Starcraft II: Reinforcement Learning with DeepQ Network and PySC2 Framework

Yiğitcan Çoban
Faculty of Computer and Informatics Engineering
Istanbul Technical University
Istanbul, Turkey
cobany16@itu.edu.tr

Egehan Orta
Faculty of Computer and Informatics Engineering
Istanbul Technical University
Istanbul, Turkey
ortae16@itu.edu.tr

Rumeysa Nur Arslan
Faculty of Computer and Informatics Engineering
Istanbul Technical University
Istanbul, Turkey
arslanru16@itu.edu.tr

İlgin Balkan
Faculty of Computer and Informatics Engineering
Istanbul Technical University
Istanbul, Turkey
balkani17@itu.edu.tr

I. Introduction

DeepMind is an institution founded by Google that develops algorithms that learn to deal with complex problems and conducts studies in the field of artificial intelligence[1].

As part of a collaboration between DeepMind and Blizzard, an environment for Reinforcement Learning research has been created on the strategy game StarCraft II.

PySC2 is a toolset for DeepMind's StarCraft II learning environment built for Python. PySC2 provides an interface for Reinforcement Learning agents to interact with StarCraft II, receive observations, and send actions. Analyses were performed on four StarCraft 2 mini games in our project.

Mini games maps used in the project:

- BuildMarines
- CollectMineralsAndGas
- CollectMineralShards
- DefeatRoaches

In our project, we developed an RL algorithm using DQN (Deep Q Network). We trained the developed algorithms with various iteration numbers. In this article, we will talk about the algorithms we developed and these train results.

II. Background

A. *PySC2*

PySC2 is the Python library that provides a component for agent development and has a huge action space. The PySC2 library, which emerged as a result of the collaboration between DeepMind and Blizzard, provides opportunities for many RL studies developed on StarCraft II.

Observations provide information about the current situation of the environment in the game. The features.py file in the library provides information about each feature on the minimap. Through observations, health status, mineral and gas number and unit type can be accessed on the map[2].

StarCraft's action space show a defiance with basic actions that can be taken. Actions in StarCraft have a hierarchical structure and these actions can be changed as needed.

B. *Deep Q Network*

DQN algorithm is one of the most commonly used reinforcement learning algorithms. The main purpose of DQN is to examine the next actions of an agent and determine its future movement depending on the reward it will earn according to the actions it will make.

With DQN, actions given to an agent are performed and the results obtained are written to the Reward Table. Mean Reward is expected to increase as the agent's actions select locations learned in each iteration.

In the first iterations, the table does not give correct values because the agent has not been learned. In each iteration, the agent predicts the next targets according to the determined algorithm and acts accordingly and reaches the award.

When the agent reaches the reward, it switches to a new iteration and repeats the same process. When it is tried on a certain number, the agent knows where to go in which situation.

C. *Reinforcement Learning*

Reinforcement Learning is a purpose-driven Machine Learning sub-branch. Reinforcement Learning agents have goals, they can observe their environment and take action according to the situation within the environment.

Basically, an agent takes an action relative to the environment, and this is called policy. After this action, the agent expects a reaction from the environment. Every reaction that occurs has a payoff and these gains are evaluated in a predetermined reward system. Agents are trained on these gains and understand the correctness of their actions. Agents within the algorithm should try out each situation by performing various actions and choose the ones that look best based on these results[2].

III. Project

PySC2 library contains several mini games for users to experiment on Machine Learning algorithms. During our research we came across many solutions using these mini games. We have focused on four mini games within PySC2 library. To conduct this study we have decided to use a template to implement mini games faster. After our team decided to use DeepQ Learning algorithm, we used ray library to implement that solution. This library natively support tensorflow. RLLib is an open source library within Ray library and it provides an easy environment and a unified interface to various machine learning libraries.

All group members contributed to building process of base environment. After its construction group members proceeded to coding their own respective mini game.

A. *Collect Mineral Shards*

This mini map's agent is coded by Rumeysa. Four different actions were defined for DeepQ model.

- Actions were; go to the nearest mineral, go to the farthest mineral, go to the other agent's nearest mineral, and go to the other agent's farthest mineral.
- Parameters of the model were; positions of each marines and positions of the minerals each step.

Fig. 1. Collect Mineral Shards

To create actions used in this model, actions.RAW_FUNCTIONS.Move_Move_pt method of PySC2 is used and different coordinate arguments were passed. To calculate coordinates of destination depending on action, algorithm calculates all possible action locations, calculates the farthest and nearest for each agent and passes the suitable coordinate.

The model is trained on Google Colaboratory GPU instance and it is trained with 300 iterations.

B. *Collect Mineral and Gas*

This mini map's agent is coded by İlgin. Five different actions were defined for DeepQ model.

- Actions were; sending idle SCV to vespene, sending idle SCV to minerals, creating SCV from command center, building refinery and building supply depot.
- Parameters of the model were; mineral source count, SCV count, refinery count, idle SCV count, supply depot count, total available mineral amount, total available gas amount, total gas source count which has no refinery built upon.

Fig. 2. Collect Mineral and Gas

To create actions used in this model,
actions.RAW_FUNCTIONS.Harvest_Gather_SCV_unit
actions.RAW_FUNCTIONS.Train_SCV_quick
actions.RAW_FUNCTIONS.Build_SupplyDepot_pt
actions.RAW_FUNCTIONS.Build_Refinery_pt
methods of PySC2 are used.

To get idle workers and send to minerals, first all terran scv unit order lengths are checked, if it is 0, the worker is idle. actions.RAW_FUNCTIONS.Harvest_Gather_SCV_unit takes SCV unit then sends it to specified mineral using tags. Building functions are similar to each other and actions which include building functionality selects a random SCV, then proceeds to build the building. Training SCVs is a fairly simple task which needs command center's tag as an argument then creates an idle worker.

The model is trained on Google Colaboratory GPU instance and it is trained with 300 iterations.

C. *Build Marines*

This mini map's agent is coded by Yiğitcan. There are four different actions were defined DeepQ model.

- Actions were; sending the idle worker to mineral, building barracks and supply depots, and creating marine. While creating barracks and supply stores, it discovers the command center by trying random values in a scale that we calculate to take radius into account.
- The parameters of the model we created for Build Marines are barrack count, supply depot count, mineral source count, gathered mineral amount, SCV count, idle SCV count.

Most of the units are selected with get_units_by_type method by giving their ID. r value is the radius value of the

Fig. 3. Build Marines

command center;
While looking at random values between the diameter of the command center (-5 * r) and (-4 * r) for the barracks, random points between (-4 * r) and (4 * r) for the y-axis were determined For supply depot, the values between (-3 * r) and -r were checked randomly on the x-axis, while (-4 * r) and (4 * r) values for the y-axis were checked. Additionally, idle workers are directed to mineral mines with the idle_to_mine action. The model is trained on Google Colaboratory GPU instance and it is trained with 300 iterations.

D. Defeat Roaches

The representative of this mini map was coded by Egehan. Four different actions have been defined for the DeepQ model.

- Actions, taking into account the action value, All marines attack the first roach, all marines attack the second roach, all marines attack the third roach, all marines attack the fourth roach, all marines run back, If the action value is higher than four, the action indexed marine will run back.
- The parameters of the model consist of marine and roach coordinates and count.

Fig. 4. Defeat Roaches

PySc2 for move movements in the model The actions.RAW_FUNCTIONS.Move_Move_pt method is used with different coordinate arguments. On the other hand, for the attack, actions.RAW_FUNCTIONS.Attack_unit method is used again from PySC2. Terran units gets better results when they run away after a while and attack again. Therefore, move and attack behaviors on the basis of the algorithm are determined according to this footnote.

The model is trained on Google Colaboratory GPU instance and it is trained with 791 iterations.

IV. Test Result

Mean rewards for each iteration can be seen on figures. Model are generally getting better performance on each iteration but we could not achieve a big performance increase due to insufficient actions and shorter train time. If we compare our training time with others on the internet, ours are equal to the 10% of others. We can see the disadvantages of not being able to train in BuildMarines mini games.

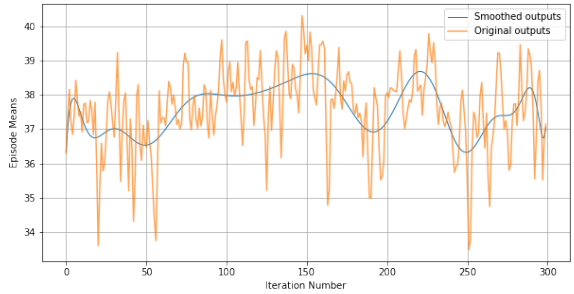

Fig. 5. Collect Mineral Shards reward mean result for 300 iterations

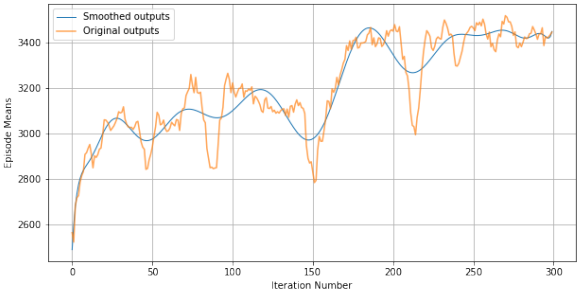

Fig. 6. Collect Mineral and Gas reward mean result for 300 iterations

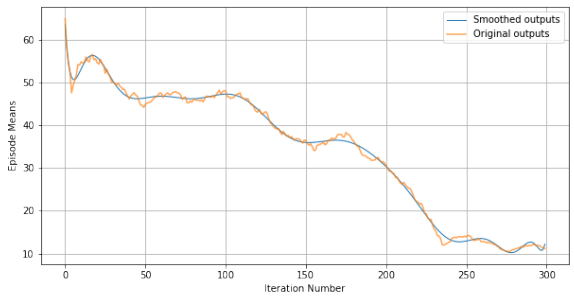

Fig. 7. Build Marine reward mean result for 300 iterations

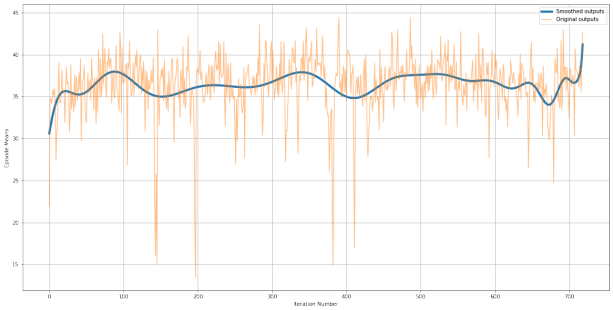

Fig. 8. Defeat Roaches reward mean result for 700 iterations

V. Conclusion

At the end of our project, we trained with different number of iteration values for each mini games over Google Colaboratory. The main reason for using Colaboratory is that the computers of our 4 project members do not have sufficient processing capacity. At the same time, we had great difficulties during the environment installation phase of the project. We had to delete many applications and programs due to the lack of space due to the enormous amount of space the Environment occupies on the hard disk.

We had to turn to the Starcraft II project, as DuckieTown, which we determined as the subject we would choose in the early stages of the assignment, could not be run on our computers as group members. However, this environment required quite a lot of action and payload like DuckieTown. In addition, the lack of documentation on the project and not knowing under which conditions we will realize this project put us in a deadlock at some points.

Our iteration numbers during the train phase were not sufficient to better observe the learning of the model. By making more iterations and making observations on our model, we could improve our values to this extent. Also, if we could choose larger action spaces across all mini games, and we could get better results with action spaces that would change and expand when our computers were of sufficient capacity.

You can reach our video and explanations through to this link: https://drive.google.com/file/d/1-mTGcFBnjB62xpdPfg1oipu2C6x0ukGD/view?usp=sharing

References

[1] O. Vinyals, S. Gaffney ve T. Ewalds, ”DeepMind and Blizzard open StarCraft II as an AI research environment,” DeepMind, 09 08 2017. Available: https://deepmind.com/blog/announcements/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment. [Access: 13 02 2021].

[2] B. Koncser, ”Introducing the PySC2 library (StarCraft II),” Medium, 26 02 2019. Available: https://medium.com/@SmartLabAI/introducing-the-pysc2-library-starcraft-ii-3b54396f421a. [Access: 13 02 2021].