– Independent Work Report Spring 2020 –

# Team Rocket: Investigating Inter-Agent Communication and Machine Learning in Rocket League Bots

Natalie O'Leary
Advisor: Ryan Adams

May 12th, 2020

## Abstract

*Artificial intelligence is a growing field in the realm of video games, however many intelligent agents are still lacking in the area of communication. This paper explores various methods of communication and teamwork among artificially intelligent agents in the game Rocket League from simple inter-bot messaging to a shared controller for several bots. It was hypothesized that the ability to communicate between agents would make teams more successful in matches of 2vs2 or more. This hypothesis is supported by the results of this study.*

## 1. Introduction

Machine learning and artificial intelligence are growing fields in the study and production of video games of all kinds. Most notably, in an attempt to further immerse players in the experience, video game producers are creating more and more NPCs, or non-player-characters, who act like real humans, through the use of artificial intelligence. However, machine learning in video games is perhaps even more prevalent than most would think, as its used not only to create humanoid characters, but also to create reactive environments, viable competitors in racing games or online strategy games such as chess or Settlers of Catan [3], and reactive story lines in narrative games. The scope of this project focuses on the use of machine learning and artificially intelligent game playing agents in the game Rocket League [5]. Rocket League is a multi-platform game that allows players to compete against each other in a game similar to soccer, except that the arena is circular, and the players, which are cars, can drive up the walls. Players can play in either single player or multiplayer games with up to six players on each team. A player can also

opt to hone their skills by playing in a single player match against artificially intelligent rocket league bots with varying degrees of skill. The presence of a non-human game playing agent already in the architecture of the game was very promising and encouraging for the area of focus and study for this project.

Rocket League is a such a conducive environment for creating intelligent agents, that players have actually taken to creating artificially intelligent bots for fun, and pitting these bots against each other, rather than actually competing and playing in games themselves. The popularity of this tactic lead to the creation of RLBot, an open source, collaborative framework for creating Rocket League bots, complete with an interactive GUI that allows developers to pit bots against each other for testing purposes and in regular online tournaments. The RLBot community is a thriving one, who have mastered the art of creating competitive and effective Rocket League Bots, however, one of the areas in which these bots typically lack is in the idea of inter-bot communication. Most development in the community up until this point has been focused on individual characteristics in bots, and creating the most effective single player agent, that can at best, likely still perform individually when there are other bots present working toward the same goal, but with no real collaboration. And while machine learning has been introduced in some cases, it is really not very prevalent in most of these bots, which typically function as complex state machines, which calculate the next best move according to the current environment, rather than learned history.

Thus, the object of this project was to create RLBots that were designed with inter-bot communication in mind, and furthermore to utilise machine learning to increase the efficacy of these bots, and their ability to communicate and succeed in a competitive environment. This was done by creating several different bots, with different goals and attributes in mind, some that were trained using ML, and others that are simply state machines, but that were designed to be able to execute more team-like behavior, such as passing and zone defending. Because of the complexity of the game, and the difficulty of creating truly humanoid communication and strategy in the game, these experiments were done in isolation, identifying single problems or

areas for improvement, which a given bot could improve upon. Thus, the state machines were used to implement communication bots, while machine learning was used to train a simple bot to learn to be an effective goalie. The results were quantitatively analyzed using Rocket League's scoring system, which automatically scores the success of a bot based on metrics such as touches on the ball, goals, assists and clears. The prediction is that bots that have been programmed with communication in mind will function better in team settings against otherwise comparable bots, while the goalie bot will have an increased percentage of saves as compared to a comparable bot that has not been trained.

## 2. Problem Background and Related Work

Artificially intelligent agents in games are not entirely new, with Deep Blue – the first chess playing bot to beat the reigning world champion – having won its first world champion match in 1996 [2]. However, studies into the capabilities of collaborative agents in multiplayer games are only now beginning to emerge. While there are no publications that directly deal with the creation of collaborative Rocket League agents, this section attempts to delve into some of the relevent publications that informed the majority of my research.

The first important publication to note is a study done by O. Vinyals, I. Babuschkin, W.M. Czarnecki et al. called "Grandmaster level in StarCraft II using multi-agent reinforcement learning" [1]. This project is groundbreaking in its successful attempt to create a Grandmaster level bot that plays the game Starcraft II. Starcraft II is a vastly more complicated game than Rocket League, with the ability to play as three completely different races of characters, combined with the fact that Starcraft II is a long, war-style game, as opposed to Rocket League's relatively simple, soccer-like game mechanics. The training for the agents in this machine is a very complex process, but it employs two main steps, using both supervised learning and reinforcement learning. The agents are first trained in a supervised learning setting that uses publicly available data from human replays of Starcraft II to create a large data set of different strategies and their success rates. These strategies were created by predicting actions conditioned on each

of the parameters in the neural network for the project. This idea of utilizing replays became essential to the success of my project as well, and luckily there is an abundance of Rocket League replay data that is also open and available to the public. The second step of the training process utilises a reinforcement learning algorithm that the authors describe as "similar to advantage actor-critic," [1] which, as its name implies, involves an actor that determines the agent behavior, and a critic that assesses the efficacy of the action taken. However, their approach differs in that they still used replayed experiences, and thus their algorithm was as a result, off policy. Their agent, Alphastar, has been incredibly successful, achieving Grandmaster status within Starcraft players, meaning that the agent defeats 99.8% of human players. [1] While the complexity of this multi-year project is far beyond the scope of my project, I did find their methods useful, with regard to their use of both reinforcement and supervised learning to train the same agents, as well as their use of replays. This study is also incredibly important, as it was one of the first, successful autonomous agents to defeat master level human competitors in a multiplayer game.

Another useful work to note is a master's thesis entitled "Team Behavior of Artificial Intelligence Bots in Games" written by Yannick VerHoeven. [8] This paper was written as more of a theoretical study than an experiment of any kind, so the author did not actually create any Rocket League bots over the course of his research. However, he delves into the mechanics and theory behind bot machine learning and collaboration, through careful analysis of existing bots and tournaments, as well as an analysis of the game play and output data itself. Though this paper is over 100 pages long, and gives in depth explanations of a variety of artificial intelligence and rocket league phenomena, I will only summarize the sections that pertain most closely to my research. The first important topic that this paper covers is that of inter-bot communication specific to Rocket League. VerHoeven correctly points out that the possibility for teams of AI bots that function at a higher level than human players is very high, due to the fact that the bots can send out and receive messages as often as 60 times per second with very little cost to the agent itself. This can all be done using a built in function of the game called QuickChat, that allows players to choose messages from a list of acceptable messages to send to all bots at any point during

a game. While humans would have to select a phrase and read and comprehend a message from another player, taking up seconds of game time, a bot can do all of this in milliseconds, providing a greater opportunity for inter-car communication within a team. While this discovery was very encouraging for my bot communication research, VerHoeven also expresses a less optimistic view of the prospect of using machine learning in the context of a full Rocket League game, stating, "optimistic complexity estimations put the number of states at $(108x50)^{23}$, and the number of policies at $1000^{3000}$, per agent per halftime." [8] This is further complicated by the relative lack of labels and indicators in Rocket League, because any number of sequences can lead to a goal, and from there it is difficult to determine exactly which actions most directly led to this outcome. His paper identifies key areas in which bots can be taught via reinforcement learning however, including a successful bot that has the ability to juggle a ball in the air. [8] This paper, though largely theoretical, gave a lot of direction to my experiments, as well as informing the scope of my project.

The final relevant work that I will discuss is actually an RLBot which has an open GitHub repository called Saltie. [7] Saltie is both a functioning RLBot that uses deep reinforcement learning and neural networks to learn how to play Rocket League, and also a framework of tools for other programmers to use to train RLBots. The Saltie Bot, much like the AlphaStar bot for StarCraft II is trained using supervised learning on a set of human generated replays from games of Rocket League. Their framework provides a way to integrate model based learning into a RLBot state machine, with the functions they have created to select a model, and generate input and output fitted for that training model. However, the most important contribution that Saltie has made to the world of RLBot is the creation and maintaining of a site called calculated.gg. This site harbors over 8 million human rocket league replays that can be processed and analyzed using an analysis library that they have created called carball. [7] Carball extracts the important information from rocket league replays that can then be used to train bots. This information comes in the form of game tick packs, which are outputted by the bot up to 60 times per second, containing all of the essential information about the state of the game at that point, including

the positions, velocities and angles of all of the cars in play, as well as the ball. This research was incredibly helpful, not only in directing my research, but also in providing useful tools that I could utilise in order to create bots of my own. However, Saltie does not deal at all with collaboration between bots, and in fact functions best when trained on 1v1 matches.

## 3. Approach

### 3.1. RLBot Structure

It's important to begin with a description of the RLBot framework, as it is a tool that allowed for all subsequent work done in this project. RLBot is an opensource, collaborative framework that is publicly available on GitHub. The framework consists primarily of a detailed API that bot creators can use to create AI bots that play the game Rocket League, and an interactive GUI that uses the Rocket League app on Steam to host bot vs bot matches.

**3.1.1. Architecture** While RLBot contains frameworks and tools for programming in many languages, I only used, and will therefore only be focusing on the structure of, their Python bots. The basic structure of the bots relies on the idea of returning a controller to the BotManager, which actually executes the commands that will move the bot on Rocket League. The controller contains information about what direction the bot should face, what angle the bot should turn at, and the velocity of the bot. A game of Rocket League can run at anywhere between 30 and 250 fps, which the player can set. An RLBot returns a controller to change the movement of the bot 60 times per second, through the getOutput() method, that is built in and required in all RLBots, as the BotManager calls this method directly. All RLBots are primarily controlled through a file called bot.py which contains the getOutput() method. Programmers can use the many RLBot tools to determine the position, velocity, yaw and angle of their own bot, other bots on the field, the ball, as well as game constants such as goal positions and field boundaries. These variables are conveyed to the bot through a game tick packet, which contains all of this necessary game information every time getOutput() is called. This allows a programmer to ascertain important game state variables, and then analyze by calculating values such as angle between the bot and

the ball, the bot and the goal and other such indicators of the next move. GetOutput() then returns a controller which determines the bots next move in the game. A diagram of the Python bot architecture is shown in Figure 1.

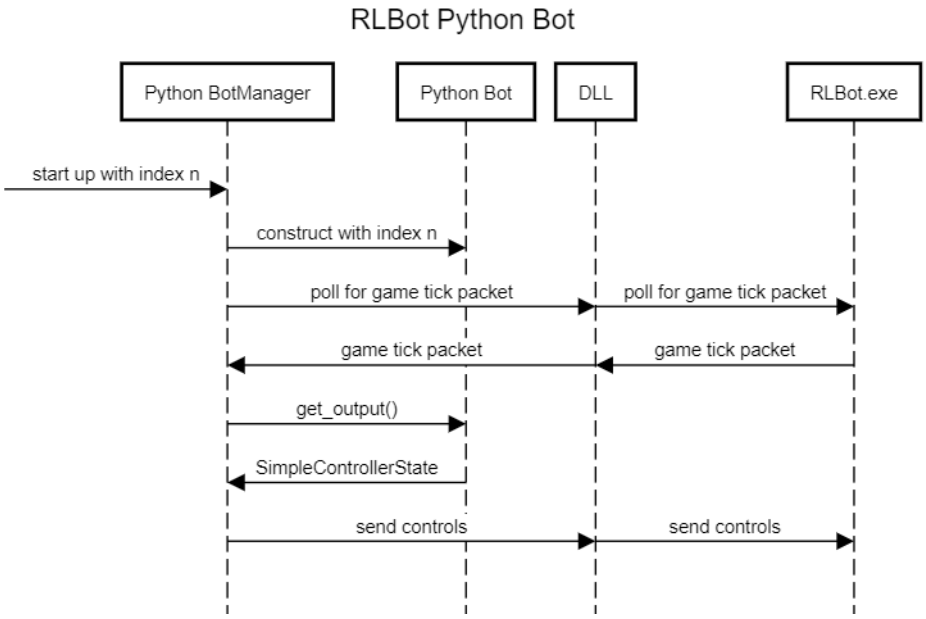


**Figure 1: RLBot Architecture [5]**

**3.1.2. GUI** The GUI for RLBot is made up of two major components, the RLBot interface that allows players to select bots to pit against each other, start and stop matches, and select game play modes, and the actual Rocket League interface, where the matches are hosted. Users of RLBot can run the matches with their own bots, as well as 27 bots that come with the download of RLBot that have been created by other users. As in the normal game of Rocket League, you can select up to 6 bots to play on one team, though matches must be balanced with equal bots on each team. However, importantly, RLBot allows you to select the same bot as many times as you want, so you can pit your bot against itself, or pit a team of all the same bots against another team of all the same bots. This is an incredibly valuable feature in measuring the success of certain bots, as you can measure its behavior against itself, and ensure some control variables in your measurements. This interface can be seen in Figure 2.

The second aspect of the RLBot GUI is actually hosted in Rocket League itself, and the RLBot framework gives you the ability to open and run a match between artificially intelligent game

playing bots on Rocket League. The RLBot framework prevents users from pitting their bots against human players, or really playing Rocket League outside of the sandbox setting that RLBot provides, so that players do not abuse the framework to farm items in the game. However, RLBot extends Rocket League's capabilities for bot scenarios, as the framework includes a series of training scenarios that repeatedly place the bot in a certain environment such as in front of the goal, or at kickoff repeatedly, lasting only a few seconds, so that users can train their bots using machine learning, to be particularly effective at a specific part of the game. RLBot achieves this by manipulating the Game State: shortening the game to last only a few seconds, setting the initial location, angle, and velocity of the bot, and the ball, and running these games on loops.

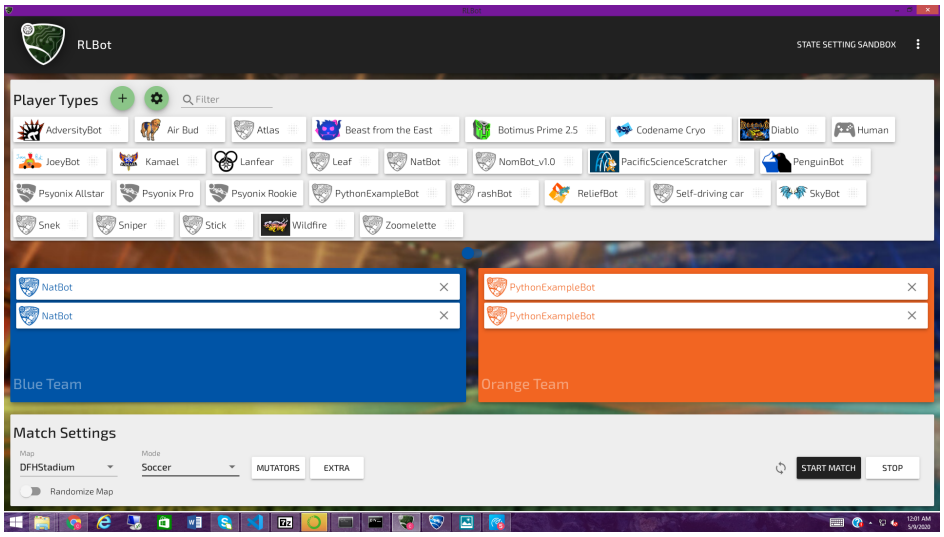


**Figure 2: RLBot GUI**

### 3.2. Game Grid

The layout of the game determines how all of the physics of bot and ball movement is calculated, so it is important to establish here. Figure 3 shows the dimensions and layout of the arena, which notably do not have units. The arena is laid out such that cars can drive up the curved walls up to a certain point. The x and y coordinates are simply extended beyond the floor of the arena, on a curved grid, so the z coordinate is not necessary to direct a bot up the walls. However, the z coordinate is used to measure ball and bot height while in the air, as well as the height of the goals themselves. Regardless of what team a given bot is on, the orange side of the arena

is always the positive half of the y coordinates, and, facing the orange side, the left direction is the direction of more positive x values. Thus, bots can always aim toward (0, -5000), and know that they are aiming toward the blue goal, so certain aspects of aiming can be hard coded into the bots. The end of the blue side of the field is always at (0, -8064), and the other end on the orange side is always at (0, 8064). The depth of the goals is also an important aspect of the game, because, like the walls, bots can also drive up and around the walls of the goals. Each goal is 880 in depth, and 642 in height. [6] The game always begins with a kickoff, with the ball located at (0,0). RLBot also provides key constants, that are necessary for calculating the physics of the bots and ball in this virtual world, including an altered gravity constant, and the maximum speeds of the bots that can be seen in Figure 4.

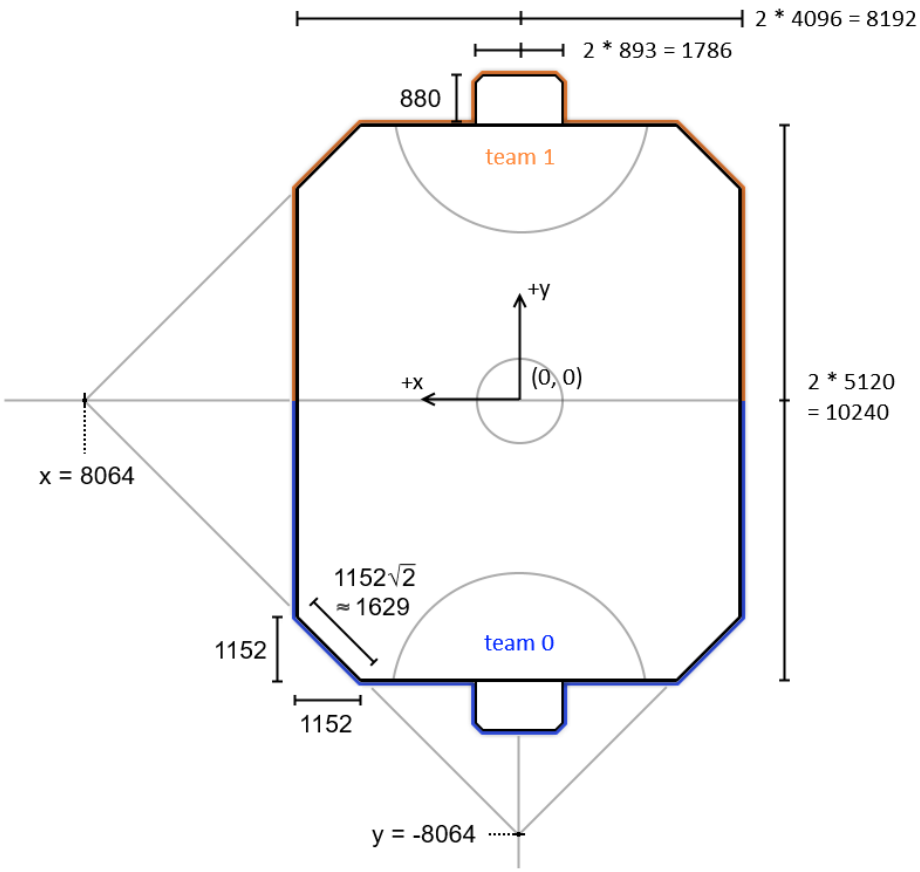


**Figure 3: Rocket League Layout**

**Conversion:** 1 uu = 1 cm (e.g. 2778 uu/s = 100 km/h)

- Gravity: 650 uu/s^2
  - "Low" mutator: 325
  - "High" mutator: 1137.5
  - "Super High" mutator: 3250
- Max car speed (boosting): 2300 uu/s
- Max ball speed: 6000 uu/s
  - "Slow" mutator: 1500
  - "Fast" mutator: 9000
  - "Super Fast" mutator: 15000
- Supersonic speed threshold: 2200 uu/s
- Max driving speed (Forward and backwards) with no boost: 1410 uu/s
- Car mass: 180.0 (unit is arbitrary)
- Ball mass: 30.0
- Boost consumption rate: 33.3/s
- Acceleration in the ground:
  - due to boost: 991.666 uu/s^2
  - due to braking (any amount): -3500.0 uu/s^2
  - due to slowing during zero-throttle coasting: -525.0 uu/s^2
- Ball coefficient of restitution: 60% (it loses 40% of the component of its velocity that's toward the surface)
- Jumping from flat ground
  - This only applies to the first jump, not the double jump
  - Jumping results in acceleration towards the roof of your car, whatever its orientation.
  - On the first physics tick, an instantaneous velocity increase of ~291.667 uu/s
  - As long as the button is held (up to 0.2s), continuous acceleration of ~1458.333374 uu/s^2 (not including gravity)
- Maximum car angular acceleration:
  - Yaw: 9.11 radians/s^2
  - Pitch: 12.46 radians/s^2
  - Roll: 38.34 radians/s^2
- Maximum car angular velocity: 5.5 radians/s
- Maximum ball angular velocity: 6.0 radians/s
- Turn radius can be found from the following approximation of the curvature:

**Figure 4: Physical Constants in Rocket League**

### 3.3. Bot Types [5]

In order to isolate key characteristics and strategies in a very complex, game, I split my bot development into two different types of bots: communication bots and goalie bots. The communication bots focus on the communication between artificially intelligent agents, and how communication can increase collaboration and team behavior in matches. The goalie bot was created as an attempt to isolate a specific behavior in a bot that could be improved upon in isolation using machine learning. While there are many ways in which communication between

bots could improve team behavior, such as passing, assisting, defending and hindering the opponent, I chose to focus on the separation between offense and defense, so I could create bots that can dynamically determine through communication which bot will pursue the ball, and which will defend their own goal. This objective works well with the design of my other bots, the goalie bots, as these two bots can theoretically be combined with further research, to create bots that not only trade off between offense and defense, but that are also trained to be particularly effective at defending their goal when on defense.

## 4. Implementation

### 4.1. Bot Mechanics

The basic structure of both the goalie bots and the communication bots is the same, and the majority of their function occurs in the getOutput function. These bots all have the same basic structure, so as to facilitate comparison and ranking among them. Each of the bots have their throttle variable set to 1. This variable controls the speed of the car, and can only be set between -1 and 1, with negative numbers indicating backward direction, and higher magnitude numbers indicating faster speed. After this, each time getOuput is called, several key values are calculated and used to determine what to return as a controller.

First, the position of the bot relative to the ball and the opponents goal is ascertained using information from the given game tick packet, and thus is used to determine where the bot should aim. If the ball is between the bot and the enemy goal, then the bot will aim toward the ball, but if not, the bot will aim toward its own goal, so as to get on the other side of the ball and defend, or possibly get a better offensive angle. The way that aiming is done is through a separate aiming function, which calculates the angle at which to aim by first calculating the arctangent between the target (either the goal or the ball), and the bot to find the angle between the two, and then subsequently subtracting from that the yaw of the bot. The bots direction and steer is then adjusted in that direction.

Next, the distance from the bot to the ball is used again to determine whether or not the bot

should "dodge," which, contrary to its name, actually means do a sort of flipping jump, that can be used to strike the ball more forcefully than simply driving into it. The issue with the timing of the jump, is that once the bot is in the air, it no longer has control over it's velocity and direction for the entire time that the bot is in the air. Additionally, the jump has to be timed such that the bot is consistently striking the ball in the middle, especially given that the game can run at as many as 200 fps, but the getoutput function is only called at most 60 times per second. Therefore, if the bot is instructed to jump too far away from the ball, it may simply miss, as the ball and the bots position are updating as the bot is jumping, but also if the bot is too close to the ball when instructed to jump, it may actually end up coming in contact with the ball before even jumping, because the bot is still moving forward as its determining what to do, and the jump will be ineffective and too late. Thus, with some trial and error, I was able to determine that the best distance in Rocket League units from the ball at which to jump is 400. However, once a bot has jumped, it remains in the air for .2 seconds, and thus cannot jump again right away, even if it is still within close enough range of the ball, so this must be taken into account as well when determining whether or not the bot should dodge.

The distance from the ball is also used to determine whether or not the bot should use the boost function, which allows a car to use its "rocket power" to drive at 2300 units/second. Boost is consumed at a rate of 33.3 percent per second, so it cannot be used all the time. [6] Additionally, if the bot is moving faster, it has less precision of movement using the getOutput method, so using the boost when trying to strike the ball is unwise, as the bot is likely to miss. Thus, the boost tool in RLBot is only used when the bot is far enough away from the ball. Since the arena is 8192x10240x2044, Rocket Leauge analysts have guaged this distance to be about 1500, so that is the distance from the ball at which my bot will use the boost. [6] My bots also boost at the kickoff, which is achieved by having boost turned on so long as the ball is at (0,0), since the ball will remain there until a bot touches it at kickoff, and will presumably not rest there for longer than a split second at any other time in the game.

These are all of the key features and considerations of all of my bots, though the individual bots have some modifications that make them more suitable for their intended goal.

**4.1.1. QuickChat Bots** The QuickChat bot was the first bot that I attempted to add communication to, so naturally it was done through the simplest channel: the games built in chat function. QuickChat is a tool of Rocket League that allows players to communicate to all other player in the match. Players can choose from a list of preset messages to send, and the messages show up in a stream at the top left corner of the game, with the message and the player who sent it. Although QuickChat only allows the use of preset messages, RLBot developers have created quite a few extra messages that can be used in a bot game, and even without these, there are plenty enough messages to send to allow adequate communication between bots. The delegation of defense and offense between bots is once again determined via distance from the ball. Whichever bot gets within 400 units of the ball – which is also the distance at which a bot will jump to hit the ball – will use the QuickChat function, through the RLBot QuickChat library, to send out a message to all other bots saying "I got it!." The bots also have a condition in their getOutput function that states that, if they receive an "I got it!" message from another bot on their team, all other directions will be abandoned, and the bot will drive toward its own goal. Because the getOutput function is called so often, and the offensive bot will be actively pursuing the ball the whole time that it is on offense, the "I got it!" message is sent continually, at about 60 times per second, give or take a few times that the ball may be too far from the bot, so the defensive bots will continue toward their own goal, and remain their to stand defense. However, if at any point the ball is far enough away from the offensive bot, the defensive bot will not receive these chats anymore, and will pursue the ball again. If that bot should get close to the ball first, their positions will switch. At this point, these bots function with two bot teams in mind, as its not logical to have all other bots defend the goal while only one of them is on offense.

**4.1.2. MatchComm Bots** The MatchComm tool is a more complex form of communication than the QuickChat function, because it doesn't have the same restrictions as the QuickChat function, and bots can broadcast whatever message they want to all other bots. This allows for more

complex communication, as bots could broadcast exactly where they are going, or where they would hit the ball toward, which could allow for more complext team functions such as passing and double teaming, as well as more effective covering of the arena. However, unfortunately within the bounds of this semester, such optimizations were outside of the scope of this project, and so other than the use of a different channel, the MatchComm bots function in the same way that the QuickChat bots do, by simply sending out a message when one bot gets within 400 units of the ball, that tells all other bots to get back on defense.

**4.1.3. Hivemind Bots** The hivemind bots go a step further in terms of teamwork amongst bots, from simple communication to an actual shared "mind" which is essentially one program that controls all of the bots on a team at a time. To create hivemind bots, I used another framework from GitHub, this one created by a user of RLBot who goes by ViliamVadocz on GitHub. [9] The framework he created allows programmers to override the getOutput function in the bot.py file, and instead have all of their bots be controlled by a single file called hivemind.py. Because RLBot automatically calls bot.py for the bot inputs, the bot.py for the hivebots calls a helper process, which is hivemind.py that handles all of the bots functions, and bot.py returns what the helper process returns, rather than directly returning a controller. The hivemind file treats the cars on the team as a Drone object, which is similar to the way bots are represented in typical RLBot bots, with the addition of an index, to determine which of the drones being controlled by the hivemind this bot is. [9] The hivemind bots have the same basic programming as the other two communication bots, however, rather than communicating to each other to determine offense and defense, all of this is done by the hivemind. The hivemind file determines which of its drones is closer to the ball at every game tick, and then directs that drone toward the ball, and the other drone toward the goal they are defending. The benefit of the hivemind is that a single entity has access to all information about all bots on a team, as well as the ability to control all of the bots. This strategy may prove to be more effective, as there is no need to set a specific distance from the ball at which to send a signal for one bot or the other to go after the ball, so the switches between offense and defense are quicker and more dynamic. Additionally, with the

chat methods, if neither bot is close to the ball, then neither will be sending out the "I got it!" message, and both will be pursuing the ball and the goal will be left untended. It's likely that if neither of my bots are close to the ball, that the opponent cars have the ball, so it is a very bad time for the goal to be left untended. With the hivemind bots, this won't happen, as the code only allows for one bot to go after the ball at a time, but if one bot is in the goal, once the ball approaches, it will immediately gain control and pursue the ball, hopefully clearing it out of its team's side. This method is also focused on teams of two, and the hivemind bots will actually only function if there are exactly 2 players on the team, due to the need to assign an offensive and a defensive bot, and the lack of provisions in place in the program that I created for a third through sixth bots.

**4.1.4. Goalie Bot** This bot, though it has the same basic framework as the other three bots, operates in a very different environment. The intended outcome of this bot was to utilize machine learning to train a bot to be a more effective goal keeper. However, this pursuit was undertaken despite warnings from the RLBot GitHub itself which says "The game can't be run headless or sped up beyond a certain point, so the necessary training time is unattainable at the moment." [6] Thus, this bot was largely unsuccessful, however, it's important to discuss my methods here. The RLBot framework provides tools that players can use as "training exercises" which place a bot in a short simulated situation such as a kickoff or a penalty kick over and over again, placing the bot or the ball in slightly different starting positions each time. While I had believed that these indicated a possible opportunity for machine learning within the framework, the real intended purpose of these scenarios is for players to manually adjust their code while watching the training exercises, because RLBot will automatically update your bot to match code as you save it, even as the game is still running. However, the training scenarios do allow you to capture all of the game tick packets from the entirety of each training exercise, and provides a method through which to pass or fail the bot on that pass of the exercise. In the case of the goalie exercises, the bot passes if the ball is heading away from the goal, indicating that the bot successfully cleared it, or if the exercise times out without the ball having entered the goal.

Otherwise, the bot fails, as the ball has entered the goal. I was able to collect this information using the logger method in RLBot, and vectorize it using sci-kit learn's Count_Vectorizer tool, with the pass and fail markers being the labels for the vectors. [4] I was then able to fit these vectors to a linear regression, however, I was not able to reintegrate this information into the function of the bot itself. I had originally believed that, since the linear regression contains the sequence of actions for each trial in a vector that is labeled with either pass or fail, that with each call of getOutput, that the bots could use the linear regression to identify the next move that would most likely lead to a passing grade. However, this attempt proved to be unsuccessful, as it was not fast enough for the bots to be able to actually function this way, as the getOutput function is called so often during a game.


**Figure 5: In Game Score Tally [5]**

### 4.2. Evaluation Metrics

Rocket League has individual scoring metrics for all players, as well as the overall outcome of the match for each team, win or lose. The game keeps track of each cars overall score, with each

touch on the ball being worth 2 points, each goal 100 points, assists worth 50 points, missed shots worth 20 points, and saves worth 50 points. [5] Players can watch this tally in the corner of their screen for any bot in play as shown in Figure 5. However, there are also many additional actions for which players can earn points that are not stated in the on screen tally, the full list of which can be found in figure 6. I considered wins and losses separately from total points, and used both metrics to do analysis on the success of each bot.

Win - Win a match 1000

Complete Game - Complete a match where you lose 750

MVP - Earn the highest score in a match 100

Goal - Hit the ball into the opponent's goal 100

Aerial Goal - Score a goal from an aerial hit 20

Backwards Goal - Score a goal by hitting the ball driving backwards 20

Bicycle Goal - Score a goal from a bicycle hit 20

Long Goal - Score a goal from a great distance (your own court) 20

Turtle Goal - Score a goal by hitting the ball while upside-down 20

Pool Shot - Score a goal by hitting an opponent into the ball 20

Overtime Goal - Score a goal after the regulation time has ended 25

Hat Trick - Score 3 goals in a single game 25

Assist - Pass the ball to a teammate who scores 50

Playmaker - Score 3 assists in a single game 25

Save - Block a shot on your goal 50

Epic Save - Block a shot on goal that's on the verge of scoring 75

Savior - Block 3 shots on goal in a single game 25

Shot on Goal - Hit the ball towards the opponent team's goal 20

Center Ball - Hit the ball towards the center of the field 20

Clear Ball - Hit the ball away from your own goal 20

Aerial Hit - Hit the ball while in the air (above goal height) 10

Bicycle Hit - Hit the ball by flipping into it 10

Juggle - Hit the ball three times in a row without letting it hit the ground 20

Demolition - Destroy another player 10

Extermination - Demolish 7 players in a single game 20

First Touch - Be the first to touch the ball on kickoff 10

Damage (Dropshot) - Hit a tile with the ball in its first or second phase 10 per panel

Ultra Damage (Dropshot) - Hit a tile with the ball in its third or final phase 10 per panel

**Figure 6: Full Scoring Guide [6]**

## 5. Evaluation

In order to measure the success of each of my bots, I set up a tournament, in which each of the following bots competed in 2vs2 matches:

- QuickChat Bot
- MatchComm Bot
- HiveMind Bot
- Control Bot - No Communication

There are no eliminations in this tournament for the sake of collecting information, so each of the bots I created competed against each other bot, and the RLBot Example Bot, as a control 3 times each. Because the Goalie bot was largely an unsuccessful pursuit, and it is used in an entirely different context than the other bots, I chose not to include it in the tournament, as other than its unsuccessful machine learning attempts, it is essentially the same as the control bot. The

| 1 | QuickChat | | Python Exar | winner |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 386 | 0 | 367 | 234 |
| goals | 2 | 0 | 2 | 1 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 5 | 0 | 2 | 1 |

| 2 | QuickChat | winner | Python Example | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 731 | 401 | 218 | 46 |
| goals | 5 | 3 | 1 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 5 | 3 | 1 | 1 |

| 3 | QuickChat | | Python Exar | winner |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 56 | 16 | 310 | 82 |
| goals | 0 | 0 | 2 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 0 | 0 | 2 | 0 |

| 1 | Hivemind | winner | Python Example | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 258 | 234 | 202 | 128 |
| goals | 1 | 2 | 1 | 0 |
| assists | 1 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 0 | 1 | 1 | 0 |

| 2 | Hivemind | winner | Python Example | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 407 | 150 | 214 | 62 |
| goals | 3 | 1 | 1 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 2 | 0 | 0 | 0 |

| 3 | Hivemind | winner | Python Example | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 583 | 14 | 204 | 142 |
| goals | 3 | 0 | 1 | 1 |
| assists | 0 | 0 | 0 | 0 |
| saves | 2 | 0 | 0 | 0 |
| shots | 2 | 0 | 2 | 2 |

| 1 | MatchComm | winner | Python Example | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 431 | 190 | 344 | 204 |
| goals | 3 | 1 | 2 | 1 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 3 | 1 | 2 | 2 |

| 2 | MatchComm | tie | Python Exar | tie |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 537 | 26 | 511 | 110 |
| goals | 3 | 0 | 3 | 0 |
| assists | 2 | 0 | 4 | 1 |
| saves | 0 | 0 | 0 | 0 |
| shots | 0 | 0 | 0 | 0 |

| 3 | MatchComm | tie | Python Exar | tie |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 152 | 52 | 240 | 188 |
| goals | 1 | 0 | 0 | 1 |
| assists | 0 | 1 | 3 | 1 |
| saves | 0 | 0 | 1 | 0 |
| shots | 0 | 0 | 0 | 0 |

| 1 | MatchComm | tie | QuickChat | tie |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 200 | 138 | 122 | 0 |
| goals | 0 | 1 | 1 | 0 |
| assists | 1 | 1 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 0 | 0 | 0 | 0 |

| 2 | MatchComm | | QuickChat | winner |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 32 | 8 | 136 | 0 |
| goals | 0 | 0 | 1 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 0 | 0 | 0 | 0 |

| 3 | MatchComm | winner | QuickChat | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 290 | 30 | 64 | 32 |
| goals | 1 | 0 | 0 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 2 | 0 | 1 | 0 |

| 1 | MatchComm | winner | Hivemind | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 271 | 106 | 204 | 90 |
| goals | 2 | 0 | 1 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 2 | 1 | 0 | 0 |

| 2 | MatchComm | winner | Hivemind | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 403 | 126 | 332 | 54 |
| goals | 3 | 0 | 1 | 0 |
| assists | 0 | 1 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 2 | 0 | 3 | 1 |

| 3 | MatchComm | winner | Hivemind | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 190 | 172 | 202 | 148 |
| goals | 1 | 1 | 1 | 0 |
| assists | 0 | | | |
| saves | 0 | | | |
| shots | 0 | 2 | 0 | 1 |

Vertical (Value) A

| 1 | QuickChat | winner | Hivemind | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 274 | 158 | 140 | 124 |
| goals | 2 | 1 | 1 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 1 | 0 | 0 | 0 |

| 2 | QuickChat | | Hivemind | winner |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 258 | 160 | 314 | 254 |
| goals | 1 | 1 | 2 | 1 |
| assists | 0 | 0 | 2 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 1 | 0 | 0 | 1 |

| 3 | QuickChat | winner | Hivemind | |
|---|---|---|---|---|
| | bot 1 | bot 2 | bot 1 | bot 2 |
| points | 306 | 132 | 94 | 66 |
| goals | 2 | 1 | 0 | 0 |
| assists | 0 | 0 | 0 | 0 |
| saves | 0 | 0 | 0 | 0 |
| shots | 1 | 0 | 0 | 1 |

**Figure 7: Results of All Games**

results and breakdown of every single one of these matches can be found in figure 7, however, the organization of this data alone is not particularly enlightening, so I have also broken down

these results into some key features. Figure 8 shows the distribution of wins and losses and ties

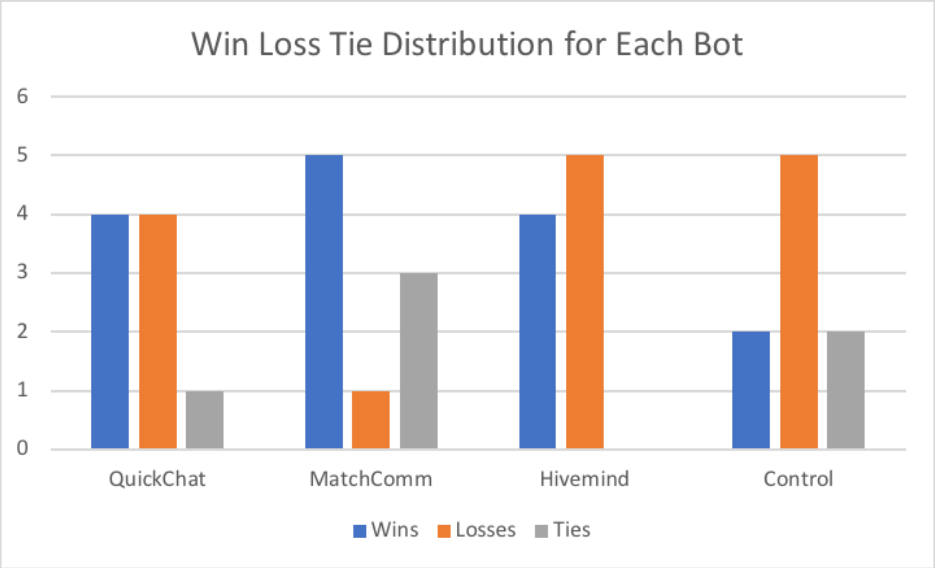


**Figure 8: Total Wins and Losses for All Bots**

for all of the bots. As you can see from this chart, the MatchComm bot team was the only one to win more matches than they lost, however the QuickChat bot, which is the most similar in structure to the MatchComm bot comes in close second. Of the 15 total wins in the tournament, 9 of them, or 60%, of these wins were won by QuickChat and MatchComm combined, where Hivemind and Control make up only 40% of these wins. Interestingly, the MatchCommm and QuickChat bots are also the two that have the most unequal distribution of points between the two bots on the team this disparity can be seen in Figure 9. This is to be expected, as these are the two bots that most directly choose one bot on the team to serve as the offensive bot for the majority of the match, and the way that this is decided is through proximity to the ball. So, if one bot is always closer to the ball, that bot will naturally have a disproportionately high point count as compared to its teammate. This indicates that the strictly delegated offense-defense strategy seems to be the most successful, as Hivemind has a more flexible delegation of offense to defense, with the closest bot at any point to the ball going after the ball, and the Control bot has no delegation of offense and defense at all. QuickChat and MatchComm have the same strategy of getting within 400 units of the ball, and then continually sending out a message to the

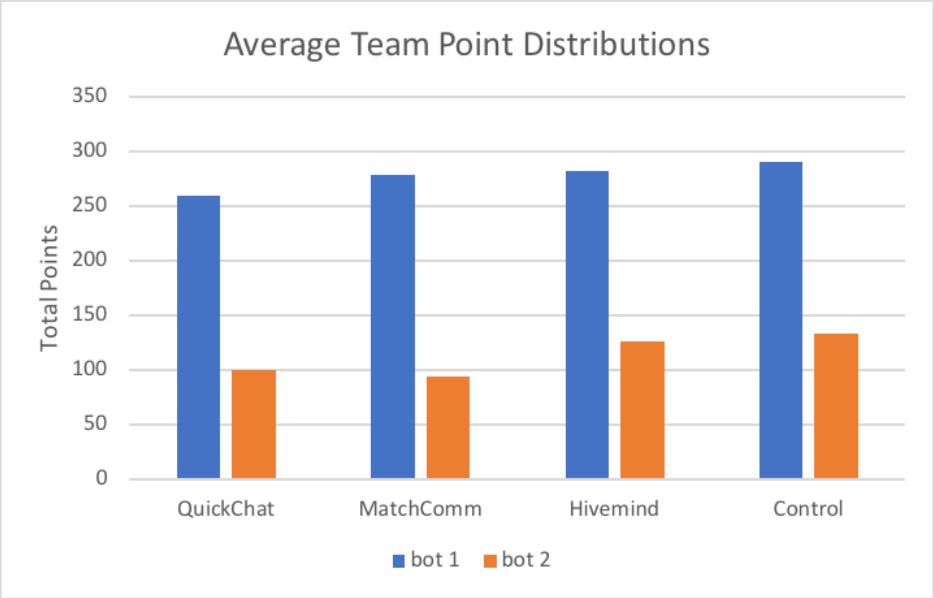


**Figure 9: Point Distributions Among Teams**

other bot to tend the goal, unless that other bot should ever come within 400 units of the ball. I had thought that the Hivemind might be more successful than the two chat bots, because it might create a more aggressive defender if the bot can go after the ball as soon as it is closer than its teammate, however it ended up having a sort of oscillating effect between the two players, where neither one would pursue the ball for too long, leading the opponent to succeed. In terms

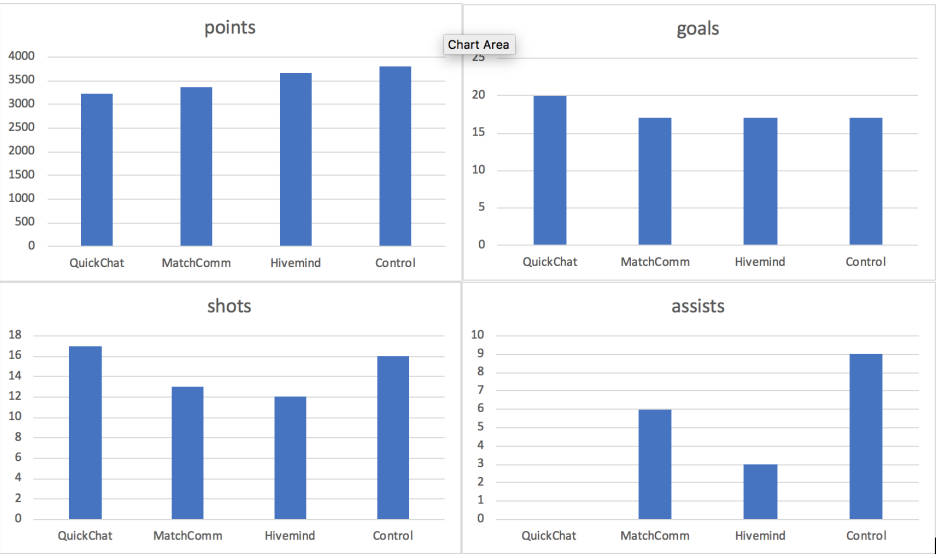


**Figure 10: Scoring Breakdowns**

of the ways that points are actually awarded, through goals, shots, assists, saves and touches,

you can see in Figure 10 that the HiveMind and the Control bots actually score the most points overall, though they tended to lose the most matches. This is likely through the earning of points by touching the ball, as in those cases, both of the bots on those teams are actively pursuing the ball for much of the game, so they were able to earn the majority of the points but losing the games in terms of goal scoring. In terms of goals, QuickChat unsurprisingly has more total goals than Hivemind and Control having won more games, however MatchComm has about the same amount as the Control and the Hivemind, despite winning the most matches in total. This possibly indicates that MatchComm had a more effective defense, and was not scored on as often, thus winning more matches, as opposed to QuickChat, which appears to have won through a more aggressive offense. However, this idea is not supported by the total saves per

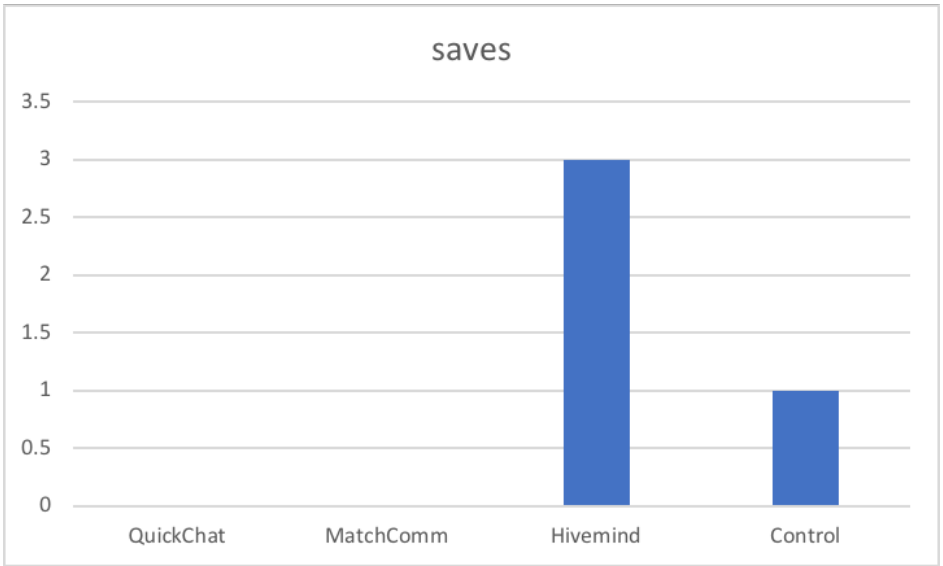


**Figure 11: Total Saves Per Bot**

each bot team across all games seen in Figure 11, which indicates that neither the QuickChat, nor the MatchComm bots made a single save across all games. The game is not completely explicit about what constitutes a save, so this could indicate that the defense method in these two bot teams was ineffective, or it could indicate that these bots were more capable of simply preventing shots on the goal from the opponent, rather than blocking them once they had been shot. Unsurprisingly, Figure 10 shows that the Control bot has the most assists, which is in line with the fact that this is the only bot that has both bots on offense at all times, so naturally one

bot on the same team would more frequently touch the ball right before the other scores in this case.

Its difficult to compare the results of this study to any other study, particularly those discussed in the related work section, as this is a fairly new area of research, especially with regard to Rocket League specifically, so there is not a lot of scholarly data available about it. However, it would be interesting to see how my bots stack up to some of the bots that come with the download of RLBot. Further research would need to be done into these bots however, to determine their design and strengths, in order to properly analyze these matches.

## 6. Conclusions and Future Work

It is encouraging to see that the data gathered through the matches suggests that the communication aspects that I added to my bots did actually improve their performance, as each of the communication bots outperformed the control in number of wins. However, the results also exposed a weakness in my bot design, in that is it largely offensively based, as I designed the bots individually first, and then added in the ability to separate the team into offense and defense. However, likely because I did not program any specifically defensive minded tactics into the bots, they did not actually perform very effectively once they were on defense, and very few saves were made, particularly in the chat based bots. In terms of further developing these bots, as I mentioned in section 5 it would be interesting to pit these bots against the default bots given in the RLBot Pack upon download, to stack them up against a wider pool of bots and bot makers. Additionally, it would be helpful to develop some form of defensive protocol, beyond simply waiting in the goal and then driving toward the ball when it approaches.

In terms of the goalie bot, this is definitely the area in which the most future work can and should be dedicated. This is especially true given the lack of success on the defensive end of most of my other bots, and the lack of concrete data or knowledge of how to improve upon it. It would be really exciting and a worthwhile pursuit to see machine learning incorporated into my other bots, particularly given that the defensive and offensive roles of these bots are completely

separate, so the bots could use machine learning to become particularly good goal tenders, and would not need to use machine learning to improve at the entire game as a whole, which many sources have shown to be far too complex at the moment, without a way to speed up the game play.

## 7. Acknowledgements

First and foremost, I'd like to thank my advisor, Ryan Adams, for continuing to support and advise me despite a late start and an extremely strange set of circumstances. He also introduced me to the world of RLBot in the first place, and without him this project would not have been possible. I'd also like to thank all of the developers on RLBot and discord, for freely sharing information and source code to new bot makers, as their work guided much of the work I was able to do.

## 8. Honor Code

I pledge my honor that this paper represents my own work in accordance with University regulations -Natalie O'Leary

## References

[1] "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.
[2] "Deep blue (chess computer)," *Wikipedia*, 2020.
[3] *Settlers of Catan*, Catan GmbH, 2020.
[4] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[5] *Rocket League*, Psyonix LLC., 2020.
[6] RLBot, "Rlbot," https://github.com/RLBot/RLBot, 2020.
[7] SaltieRL, "Saltie," https://github.com/SaltieRL/Saltie, 2019.
[8] Y. Verhoeven, "Team behavior of artificial intelligence bots in games," Master's thesis, Westfalische Wilhelms-Universitat, Munster, 2020.
[9] ViliamVadocz, "Bots," https://github.com/ViliamVadocz/Bots/tree/master/HiveExampleBot, 2020.