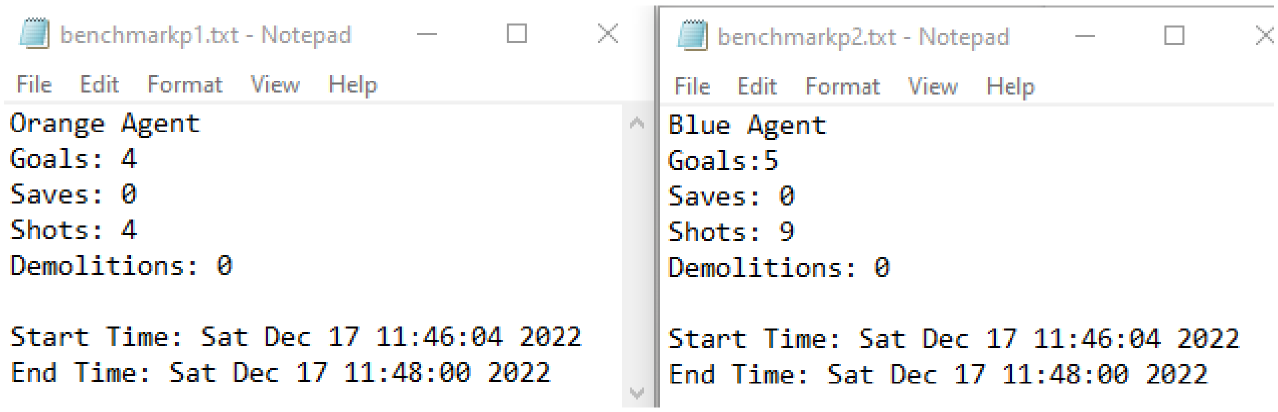
```
Orange Agent
Goals: 1
Saves: 0
Shots: 1
Demolitions: 0

Start Time: Sat Dec 17 11:32:27 2022
End Time: Sat Dec 17 11:34:21 2022
```

benchmarkp2.txt - Notepad
File Edit Format View Help

```
Blue Agent
Goals:2
Saves: 0
Shots: 2
Demolitions: 0

Start Time: Sat Dec 17 11:32:27 2022
End Time: Sat Dec 17 11:34:21 2022
```

Figure 7: Benchmark results for "exBot" with 100 hours of training

The third benchmark, as seen in figure 8, showed significant progress in the goal-scoring ability of the bot. The agent now clearly attempted to move towards the ball at all times, despite still being rather inefficient at moving around.

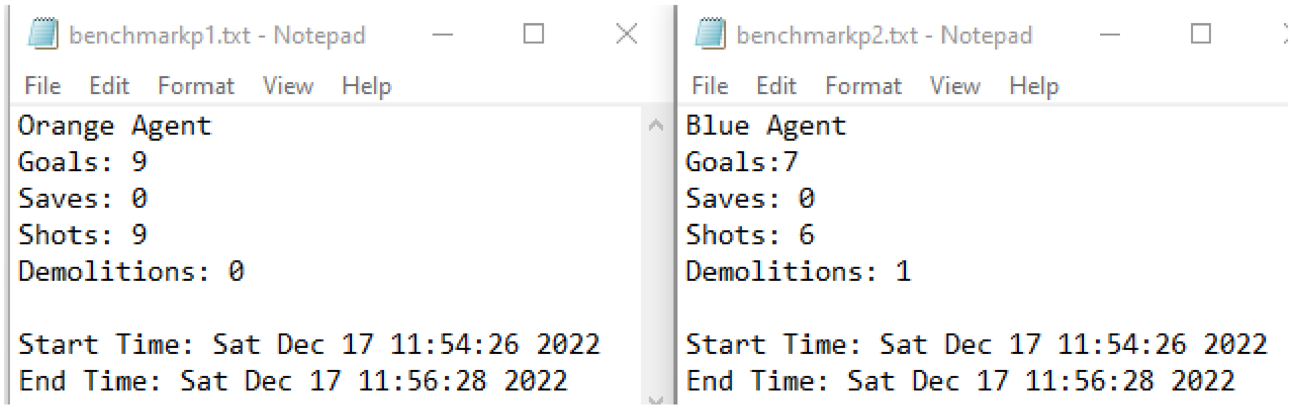

benchmarkp1.txt - Notepad
File Edit Format View Help
benchmarkp2.txt - Notepad
File Edit Format View Help

```
Orange Agent
Goals: 4
Saves: 0
Shots: 4
Demolitions: 0

Start Time: Sat Dec 17 11:46:04 2022
End Time: Sat Dec 17 11:48:00 2022
```

```
Blue Agent
Goals:5
Saves: 0
Shots: 9
Demolitions: 0

Start Time: Sat Dec 17 11:46:04 2022
End Time: Sat Dec 17 11:48:00 2022
```

Figure 8: Benchmark results for "exBot" with 200 hours of training

The fourth benchmark was recorded with the network at 300 hours of training (See figure 9). It was around this point that the agent began driving in relatively straight trajectories, as opposed to flipping around inefficiently. This drastically improved the agent's ability to move around the pitch as well as score goals (See figure 10) as seen in the 16 goals scored in this test compared to 9 in the last benchmark. The demolition achieved by the blue team agent also shows the progress made in movement, as a demolition is only possible when the car is traveling at the maximum speed.


benchmarkp1.txt - Notepad
File Edit Format View Help
benchmarkp2.txt - Notepad
File Edit Format View Help

```
Orange Agent
Goals: 9
Saves: 0
Shots: 9
Demolitions: 0

Start Time: Sat Dec 17 11:54:26 2022
End Time: Sat Dec 17 11:56:28 2022
```

```
Blue Agent
Goals:7
Saves: 0
Shots: 6
Demolitions: 1

Start Time: Sat Dec 17 11:54:26 2022
End Time: Sat Dec 17 11:56:28 2022
```

Figure 9: Benchmark results for "exBot" with 300 hours of training

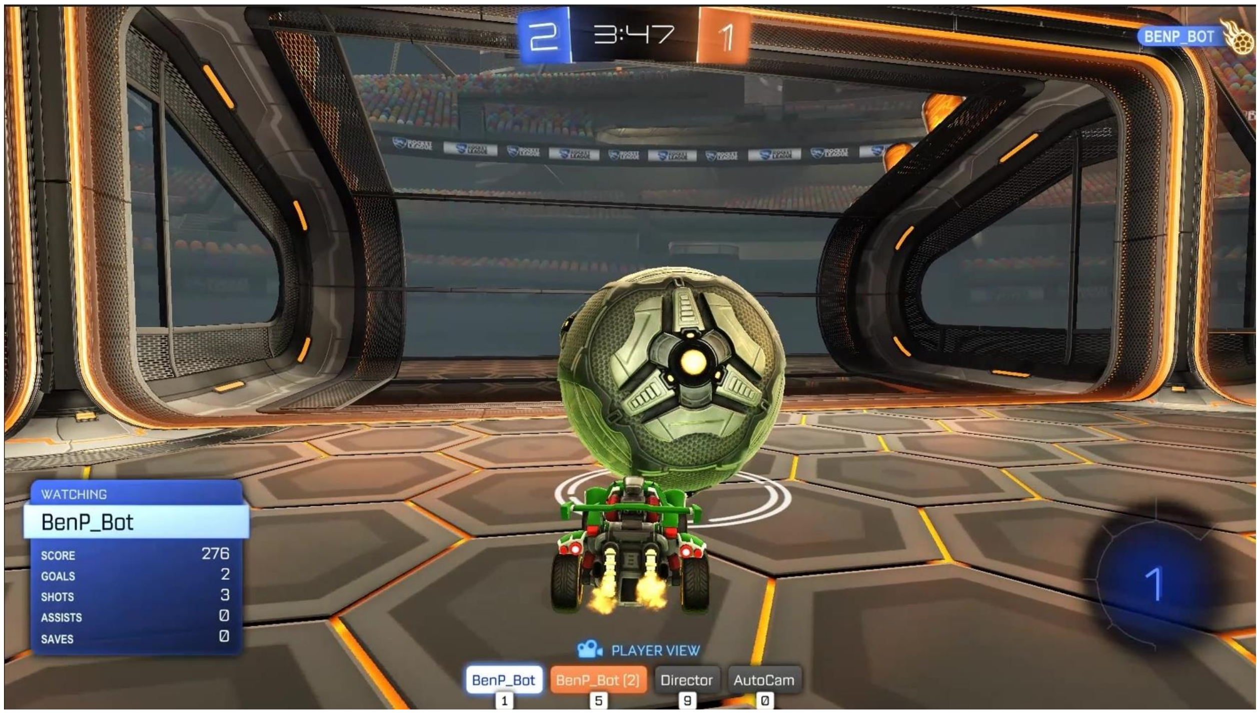
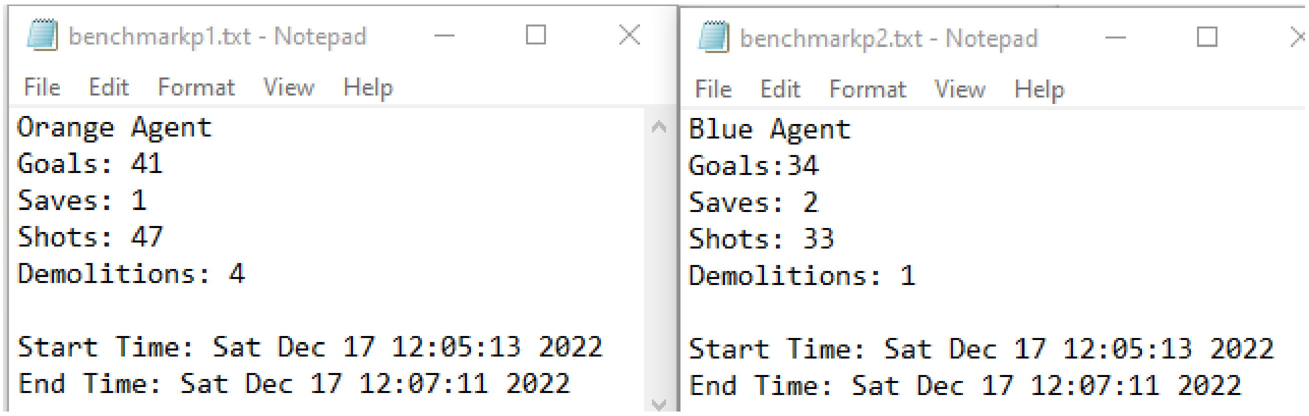


Figure 10: "exBot" delivering the ball into the opponent's net

The fifth and final benchmark results, as seen in figure 11 below, shows a dramatic increase in all stats. At this point, the agent had much more effective gameplay and is now able to keep the ball close and dribble it down the field into the opponent's net. The agent also began shooting the ball with more power and accuracy by flipping into the ball rather than simply driving into it. These skills allowed for consistent scoring, as seen in the 75 goals scored in the final benchmark. The agent also became much more adept at defending and was now able to record several saves, a stat that was previously quite elusive (See figure 12).


benchmarkp1.txt - Notepad
File Edit Format View Help
benchmarkp2.txt - Notepad
File Edit Format View Help

```
Orange Agent
Goals: 41
Saves: 1
Shots: 47
Demolitions: 4

Start Time: Sat Dec 17 12:05:13 2022
End Time: Sat Dec 17 12:07:11 2022
```

```
Blue Agent
Goals:34
Saves: 2
Shots: 33
Demolitions: 1

Start Time: Sat Dec 17 12:05:13 2022
End Time: Sat Dec 17 12:07:11 2022
```

Figure 11: Benchmark results for "exBot" with 400 hours of training

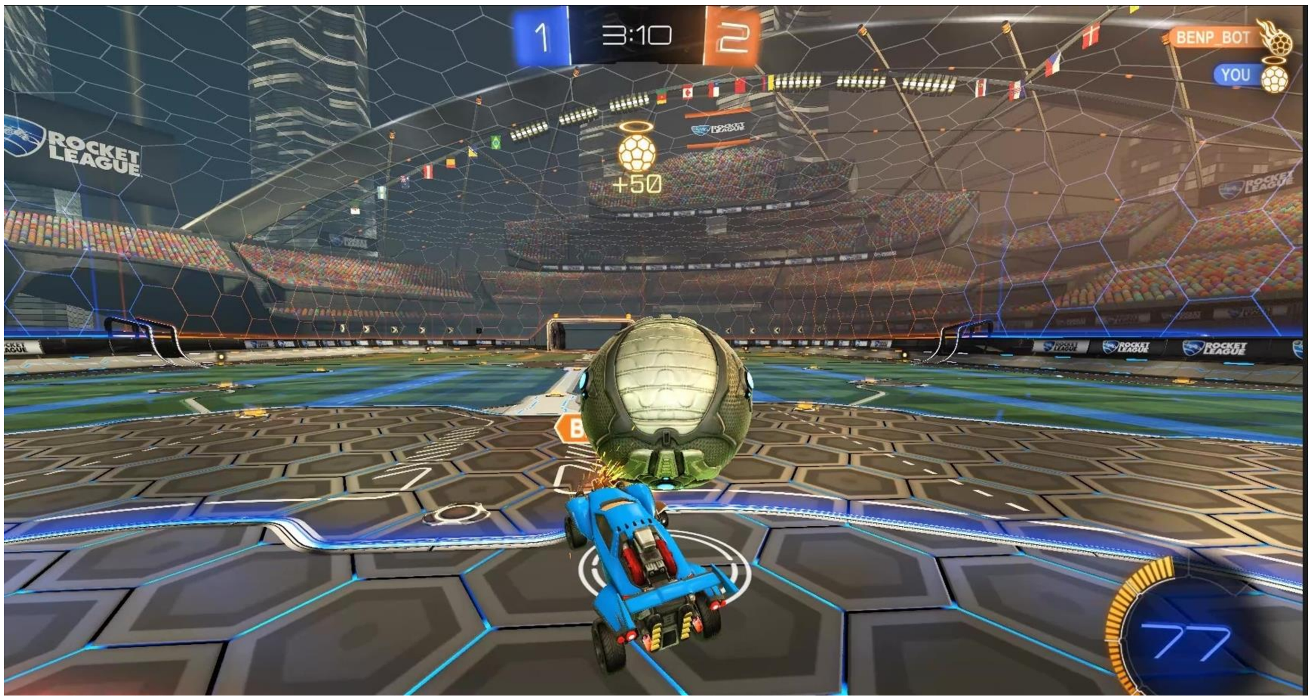
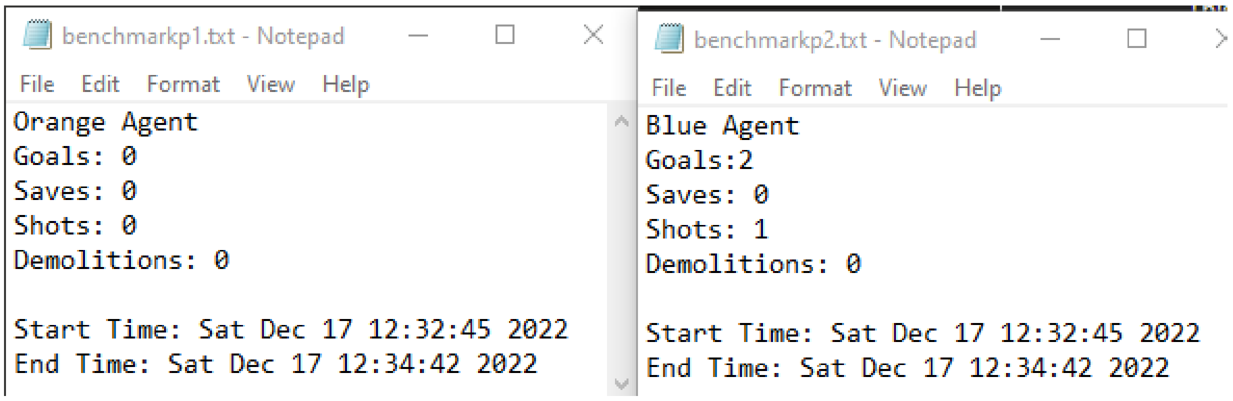


Figure 12: "exBot" getting a save

Bot #2

The second bot, or "demoBot", was created with the goal of getting as many demolitions as possible. This meant it would have to maintain significant speed in addition to directly impacting the opponent's car. This agent trained for approximately 100 hours, or 15,000 hours of in-game time.

The first benchmark results show 2 goals scored and no demolitions (See figure 13). While the agent has no reward involving scoring, the random nature of the agent at this stage led to goals being scored. At this point, not enough training had occured for the agent to be successful in achieving demolitions.

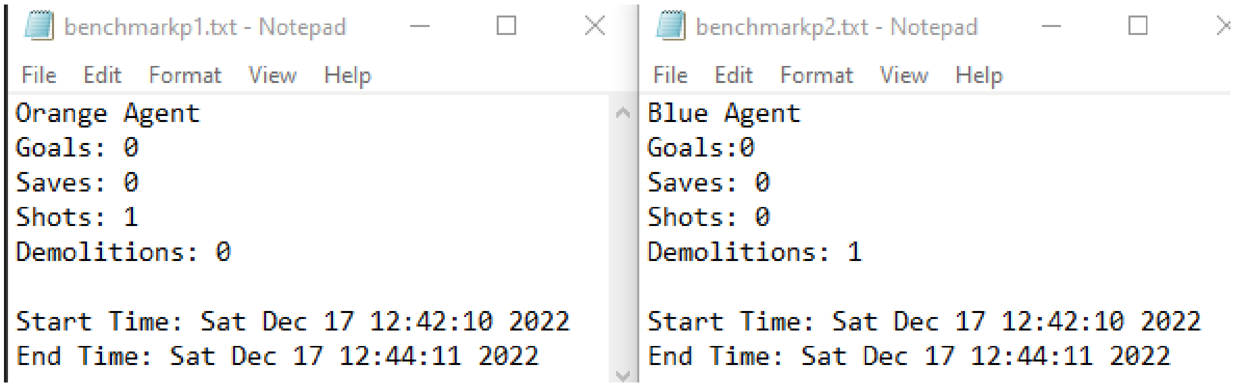

benchmarkp1.txt - Notepad
File Edit Format View Help
benchmarkp2.txt - Notepad
File Edit Format View Help

```
Orange Agent
Goals: 0
Saves: 0
Shots: 0
Demolitions: 0

Start Time: Sat Dec 17 12:32:45 2022
End Time: Sat Dec 17 12:34:42 2022
```

```
Blue Agent
Goals:2
Saves: 0
Shots: 1
Demolitions: 0

Start Time: Sat Dec 17 12:32:45 2022
End Time: Sat Dec 17 12:34:42 2022
```

Figure 13: Benchmark results for "demoBot" with 50 hours of training

The second and final benchmark, shown in figure 14, displays the results of a 30 minute match with 100 hours of training. At this point, the agent had begun navigating the pitch quickly, maintaining the maximum speed for the majority of the test. While the agent was not successful at tracking the opponent, this speed allowed the bot to get one demolition (See figure 15).


benchmarkp1.txt - Notepad
File Edit Format View Help
benchmarkp2.txt - Notepad
File Edit Format View Help

```
Orange Agent
Goals: 0
Saves: 0
Shots: 1
Demolitions: 0

Start Time: Sat Dec 17 12:42:10 2022
End Time: Sat Dec 17 12:44:11 2022
```

```
Blue Agent
Goals:0
Saves: 0
Shots: 0
Demolitions: 1

Start Time: Sat Dec 17 12:42:10 2022
End Time: Sat Dec 17 12:44:11 2022
```

Figure 14: Benchmark results for "demoBot" with 100 hours of training

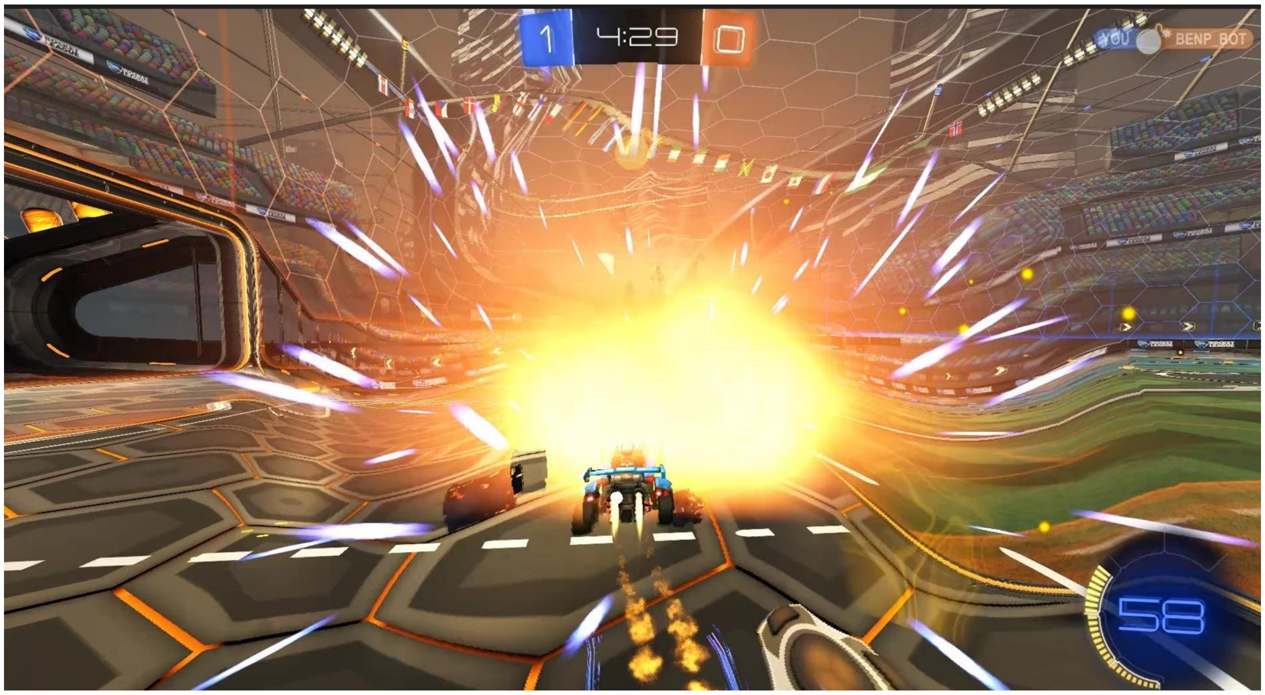


Figure 15: "demoBot" accomplishing a demolition

Discussion

This project required me to learn and use several new skills as I interacted with various libraries, softwares, and applications. This was my first experience using python, so I had to learn the syntax of the language in addition to in the installation and use of several libraries and package dependencies using pip. I had to utilize elements of Stable Baselines 3, a library for reinforcement learning algorithms, RLGym, the python API for the Rocket League gym environment, and RLBot, the software platform for testing and playing against various bots [6]. Throughout the course of the project, I gained significant knowledge of neural networks, specifically reinforcement learning. I also improved my ability to research topics and troubleshoot errors.

I was pleased with the steady progress that could be seen in the learning process for the “exBot”. While the learning was slow at first, it was exciting to watch the agent develop new skills over time. Despite having little training time, the “demoBot” still showed visable progress in moving around the pitch and would have continued to improve with time.

There were several issues that appeared during the initial setup of this project, mostly due to missing dependencies and coding bugs. While these first few problems were shorly fixed, they delayed the start of the training process. This exaggerated the biggest limiting factor for this project, which was the time available. It took several weeks to research topic possibilities and sufficiently narrow the focus. Additional time was consumed in researching reinforcement learning and completing a simplified version of what is now the main file for each bot. This all meant that the training started rather late into the semester. This was an issue because the training takes many hours, so the “demoBot” did not recieve as much training time as I would have liked. This also meant that I did not have enough time to experiment with reward functions and additional bots.

Knowing what I do now, I would have done more to try and start the learning process much sooner and I would have tried to utilize all downtime on my PC for training. This would have allowed for better results for the demoBot, as well as time to explore additional reward function combinations and bot types. I also might have looked into training my network using outside resources such as Google Colab.

Conclusion and Future Work

This project has been successful in creating and training two agents in Rocket League. These two agents have displayed consistent advancement over time through their play and benchmark scores. This improvement shows that the neural networks and reward functions are working properly and are able to produce specialized agents that continuously improve at accomplishing their tasks.

With additional time, the two existing agents would be able to continue learning, becoming more advanced and efficient. The skill ceiling for these bots is substantially higher than their current state, so many additional hours of training is required to realize the potential of this network. Also, the reward functions are an area for improvement. With more time, different combinations of rewards could be tested, as well as creating custom rewards in an attempt to produce even better results.

Lastly, more types of specialized agents could be created and trained. This project explored an agent that aimed to outscore the opponent to win games and an agent whose goal it is to get demos on the opponent. There are numerous possible objectives for new agents, such as a bot that dribbles the ball on its car, a bot that stays as far away from the ball as possible, a bot that tries to keep the ball as far above the ground as possible, etc. Each of these would require a custom reward function, potentially with customized individual rewards, as well as many hours of testing and training. With additional time, however, each of these is certainly a possibility.

References

[1] *RLBot*. [Online]. Available: https://rlbot.org/. [Accessed: 16-Dec-2022].

[2] "The Rocket League Gym," *RLGym*. [Online]. Available: https://rlgym.org/. [Accessed: 16-Dec-2022].

[3] R. Arild, D. Downs, M. Joling, and L. Emery, "Necto," *GitHub*. [Online]. Available: https://github.com/Rolv-Arild/Necto. [Accessed: 16-Dec-2022].

[4] Impossibum Creations, "Rocket League Reinforcement Learning Bot Quick Start Tutorial Guide," *YouTube*, 18-Feb-2022. [Online]. Available: https://www.youtube.com/watch?v=C92_UFZ1W-U&t=1279s&ab_channel=ImpossibumCreations. [Accessed: 17-Dec-2022].

[5] "Custom policy network," *Custom Policy Network - Stable Baselines3 1.7.0a5 documentation*. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html. [Accessed: 16-Dec-2022].

[6] Impossibum Creations, "Rocket League Reinforcement Learning Bot Quick Start Tutorial Guide #2 - playing with your bot," *YouTube*, 05-Mar-2022. [Online]. Available: https://www.youtube.com/watch?v=DXH8zBepRmY&t=300s&ab_channel=ImpossibumCreations. [Accessed: 17-Dec-2022].

Appendix

This project requires the following to run:

Python 3.9.0

Python packages:

gym 0.21.0

keras 2.10.0

numpy 1.21.3

rlbot 1.67.1

rlgym 1.2.0

rlgym-compat 1.1.0

rlgym-tools 1.8.0

stable-baselines3 1.6.2

tensorboard 2.10.1

tensorflow 2.10.0

torch 1.12.1

CUDA 11.8, tutorial here:
https://www.youtube.com/watch?v=cL05xtTocmY&ab_channel=NVIDIADeveloper

Rocket League, installed and updated through the Epic Games Launcher

RLBot, for the agents to play outside of a training environment, tutorial for installation:
https://www.youtube.com/watch?v=oXkbizklI2U&ab_channel=ddthj