**Exploration of DeepMind’s StarCraft 2 API**

**Jeffrey Cain**

Kansas State University

jeffreycain4@ksu.edu

**Abstract**

Over the past few months, I have been looking into DeepMind’s StarCraft 2 API. Blizzard and DeepMind partnered together to create a Machine Learning and AI API for the video game StarCraft 2. Python is one of the languages that I utilized in their API called PySC2. Finally, there are also a large community that worked together to teach others about this API through Wikipedia.

**Objectives**

The purpose of this project is to learn more about Artificial Intelligence and understand more about how it can be utilized in the real world. Exploring how to create an AI in StarCraft 2 is how I’ve chosen to learn more about Artificial Intelligence. How does an AI learn in the environment it was placed in? How effective is the AI able to learn? What are some things that could improve the way that the AI learns? These are all questions I was expecting to learn about as I dove deeper into understanding DeepMind’s API.

**Background**

DeepMind and Blizzard partnered in August of 2017 to create a StarCraft 2 API. This was for the purpose of providing an environment for Machine Learning agents. This also provided an API for reinforcement learning using Python. Everything that was ran in this project is free and open source. I also spent a lot of time utilizing the third party support to help learn and understand different aspects of the API. To run this API, first you need to have downloaded StarCraft 2 from Blizzard’s Desktop Application. Afterwards, you need to go to DeepMind’s GitHub page and download their pysc2 code. They have installation instructions on their GitHub page. Also, I used pip to install pandas since it was utilized in the tutorial I was following. Once this is installed correctly, based on many of the third party support, I found understanding Version 1.2 was useful in quickly understanding the API. I had used Notepad++ to create the AI agents using Python code. Finally, I ran these agents using python in the command prompt.

**Methodology**

The project first started out by understanding python code and how the API functions. While I had spent some time learning Python in class previously, this project required me to spend more time understanding code in Python. After understanding how the API and Python works, I spent time learning about different ways to teach an AI using reinforcement learning. This also included testing the API by running different forms of AI to test how the program runs. The final portion of the project was running the AI with all of the functionality and recording the results.

**Python and PySC2**

This was the first stage of the project, understand more about Python and learn how the PySC2 API can be used in a python program. PySC2 has many unique functions and classes that it had been created. These functions include, tracking the camera, placement of buildings, and many other syntax that the API introduces into the Python language.

Another aspect that I had to learn when starting this project were different ways to initialize the agent when starting the agent in the command prompt. This includes understanding different scrips to run. You are able to set up the specific map, agent, race, and max agent’s steps. I also included a no render function to the script to save memory space while running the program. This is the python call I would run:

python -m pysc2.bin.agent --map Simple64 --agent refined\_agent.SparseAgent --agent\_race T --max\_agent\_steps 0 –norender.

**Q-Learning Table**

One of the things that I learned about on this project was about the Q-Learning Table. This table has 5 basic actions that it will take. These actions are Initialize, determine action, perform action, calculate rewards and finally update the Q-Table. After initializing the table, the Q-Table runs through the other actions in order. This allows the Q-Table to constantly learn what actions are affective and what actions aren’t. When chosen what action to perform, it looks at its “brain” to determine which action is most effective. Each individual step is given a number to show its potential reward where each number starts at 0. As the Q-Table learns what actions are affective and what actions aren’t, it changes its potential reward number to correspond with what it learned. As the agent runs hundreds of games, it is able to learn how to win a game, even 60% of the time.

One aspect of this table is that it isn’t able to hold the information from one session to the next. In order for the table to not have to re-learn after stopping and restarting, a way to save this information onto a file and load it in when the table is initialized was created. When the table updates itself, it sends the new information into a file called file\_name.gz and is compressed into gzip. Also, if this file exists, then it is loaded into the Q-Table during the initialization action. This allows the AI to learn through multiple sessions.

**Reward System**

In order for the Q-Table to actually learn while running, there needs to be a reward system to tell the “brain” what is effective and what isn’t. The reward system is quite basic. First, there is a reward given for killing enemy units, 0.2, and enemy barracks, 0.5. In order to win, the AI has to defeat the opponent and giving a reward for killing units and buildings allows the ability to teach the AI that killing the enemy is important.

Another aspect to the game is if the AI wins, draws, or loses the game. This provides a single value at the end of the game, 1, 0, or -1 which is used to affect the Q-Table. A win happens when the AI destroys all of the opponent’s units and buildings and a loss is when the opponent destroys all of the AI’s units and buildings. A draw exists when the specific game reaches the step limit for that unique game. The time limit for a draw is about 25 minutes.

**How the AI Runs**

The AI isn’t the most effective AI and utilizes many aspects to speed up learning. One of the first things I did was only use one race for the AI to learn. This reduces the amount of possible units and buildings that need to be built. Another restriction placed on the AI is that it is only able to train 1 type of attacking unit, the Marine. Since I only have 1 machine to run these games, and was limited to how many hours of a day for 1 machine to run these tests, these restrictions are important to improve in a short time frame.

A second interesting aspect about the AI comes from the reward system. Since there is a time limit for each individual game, a draw is a possible outcome. This time limit was placed on the games to increase how quickly the AI is able to learn during the first 25 minutes of the game. Also, one way to speed up learning was to ignore learning from a game that ends in a draw. While this might seem like counter-intuitive, however; this allows the Q-Table to only learn when it wins and when it fails. Ultimately, focusing only on wins and losses allows the AI to learn using a restricted data set.

**Results**

After having run the program a few hundred times, I was able to collect a bunch of interesting data. At the start of the AI’s lifespan, the AI would lose or draw 70% of the games. This happens because it took a lot of time for the AI to realize where it needs to go to win the game. Most of the early games had the AI build a couple dozen Marines and run them around on the map. However, eventually the AI would lose because it doesn’t know how to build stronger units and didn’t understand how to win since it hasn’t learned how to receive the best rewards yet.

By the 200th game, the AI starts understanding how to win the game. By this point, the AI knew where the opponent would have their units and buildings and where to go to receive the most rewards. This stage was between the 200th and 500th games. There is a gradual increase in wins during this period, however; there is still an interesting problem that the AI still hadn’t learned at this point. The AI could destroy most of the opponents units and base but if there are still some units or base left alive then the game wouldn’t end automatically. This caused the AI to draw or even lose games it should’ve won.

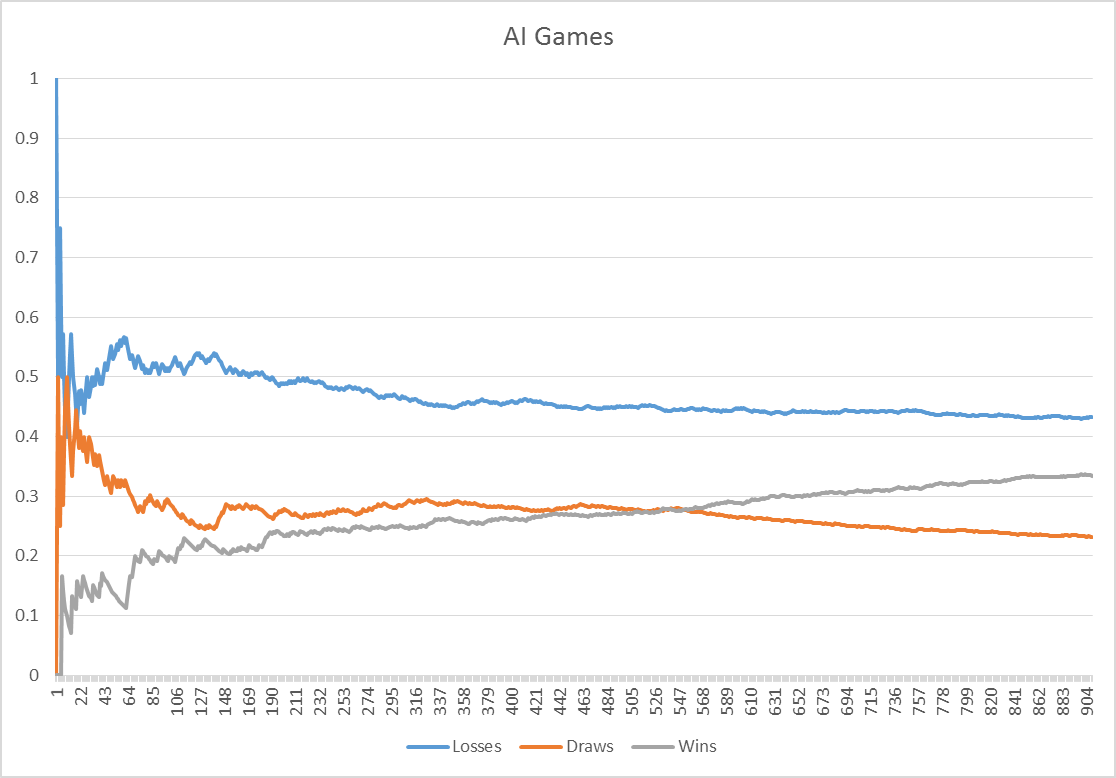


Figure 1: Graph of Wins/Losses/Draws

By 551, wins and draws have become equalized at 27.77% of all of the games played. The AI had still lost 44.46% of the games, but this became a huge stepping stone. As show in the graph, after game number 550, the AI starts to massively increase its likely hood to win over the next 350 games. If the AI stays on its current course, I would even predict that in the next 300 games, the amount of won games will overshadow the amount of lost games. Figure 2 shows the percentage of games won, lost and drew.

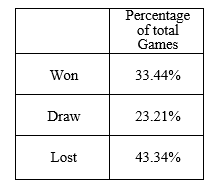


Figure 2: Total Games

Since there is such a significant difference between the first 500 games and the last 400 games, I wanted to also look specifically looks at the most recent data. This data can be found in Figure 3 and Figure 4.

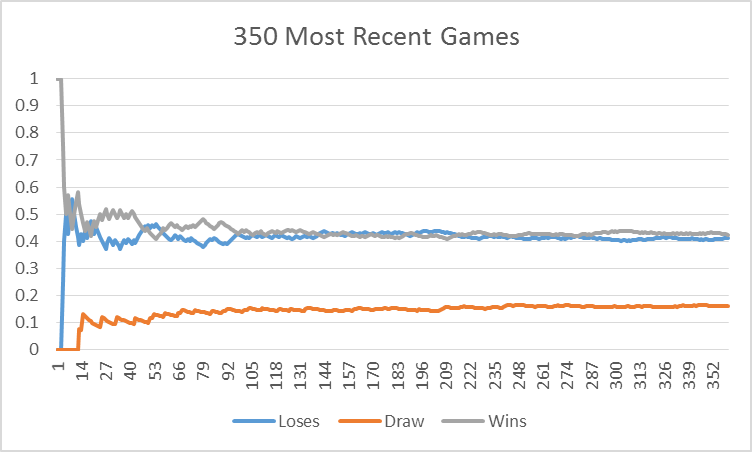


Figure 3: Most Recent Games

From this graph, it I am unable to predict anything about how the AI will act in the future. It seems like it has reached an equilibrium point. This knowledge does prove my point that the AI has learned enough to win the same amount of time that it losses. Figure 4 shows the true percentages for this graph

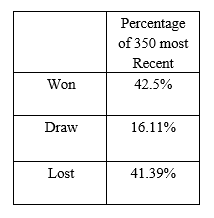


Figure 4: 350 Most Recent Games

While the AI hasn’t truly overcome the total losses, it is slowly improving over each game. Even though the AI only has won 1.11% more games over losing in the past 350 games, in the next 350 games I would predict that it would have a significant difference causing there to be more games won than lost.

**Future Work**

There are a lot of things that can be improved upon for the future. One of the most obvious thing that can be improved is adding support for more units and buildings to be built. This is one of the strongest restrictions on the AI right now. Another piece of functionality that would be important to add would be knowledge of multiple races. Right now it only runs with one StarCraft race. Allowing the AI to learn about all of the different races would create a more flexible AI to play the game.

Something else that would be important to add would be understanding text input from the opponent to understand when it sends the message “gg”. This message is a common for the phrase “good game” which means the person is conceding the game. Implementing this change would increase the amount of games the AI would win. This change would fix the problem of the AI not destroying all of the opponents units and buildings.

Finally, the next steps of this AI would be training it on different StarCraft 2 maps. Since it only trained on one map, learning different maps would increase the strength of the AI in a game.

All of these are ideas that I have to possibly improve the functionality of the AI. It is also possible to

mprove the “Brain” behind the AI. By changing how the AI fundamentally learns, or even by tweaking a

few of the numbers are all ways that the AI could learn faster than it does currently.

**Summary**

As mentioned previously, the purpose of this project was to explore how to use DeepMind’s StarCraft 2 API and how to utilizing it with a real AI agent. Through exploration of Python and PySC2 I had learned many different aspects of the programming language. I also learned how to implement a Reinforcement Learning agents. Using a Q-Table, I learned how to make an AI that can learn step by step what an effective action is and what isn’t an effective action. Thanks to the Q-Table, it is able to make incremental steps to win more frequently that before.

The AI is able to constantly run game after game while it learns. Many restrictions are placed upon the AI to decrease the randomness and speed up the learning process. Some of the things that were restricted included the type of units utilized and the map the games are played on. Even with these restrictions, the AI was able to learn. By the 500th game there were the same amount of wins as there were draws. I also predict that by the 1300th game, the wins will overcome the losses.

In conclusion, I was able to learn a lot about Artificial Intelligence and StarCraft. This project gave me a lot of knowledge about how AI’s function in the real world and how complex even simple ones are. Seeing how the AI has learned over 900 games, its unique quirks, and how it learns to overcome obstacles has been an educational experience.

**References**

Brown, Steven, “pysc2-tutorial”, (2018), GitHub

repository, <https://github.com/skjb/pysc2-tutorial>

“DeepMind and Blizzard open StarCraft II as an AI

research environment” *DeepMind*, deepmind.com,

Aug 9, 2017,

<https://deepmind.com/blog/deepmindand-blizzard-open-starcraft-ii-ai-research-environment/>

DeepMind, “pysc2”, (2018), GitHub Repository,

https://github.com/deepmind/pysc2

“SC2 AI Main Page”. *Wiki StarCraft 2 AI*

wiki.sc2ai.net, Nov 26, 2018,

http://wiki.sc2ai.net/Main\_Page

Zhou, Morvan, “Reinforcement-learning-with-

tensorflow”, (2018), GitHub Repository,

<https://github.com/MorvanZhou/Reinforcement->

learning-with-tensorflowll