StarCraft II Mini-Games Reinforcement Learning Model using Deep Q-Network

Mürüvvet Bozkurt
Faculty of Computer and Informatics
Istanbul Technical University
Istanbul, Turkey
Student ID: 150160133
Email: bozkurtm16@itu.edu.tr

Neslihan Çekiç
Faculty of Computer and Informatics
Istanbul Technical University
Istanbul, Turkey
Student ID: 150160060
Email: cekic16@itu.edu.tr

Dilasude Göktürk
Faculty of Computer and Informatics
Istanbul Technical University
Istanbul, Turkey
Student ID: 150150134
Email: gokturk15@itu.edu.tr

Yarkın Söyler
Faculty of Computer and Informatics
Istanbul Technical University
Student ID: 150160018
Istanbul, Turkey
Email: soyler16@itu.edu.tr

I. Introduction

Starcraft II is a real time strategy game and the aim of this game is to create an army to expand the base and destroy the opponent. It is a fast-paced game that requires the player to make hundreds of decisions and micro-actions every minute. With the limited observation area and feature richness in the game, it is a very good environment for developing a reinforcement learning model.

In this paper, using the StarCraft II learning environment, the sub-problems in the game were tried to be solved with the mini games in the PySC2 library. A model was created for each selected mini game by training agents with the Deep Q-Network function in the Ray library.

Team name: Group 11

Problem Distribution:

Neslihan Çekiç - CollectMineralsAndGas Mini-Game

Mürüvvet Bozkurt - DefeatRoaches Mini-Game

Dilasude Göktürk - BuildMarines Mini-Game

Yarkın Söyler - MoveToBeacon Mini-Game

Neslihan and Mürüvvet - Implementing base environment for mini-games, selecting algorithms, methods and libraries.

CLICK FOR PROJECT VIDEO

Running codes: run python ray_dre.py, python ray_cmg.py, python ray_bme.py or python ray_mtb.py to starting train.

II. Environment

Our environment is StarCraft II Learning Environment based on the StarCraft II Real Time Strategy (RTS) game. Building a base and army, managing an economy and destroying their enemies are the goal of Star Craft II. There are 3 different races that consist of Terran, Protoss and Zerg in Star Craft II. These races have distinct strategies, unit and advantages over each other.

Interface that observation and actions are defined with regards to low resolution grids of features is need in order to make Reinforcement Learning in StarCraft II[1]. Rewards are specified as taking score against the computer.

PySC2, one of the component of SCLE, provides to define observation specification and actions. Besides, a random agent and a some rule-based agents are comprises by it. Also, this component includes some mini games which have tools for visualisation and challenges in order to understand what the agent can do. The reward structure for mini-games can reward distinctive actions, in contrast to the full game the place the reward is just triumphing or losing.

A. Problems

1) CollectMineralsAndGas: There are 12 SCVs(workers), 1 Command Center(produces SCV), 16 Mineral Fields and 4 Vespene Geysers as the starting state in this mini game. Rewards are earned according to the number of resources collected by Agents, ie SCVs. Gathering resources requires increasing capacity, generating more SCVs with Command Center, and generating additional Command Centers to achieve a better result. The ending condition is the passage of 300 seconds. The total number of rewards is equal to the total amount of Minerals and Vespene Gas collected.

Fig. 1. CollectMineralsAndGas Mini-Game

2) DefeatRoaches: In the Defeat Roaches mini-game map, there are 9 Marines in vertical line at random in map and 4 Roaches groups on the opposite side.The reward is earned by the Marines destroying Roaches with an optimal combat strategy that focuses fire on the Roaches; There is no need for a separate action to fire. The Marines fires spontaneously when the roach is within a certain range. Each time marines

are defeated, a new group of 4 roaches will spawn. Marines are given 5 additional marines with the full health and surviving Marines maintain their current health. Also, new units are positioned on opposite sides of the map. If Roach defeated, the reward will increase by 10 and if Marine defeated the reward will decrease by 1. Besides, time limit of this mini-game is 120 seconds.

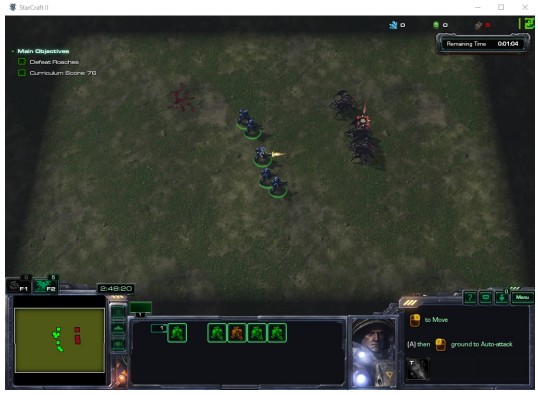

Fig. 2. DefeatRoaches Mini-Game

3) BuildMarines: This minigame goal is to train maximum marines with limited sources(mineral and workers). The game is starting with 1 Command Center with 12 SCV's and Mineral Fields. SCV's start to collect minerals and after that, they have to build supply depots for places for marines and they have to build barracks for training marines. All buildings and training operations need minerals and the game duration is 900 seconds. The curriculum score is equal to the trained marine number. The agent must use sources intelligently and give decisions in the right order for maximum reward.

Fig. 3. BuildMarines Mini-Game

4) MoveToBeacon: A map with 1 Marine and 1 Beacon. Awards are won by taking the marine to the beacon. When the Marine wins a reward for reaching the Beacon, the Beacon is teleported to a random location (at least 5 units away from the Marine).

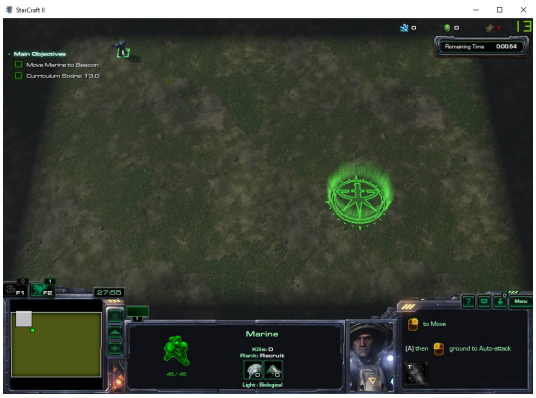

Fig. 4. MoveToBeacon Mini-Game

B. Difficulties

One of the biggest challenges was to train the network due to the CPU capacity of our computers was not enough, we could not train it first. Then, when we tried it with another computer, we trained it with a certain number of iterations because it took a long time to train.

Learning libraries and environments was one of the challenges because time to learn them was not enough.

III. Micro-Actions

In our implementation we created micro actions for each mini-game as below.

1) CollectMineralsAndGas:

- **Harvest Minerals:** Collects Minerals from mineral fields closest to agent.
- **Harvest Gas:** Collects Minerals from refineries closest to agent.
- **Build Refinery:** Builts refineries into gas geysers, in order to collect gas.
- **Build Command Center:** Builts command center, in order to train SCV and increase capacity, and thus collecting speed increases.
- **Build Supply-Depot:** Builts supply-depot, in order to train SCV, and thus collecting speed increases.
- **Train SCV:** Train new SCVs for harvest speed. Needed 50 minerals and 1 supply for training SCV.

2) DefeatRoaches:

- **Move Up:** Move marine 2 block up.
- **Move Down:** Move marine 2 block down.
- **Move Right:** Move marine 2 block right.
- **Move Left:** Move marine 2 block left.
- **Attack:** One marine attack one roach.
- **Attack All:** All marines attack one roach.

3) BuildMarines:

- **Harvest Minerals:** Collects Minerals from mineral fields closest to agent.
- **Build Command Center:** Builts command center, in order to train SCV, and thus speed increases.
- **Build Supply-Depot:** Builts supply-depot, in order to train SCV, and thus collecting speed increases.

- **Build Barracks:** Builts barracks, in order to train marine.
- **Train SCV:** Train new SCVs for harvest speed. Needed 50 minerals and 1 supply for training SCV.
- **Train Marine:** Train new marines. Needed 50 minerals and 1 supply for training marines.

4) MoveToBeacon:

- **Move Up:** Move marine 2 block up.
- **Move Down:** Move marine 2 block down.
- **Move Right:** Move marine 2 block right.
- **Move Left:** Move marine 2 block left.
- **Move to Beacon:** Move marine to beacon location.

IV. Used Library

A. Ray

In the Reinforcement Learning applications, a lot of processing may be required and may need to be scaled to a cluster. Libraries can be used for managing these operations and clustering easily. Ray is the one of these libraries and Ray has simple usage, fast and offers distributed computing.[2]

1) Tune: Tune is a Python library for experimenting and hyperparameter tuning at any scale. It can easily initiate a multi-node distributed hyperparameter scan. It can support any machine learning framework including PyTorch, XGBoost, MXNet and Keras. It can automatically manage checkpoints and logging to TensorBoard.[2]

2) RLlib: RLib is library for Reinforcement Learning and it is open source. Thanks to the RLib developer community, various algorithms and frameworks are getting integrated into a shared library. This way, it is much easier to evaluate the performance and the tradeoffs of many other alternative approaches. Despite the fact that RLib incorporates combinations for the mainstream TensorFlow and PyTorch structures, the greater part of its internals are skeptical for specific frameworks. Due to the unified API, a wide range of cases are addressed, from integrating RL support into a customer application at scale, to conducting research with a large volume of offline data.[2]

B. Gym

Gym is interface that is open source in order to do tasks of reinforcement learning. This library provides an environment. An action_space and an observation_space are come with every environment. These are one of the type "Space", and they describe the format of valid actions and observations. The Discrete space provides a fixed range of non-negative numbers and valid actions are either 0 or 1. An n-dimensional box is represented The Box space, valid observations will be an array of 4 numbers. Also, Box's bounds can be checked.

V. Methods

A. Reinforcement Learning

Reinforcement learning is one of the type of machine learning training model how to map state to actions for maximizing reward. The agent does not know that actions to take but it discover that actions achieve the most reward as trying them. Additionally, reinforcement learning is making a sequence of decisions. Effect of actions may be related not only to immediate reward, but also to the state in which rewards are awarded at all subsequent and subsequent stages. In other words, next state is depend on the output of the previous state.[3]

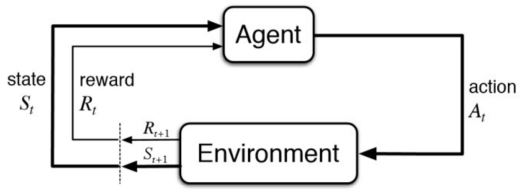

Fig. 5. Agent-environment interaction

A finite Markov Decision Process often is used in representation of Reinforcement learning.

$$M = (S, A, P, R, \gamma)$$

S is the state space and A is the action space , next state is shown by s' and current immediate state is shown by s. Probability of next state is represented by P= (s' | s,a). After taking actions, immediate reward is represented by R = R(s,a). The discount factor affecting the future award is expressed in γ.

1) Deep Q-Network: Training an agent that is involved in an interaction with its environment is about a reinforcement learning. The agent are affected at particular scenarios known as states by performing actions that cause to rewards which could be negative or positive.Agent has purpose which perform the sequence of actions that will eventually generate the maximum total reward also called the Q-value across an episode. In Q-Learning, the expected reward of each action at every step are know and as a result of this, the action to execute are known by the agent. In Q-learning, required memory for updating and saving is increases and required time to explore Q-table for each state is extreme, so that using Q-Learning in sophisticated and complex problems is inconsequential.

In Q-Learning, reward is calculated for each action at every step when we work in more complex and sophisticated environment estimate to q value for each action-state pair in a given environment, network will approximate the optimal Q function.The act of combining Q learning with a deep neural network is called deep Q-network. Deep Neural network that approximate the Q function is deep Q-Network. Deep Q network accepts states from a given an environment as input. For each given state input the network output estimated Q values for each action that can be taken from that state. The objective of this network is to approximate the optimal Q function. The optimal Q function will satisfy the Bellman Equation. The lost of network is calculated by comparing outputted Q values to the target Q values from Bellman equation.Network want to minimize their loss. After calculated loss, weights within network are updated via stochastic gradient descent and

back propagation. Bellman equation to compute and update Q values and Q table in order to find optimal Q function.[3]

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

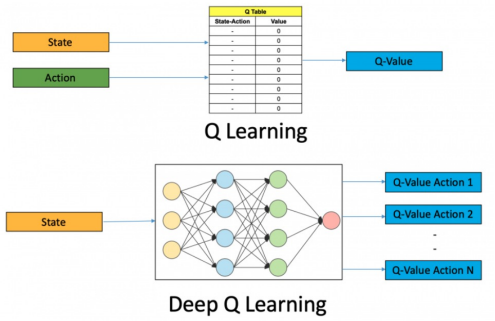

Fig. 6. Q Learning and Deep Q-Network Structure

As a result, we used the DQN algorithm in the RLlib library, thinking that it would be more useful to use the deep q-network algorithm in our implementation.

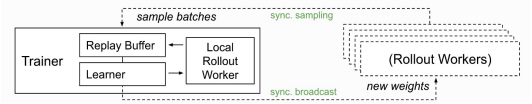

Fig. 7. Structure of Deep Q-Network in Ray Library

VI. Results

1) CollectMineralsAndGas: This mini game trained approximately 8-9 hours with about 600 iterations. In this process, the mean reward results of this model is as follows.

Fig. 8. CMG Means of Episodes' Reward

TABLE I
CMG Mean Results

Max Mean	Smoothed Last Mean	Best of DeepMind
3630	3475	3978

2) DefeatRoaches: This mini game trained approximately 8-9 hours with about 600 iterations. In this process, the mean reward results of this model is as follows.

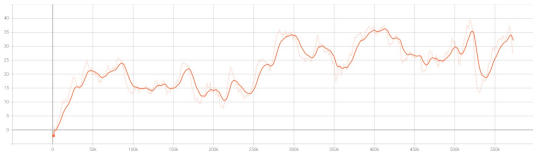

Fig. 9. DRE Means of Episodes' Reward

TABLE II
DRE Mean Results

Max Mean	Smoothed Last Mean	Best of DeepMind
39.52	28.28	101

3) BuildMarines: This mini game trained approximately 6-7 hours with 500 iterations. In this process, the mean reward results of this model is as follows.

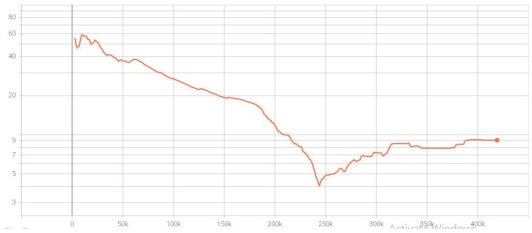

Fig. 10. BME Means of Episodes' Reward

TABLE III
BME Mean Results

Max Mean	Smoothed Last Mean	Best of DeepMind
60	9	6

4) MoveToBeacon: This mini game trained approximately 1-2 hours with 100 iterations. In this process, the mean reward results of this model is as follows.

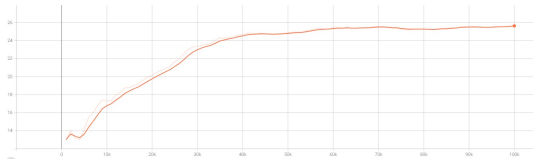

Fig. 11. MTB Means of Episodes' Reward

TABLE IV
MTB Mean Results

Max Mean	Smoothed Last Mean	Best of DeepMind
25.64	23.25	26.4

VII. Conclusion

We think that our models will get better results with longer train times and get faster results with better CPU capacity.

Some improvements that can be made:

For the MoveToBeacon mini-game RL model are not necessary because one action is the solution. The other actions

make no sense for that mini game and agent choose move to beacon action. This is why this model trained itself very quickly.

In the DefeatRoaches mini-game, some actions such as a single running back or all of them running back can be added, we think that this may result higher reward in fewer iterations.

In the CollectMineralsAndGas mini-game, Command Centers build in 5 step right of already built Command Center. This is not an effective solution for limited map size, but a better location can be strategically chosen. Additionally, supply-depot build in random location we can select better strategic location for that.

We did not have time to learn and implement these strategies as we do not know the game very well.

REFERENCES

[1] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A.S. Vezhnevets, M. Yeo and etc. *StarCraft II: A New Challenge for Reinforcement Learning*, 2017.

[2] *Ray v2.0.0 Documentation*, https://docs.ray.io/.

[3] M. Lapan. *Deep Reinforcement Learning Hands-On : Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*, Birmingham: Packt Publishing, 2018.