Repurposing StarCraft II Deep Reinforcement Learning Agent for Missile Defense Simulation

Ocdt Sweetnam, NJ

***Abstract*—Many video games today have complex environments which provide a difficult domain to solve for those studying multi-agent reinforcement learning (RL). This is due to their well defined environments which provide a clear goal to measure success, a large set of observations, and a challenging problem to solve. A notable game that has been studied is StarCraft II due to the multi-agent problems it possesses. Which can be seen as multiple players have to interact in a large action set that includes the observation, selection, and control of hundreds of units. This provides a great opportunity as these games can be provide alternative applications that can be found in the real world. In this work, we utilize an established advantage actor-critic method that was used for StarCraft II to determine if it can be used to provide insight into real world attack strategies. Therefore a missile defence scenario was provided based on its similar problems to test the selected method. An wrapper was needed in order to convert a set a feature layers from the missile defence scenario to match those provided by StarCraft II to reduce editing and adapting the network architecture used. Future work will consist of adapting the agents choice to be based on a action map rather than action set. In addition, changing the agent to act as a multi-agent defender instead of an attacker.**

I. Introduction

In this paper, I re-purpose an established multi-agent reinforcement learning algorithm which was successfully demonstrated within the StarCraft II environment to a missile defence scenario. We decided to utilize an agent from the StarCraft II environment based upon the comparable multi-agent problems faced in both scenarios. This can be seen as the missile defence scenario consists of an attacker and a group of defenders. Where the attacker can fire missiles upon a set of targets, and the defenders can intercept missiles to protect the targets.

A. Reinforcement Learning

Reinforcement Learning (RL) problems involve learning what to do by interacting with an environment through a defined set of actions that change the state of the environment. In order to have optimal behaviour the learner called the agent needs to learn what to do and how to map states to actions in order to maximize a numerical reward signal defined by the environment. Therefore the agent will interact with the environment at each discrete time steps, $t = 0, 1, 2, 3, ...$ where the agent receives the state of the environment S_t to select an action A_t. Then the next time step the agent will receive a reward R_{t+1} and move to the next state S_{t+1}. In addition we can define the decision-making process of the agent as the policy π_t where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. As stated previously the goal of the agent is to maximize the returned reward R_{t+1} and the agent can compute the expected rewards for state-action pairs for future actions and calculate the sum denoted $G_t = R_{t+1}+R_{t+2}+R_{t+3}+...+R_T$ where T is the final step. This is where a concept called the discount (γ) is necessary to determine how much we care about rewards in the future relative to an immediate reward. Therefore an agent wants to maximize the expected discounted return: [2]

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

This estimation of how good it is to perform a certain action in a given state is also known as the value function $v_\pi(s)$.

$$v_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \quad (2)$$

where E_π denotes the expected value of a random variable given that the agent follow policy π. Therefore through the value function and previous actions taken an actor can make decisions to optimize the return. In this paper we utilize a certain class of reinforcement learning methods known as actor-critic methods. This method works with an actor that will output the best decision based upon the current state of the environment by learning an optimal policy. While the critic will evaluate and criticize the actions chosen by the actor and advise it how to adjust by computing the value function. The goal of the actor-critic method is to find the best policy possible to maximize the reward.

B. Advantage Actor Critic

The selected agent that has been proven successful in similar complex environments to the missile defence simulation such as StarCraft II [3] utilizes the Advantage Actor Critic (A2C) method. By utilizing a wrapper to adapt the observations provided by the simulation, we can use this agent without making drastic changes that could potentially risk the agents performances. The A2C agent works by combining two types of reinforcement learning algorithms which are policy based and value based. Policy based agents directly learn a policy or a probability distribution of actions by mapping input states to output actions. While value based agents learn to select actions based on the predicted value of the input state or action. Therefore the A2C agent utilizes an actor and a critic in order to solve a problem. Where the actor chooses the action and the critic evaluates the quality of the given action. This is accomplished through the following temporal difference (TD) error evaluation:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (3)$$

where V is the current value function evaluated by the critic. By utilizing this TD error, or advantage estimate, the critic is able to determine the quality of the action, a_t, that causes the agent to transition from state s_t to state s_{t+1}. If the advantage is positive, and therefore indicates that the action result was better than expected, the action leading to that state is reinforced by increasing the probability of selecting that action from the given starting state. The converse is true should the TD error be negative. [2]

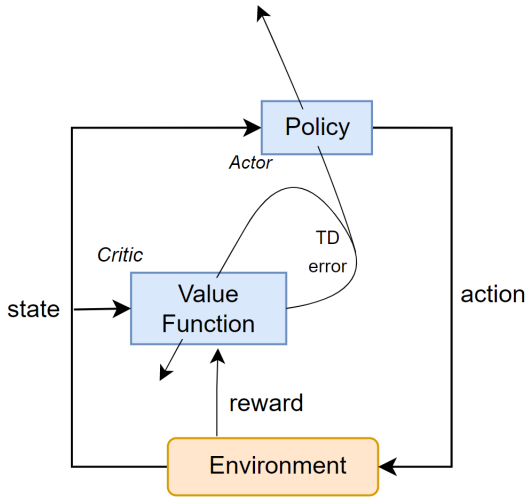

Fig. 1. Actor-Critic architecture [2]

C. Epsilon-Greedy Action Exploration

As described previously with reinforcement learning the agent learns how to solve a problem by mapping states to an action in order to maximize a numerical reward. With the case of our agent it is not explicitly told which actions to take, but instead must find which actions yield the most reward. To achieve this the A2C agent we chose utilizes the epsilon-greedy action algorithm in order to do so. Epsilon-Greedy is a simple method which randomly chooses between exploration and exploitation with a higher chance of exploitation. Through the use of exploration it allows an agent to learn about each action in the environment and hopefully allows the agent to learn a better policy which has not been found. While exploitation on the other hand, chooses the greedy action based on the value function calculated by the critic. The epsilon-greedy, where epsilon refers to the probability of choosing to explore over exploiting. The reason why we cannot utilize the greedy choice solely is because the agent may have not found found an action that gives a better reward. [1]

D. Starcraft II

StarCraft II is a Real time strategy game where one must construct a base, build an army, and attack or defend assets from an opponent in order to win. This game has a lot of different strategies that need to be learned in order to play effectively. This includes multitasking where a player must juggle big-picture economy, fast paced micro-actions and manage units to strategical and tactical destroy their opponent. The community of StarCraft II has pioneered the e-sports scene with millions of casual and high-level competitive players. Therefore, giving a long term goal in reinforcement learning to develop an agent to beat players of all varying skill levels.

The reason why we choose to use StarCraft II is because we wanted to utilize an agent that was proven to work within a similar environment to our Missile Defence simulation. This is because they are both making quick real time decision in an ever-changing environment. As well as both having multi-agents where they need to defend or attack a set of assets. In addition to them both having similar observations within each game. Although the Starcraft 2 environment contains a lot more than the missile defence scenario we can modify those to easily match one another.

II. Related Work

A. SC2LE and PySC2

The StarCraft 2 Learning Environment (SC2LE) [3] was created by DeepMind with the help of Blizzard entertainment to aid in the research of Reinforcement Learning problems within StarCraft. The SC2LE achieves this by providing both an API that allows programmatic control of the StarCraft environment to obtain game observations and send actions. In addition to a python component called PySC2 that allows interaction between a python reinforcement learning agent and the environment. All components of the SC2LE environment and how it interacts with a agent can be seen in figure 2 However, when agents are trained on the main game, they are unable to make significant progress. Due to the vast number of strategies that are need to be learned to play the game effectively. Therefore, PySC2 includes mini-games to play instead where an agent can learn certain skills before moving to the full game. Lastly, the environment also provides a means to speed up in game time to allow for agents to train faster.

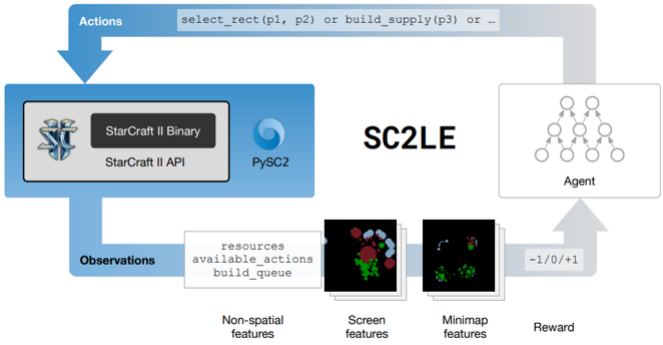

Fig. 2. StarCraft II Learning Environment [3]

B. StarCraft II A2C Agent

The agent we are re-purposing was developed by Xiaowei Hu, and Yilei Yang. The agent utilizes a fully convolutional

neural network, which predicts spatial actions directly through a sequence of convolutional layers. The network utilized has no stride and uses padding at every layer, thereby preserving the resolution of the spatial information in the input. The full architecture of the network which shows all the hidden layers including the input and output can be seen in figure 3.

Thanks to the PySC2 environment we can obtain the observations from StarCraft II which will be used as the input for the network which consists of the screen, minimap, and a list of actions. These observations provided are not rendered in RBG images but rather a set of feature layers. The screen, consists of a detailed view of a subsection of the world corresponding to the player's on-screen view, and in which most actions are executed is 64x64 pixels with 23 features (64x64x23). The minimap, which is a coarse representation of the state of the entire world with the size of 64x64 pixel image with 17 features (64x64x17). The list of actions, defined by the PySC2 environment where there is 524 possible actions that can occur depending on the state of the game. Next, the output which is used by the PySC2 environment to send an action to the StarCraft 2 environment for the next state of the game. The output consists of the spatial action, non-spatial action, and the Q-value. Spatial Action, is a 64x64 action map that is flattened where the x,y coordinates will be determined by utilizing argmax to find the location with the highest value. The coordinates founded is where the non-spatial action will occur which has the size of 524 for all actions. In addition to the quality value or q value is provided which is calculated by the Critic.

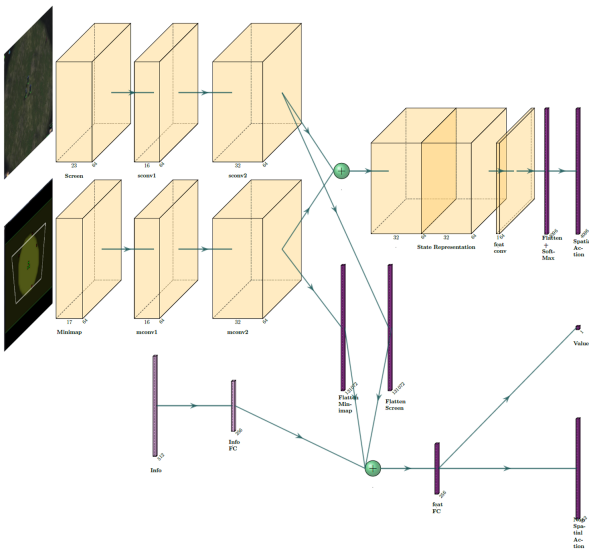

Fig. 3. A2C Agent Architecture

III. Proposed Work

A. Missile Defence Simulation

The purpose of the Missile defence simulation is to determine if machine learning can be used to find an optimal policy or strategy for attacking a set of targets as well as the defence of a set of assets. The simulation is composed of two separate teams one acting as the defender and the other as the attacker. Where the attackers goal is to maximize the amount of damage done to the targets by avoid being intercepted by the defender who wants to minimize the amount of damage. With respect to the work done for this paper the agent defined previously is acting as the attacker trying to find an optimal policy to maximize damage.

The simulation begins by initializing an environment that is similar to Canada's east coast. This is accomplished by generating a map that is 2000x2000 units which is divided into 4 separate sections vertically. Where the attacker is randomly positioned in the water on the right most side of the map and is represented as an Cruise Missile Launcher on a boat. The defenders will act to intercept missiles launched by the attacker who are positioned within the two middle sections of the map with a higher probability of them being positioned in the middle right section. This is because the targets will be randomly placed on the three left most sections with a peak of targets placed behind the defenders in the middle of the map. By increasing the chance of defenders and targets being positioned in certain places of the map will provide a general positional strategy by putting defenders in front of the targets. In the future one can further investigate an optimal strategy for the positing of defenders and attackers to provide a more realistic environment.

In the simulation there are 10 defenders who are given 5-10 interception missiles with a detection radius of 400km and an interception radius of 160km. Each interception missile has the speed of 12 units per time step along with the reload time of one time step. There is a calculation done for the defenders to determine a set of opportunities. These opportunities represent when a defender can intercept an attackers missile. It also must be noted that each interception has a success rate of 85%. For this paper we have a greedy defender who will select an opportunity for each defender if its available where the first opportunities found are favored over the rest.

There are 25 targets which the defenders must defend. Each target has a value set representing its worth, this value is randomly set between 50-150 units of score which is utilized to determine the reward or penalty signal for the attacker or defenders.

For the attacker they have 15-20 missiles that can travel at a speed of 14 units per time step where each missile same as the defender as a success rate of 85% on impact of hitting a target. When a missile hits a target, the targets value is then subtracted by its current value multiplied by a constant representing damage done. Therefore if missiles constantly hit one target eventually there will be zero reward given as there is no value to destroying a target that's already destroyed. An example of the environment can be seen in figure 4.

Since the current version of the agent only utilizes a list of available actions to determine the attackers next move and not the action map that is given. The agent can only learn which actions seek out good results and cannot determine this based upon an image that is provided. Hence for the training of the

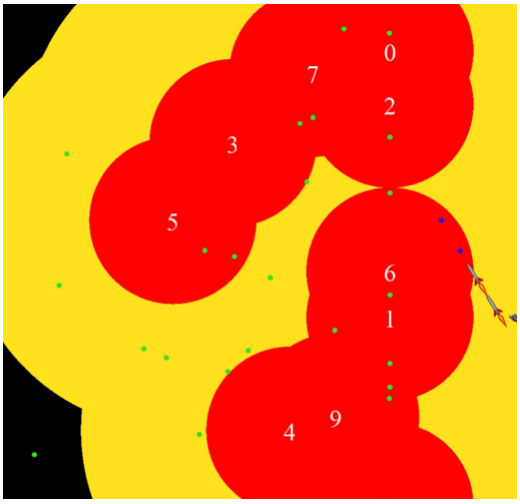

Fig. 4. Simulation Environment

agent once an environment is initialized we did not reinitialize a new environment as if any defenders or targets move from their position the current policy of the agent will not be useful.

B. Wrapper

In order to make the A2C agent work with the Missile Defence Simulation a wrapper was necessary to modify observations gathered from the simulation to match the expected input of the A2C agent utilized in the StarCraft II environment. In addition to converting the output of the agent so that the simulation can understand what action to do. The wrapper accomplishes this by converting the observations from the simulation to a set of feature layers which are then scaled down to a 64x64 pixel image. Next the image will then be converted to match the input of the agent which is two 64x64 pixel images that have 23 and 17 feature layers and one action list that has 524 different actions that can be taken. Since the StarCraft II environment consists of more features we fill in the unused features layers with zeros. This way the agent does not influence its decision based on these layers. By encoding and formatting the input in this manner makes it simple to make both components interact easily with one another by reducing the number of changes needed to be made. In addition to allowing easy extraction of information from the defence simulator and formatting to be identical to the StarCraft II environment. All extracted information from the defence simulator can be seen in figure 5 including how the information is encoded for the agent to understand each feature.

Once the input is sent to the agent, an output is given of three variables which includes a 64x64 action mask image, an action list of size 26 to match the number of targets to fire at plus one for no action, and a Q-value that is determined by the critic which determines the quality of the input state. From this output we only utilize the action list to determine which target we wish to fire at. This is achieved by completing the argmax algorithm on the list to determine the argument that

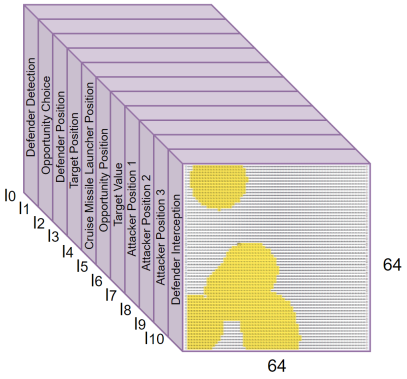

Fig. 5. Input Encoding

gives the highest value. Once the target index is obtained from the output it is sent to the simulator for the next step to attack the desired target. This loop is then repeated until all steps are completed in the simulation. The wrapper's architecture can be seen in figure 6.

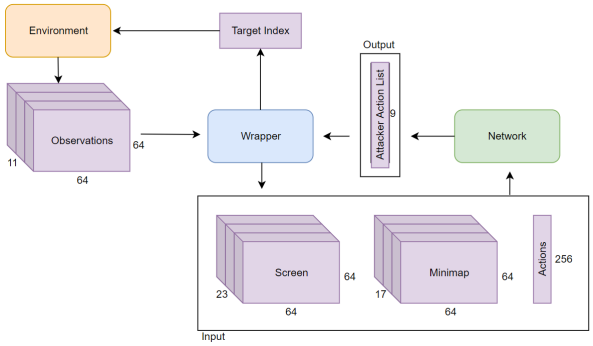

Fig. 6. The Wrapper Architecture

IV. Results

A. StarCraft II Mini Games

In the StarCraft II Learning Environment it provides many mini games to aid in the training of an agent to learn how to complete simple tasks before moving onto the full game. Therefore before moving the agent onto the missile defence simulation we obtained results on two mini games. The first mini game is called "Move to Beacon" as the name suggests the goal of this game is to move one troop to a beacon on the map. A point is rewarded to the agent each time he success fully moves the agent to the beacon then the beacon randomly is placed in another location on the screen, the agent has two minutes to obtain as much points as possible. In the initial testing of the agent the agent was utilizing the root mean square propagation which is a gradient descent-based learning algorithm. In addition we ran it with a learning rate of 0.0005 and a discount rate of 0.99. The learning rate is a tuning parameter in the optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. The results of these initial tests can be seen

in figure 7 where we ran 4000 iterations of the game. We obtained a mean score of 9 and a high score of 17.

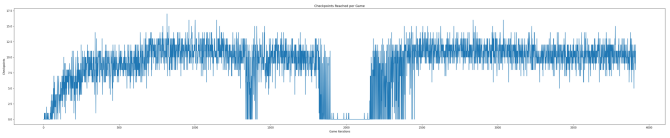

Fig. 7. Move to Beacon Initial Testing

From these results which shows a graph representing the score obtained per game iteration that the agent indeed learned a policy and optimized it over time even with the small deviation it had in the middle. Further testing was completed on the same mini game with changes in some parameters of the agent. This includes changing to the Adam optimization algorithm with a learning rate of 0.001, a discount rate of 0.9, and running the mini game for 40,000 iterations. The results can be seen in figure 8 where a mean score of 11 and a high score of 17 was obtained.

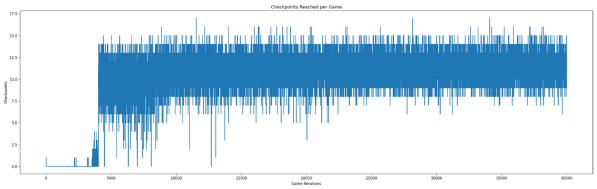

Fig. 8. Move to Beacon Further Testing

In these results we can see that it is more favourable as the agent learns an optimal policy quickly and enhances that policy over time.

Next we began testing on another mini game called "Defeat Roaches". In this mini game the agent has to use 11 troops in order to kill 4 enemies called "Roaches". The agent is rewarded by killing the enemies and is penalized when his troops are injured or killed. For this we utilized the same parameters ran in the previous test completed. The results can be seen in figure 9 where a mean score of 10 and a high score of 17 was obtained.

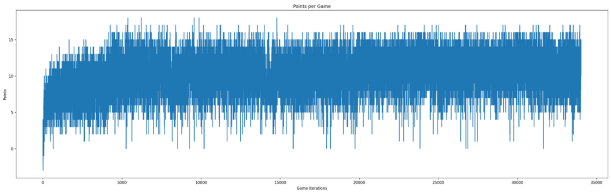

Fig. 9. Defeat Roaches Results

B. Missile Defence Simulation

Once we demonstrated that the A2C agent provided was capable of learning in the StarCraft II environment we then began testing the agent in our Missile Defence Simulation. Since the agent is acting as the attacker its goal is to do as much damage as possible to the targets. Score is determined by the loss of a targets value which is calculated by the targets value being subtracted by its current value multiplied by a constant representing damage done. For this we utilized the same parameters that were proven in the StarCraft II environment for 22,000 iterations. The results can be seen in figure 10 where a mean score of 149 and a high score of 264 was obtained.

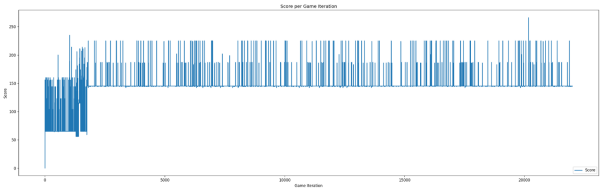

Fig. 10. Missile Defence Simulation Results

Therefore from the results obtained we can determine that we can utilize an agent from the StarCraft II environment to learn strategies for an attacker in the missile defence simulation.

V. Future Work

While the agents performance on the missile defence simulation was enough to prove that it could indeed learn a policy. Through further testing the agent can perform better to obtain a more efficient policy. In addition to exploring another Exploration algorithm such as Thompson Sampling (TS) or Upper Confidence Bound (UCB) may provide an increase in performance to find an optimal policy faster. This is because with epsilon-greedy action exploration there is a small chance that a random action is taken the agent will fire a missile upon a different target where it will be shot down immediately. For the agent to hit another target it needs to fire multiple missiles.

Additionally, the simulation in its current version randomly generates the location of defenders and attackers on the map, therefore to explore and find more realistic strategies an agent can be developed to learn strategic locations to attack or defend from. In addition we are currently only using the list of actions output of the agent to determine who we want to attack. Since the agent also outputs an action map we can modify the wrapper component to utilize this unused feature in order to determine who to attack using the images provided. As well as adapting the current version of the agent to work for the all defenders and learn an optimized policy for defending from an attacker.

VI. Conclusion

In this paper we re-purposed an agent that was successful in the StarCraft II environment for our missile defence simulation. We provided information on both the StarCraft II environment and the missile defence simulation with an explanation on why we choose to utilize an agent proven in StarCraft II. Including details regarding the Reinforcement Learning method, Advantage Actor Critic (A2C) used by the

agent. In order to utilize this agent a wrapper component was necessary to be developed in order to provide a way for the agent to interact with the missile defence simulation. From this results were obtained to prove that the agent did indeed learn a policy for both StarCraft II and the missile defence simulation acting as the attacker. Therefore proving that the simulation defined can be used to learn real world strategies for an attacker on Canada's east coast.

REFERENCES

[1] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Oriol Vinyals et al. "Starcraft ii: A new challenge for reinforcement learning". In: *arXiv preprint arXiv:1708.04782* (2017).