TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING
IOE CENTRAL CAMPUS, PULCHOWK

MINOR PROJECT REPORT

ON

Capture The Flag Game Using Multi-Agent Reinforcement Learning

Submitted by:

Amrita Bhattarai (075BEI006)

Brabeem Sapkota (075BEI011)

Love Panta (075BEI016)

Prashant Shrestha (075BEI024)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

May 17, 2022

Acknowledgement

Any project, small or big, has contributions from many people behind the curtain for its timely accomplishment with certain success. This project, indeed is also not the only effort of the four people that were credited for its completion. During different phases of this project, there were many helping hands who bombarded us with their careful suggestions, precious knowledge and encouragements. We certainly would like to take this opportunity to thank all the people who helped us for timely and successful completion of this project. Firstly, our greatest acknowledgement goes to Dr. Nanda Bikram Adhikari, HOD, Dr. Ram Krishna Maharjan, Dr. Diwakar Raj Pant and DHOD, Mr. Lok Nath Regmi whose supervision was most valuable throughout this project. Without their true guidance this would be no success at all. Their positive attitude and precious knowledge played a crucial role as a driving force toward success of this project. We owe a great many thanks to great many people who have helped us and supported us throughout the project either way; direct or indirect.

Amrita Bhattarai (075BEI006)
Brabeem Sapkota (075BEI011)
Love Panta (075BEI016)
Prashant Shrestha (075BEI024)

Abstract

Society can be thought of as an environment with multiple agents each acting independently to cooperate and compete with other agents. In this project we aim to develop agents that can perform well in such cooperative-competitive environment. The goal of the project was to train agents to perform well in cooperative-competitive environment of Capture The Flag setting with hybrid action space. We employed centralized training and decentralized execution, i.e. actor critic method with self play and curriculum learning for training the agents.

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Multi-Agent Reinforcement Learning, Capture the Flag, Self Play, Curriculum Learning, Actor-Critic

Contents

List of Tables

List of Figures

List of Symbols and Acronym

CTF Capture The Flag. 1, 2

MADDPG Multi-Agent Deep Deterministic Policy Gradient. iii, vi, 3, 7, 12, 13, 16, 21

MARL Multi Agent Reinforcement Learning. 2

RL Reinforcement Learning. 1–3, 20, 21

1 Introduction

1.1 Background

Reinforcement learning has recently been applied to solve challenging problems, from game playing [12] [16] to robotics [8]. In industrial applications, Reinforcement Learning (RL) is emerging as a practical component in large scale systems such as data center cooling [3]. Most of the successes of RL have been in single agent domains, where modelling or predicting the behaviour of other actors in the environment is largely unnecessary [9].

However, there are numerous applications of multi-agent domains where the agents learn to cooperate and compete for certain tasks. For example, multi-robot control [10], the discovery of communication and language [17] [5] [13], multi-player games [15], analysis of social dilemmas [7] all have multi-agent domains. Further progress in multi-agent domain is important to development of robot ecosystem where robot interact with each other and humans.Unfortunately, traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments [9]. Our objective is to implement cooperative competitive deep reinforcement learning in capture the flag game environment with hybrid action space.

1.2 Problem Statement

3D multiplayer games with hybrid action space(both continuous and discrete actions) represent the most popular genre of video game. Because of complex space of possible cooperative strategies and tactics, training agents to perform well in such environment is an extremely challenging task and of growing interest in AI research.

The setting considered is a Capture The Flag (CTF) game environment with two opposing teams and hybrid action space. The teams compete with the goal of capturing the opponent team's flag while defending their own. The team with the most flags captures within the play time limit wins. Additionally, players can tag the opponent team members to send them back to the spawn point and drop own's flag. CTF thus requires players(agents) of the same team to collaborate with team members as well as compete with the opponent team.

1.3 Objectives

- To implement multi-agent reinforcement learning in collaborative - competitive environment.
- To compare different learning approaches to multi-agent scenarios.

1.4 Application

- Improved gameplay experience

  Compared to rigid algorithmic opponents, trained agents are capable of expressing complex general strategies and can adapt to the human opponent's play style enriching the gameplay experience.

- Security

  Agents, as trained in this project can be deployed in security and surveillance purposes.

- The approaches used in this project and their results can serve as an insight to the current best approach for Multi Agent Reinforcement Learning (MARL) in other collaborative-competitive environments.

- As a test bed for RL.

2 Literature Review

CTF is played in a visually rich simulated physical environment and agents interact with the environment and with other agents through their actions and observations. Traditional reinforcement learning approaches like Q-Learning and policy gradient have so far not succeeded in training agents in multi-agent games that combine team and competitive play [9] due to the high complexity of the learning problem that arises from the concurrent adaptation of other learning agents in the environment. Here, we approach this challenge by studying team-based multiplayer video games, a genre which is particularly immersive for humans and has even been shown to improve a wide range of cognitive abilities. In contrast to previous work agents do not have access to models of the environment, other agents, or human policy priors, nor can they communicate with each other outside of the game environment. Each agent acts and learns independently, resulting in decentralised control within a team.

On the video games domain, Deep-Q learning(DQN) was developed as the first and tested Atari Games [11]. It was the first deep learning model to successfully learn control over policies over high dimensional input data. The network was not provided with any game-specific information or hand-designed visual features, and was not privy to the internal state of the emulator; it learned from nothing but the video input, the reward and terminal signals, and the set of possible actions—just as a human player would.

OpenAI developed an algorithm to train RL agent by self play and were able to beat professional in Dota 2 players [1]. The training was done by only using self-play, and the algorithm

learned how to exploit game mechanics to perform well within the environment. DOTA 2 is used actively in research where the next goal is to train the AI to play in a team-game based environment.

In the domain of strategic sequential game, AlphaGo [16] learned to play the game of Go using mix of RL and supervised learning i.e. with human knowledge. It uses value networks to evaluate board positions and policy networks to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. AlphaGo Zero [16] mastered the game of Go without human knowledge. Unlike supervised learning, here the algorithm is based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games.

A common approach to train agents in multi-agent environments is actor critic methods with centralized training(via critic) and decentralized execution (via actor). Notable works in competitive-collaborative multi agent environment include multi agent variation of DDPG [9] for continuous action spaces and counterfactual multi-agent policy gradient [4] with a focus on collaboration. We used a variation of MADDPG with self play and hybrid actions space for this project because of the simplicity of the algorithm and fewer networks required.

3 Theory

3.1 Reinforcement Learning

Reinforcement learning(RL) is a type of machine learning where an agent performs different actions in the environment, gets positively rewarded for actions that help in accomplishing the goals and negatively rewarded for actions that are not important. In this way the agent learns to play certain games or achieve certain goals by gaining experience.

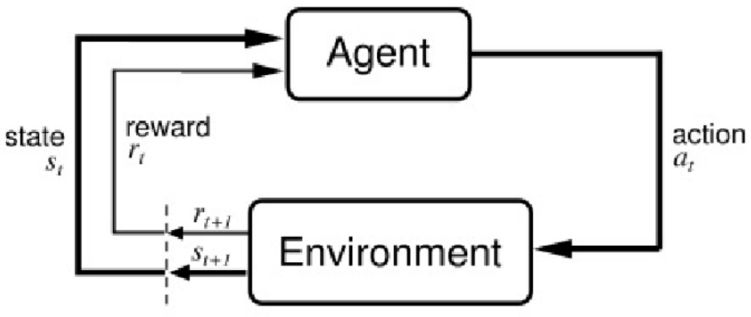

Figure 1: General Block diagram of RL

A basic reinforcement learning agent AI interacts with its environment in discrete time steps. At each time t,the agent receives the current state s_t and r_t.It then chooses an action a_t from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The goal of a reinforcement learning agent is to learn a policy: $\pi : A \times S \rightarrow [0, 1]$, $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$ which maximizes the expected cumulative reward.

3.2 Policy gradient

Policy gradient is a method by which an agent learns its policy i.e $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$ directly from the experiences i.e (s_t, a_t, s_{t+1}, r_t) collected in the environment. The algorithm tries to approximate the actual gradient of the expected return $\nabla_\theta(E)$ by collecting some sample trajectories and approximating the actual gradient with the average gradient of these sampled trajectories.The update of the parameters is in the direction of gradient of expected return because of which after every step approximately the expected return increases i.e better performance. The vanilla policy gradient uses the following equation for updating its parameters:

$$\nabla_\theta E(\theta) = \frac{1}{m} \sum_{\tau=0}^{m} \sum_{t=0}^{H} \nabla_\theta \log(Pr(a_t|s_t))\, R(\tau) \tag{1}$$

m = number of sample trajectories

τ = concerned trajectory

H = total number of steps in a trajectory

$R(\tau)$ = return of a particular trajectory

Q = network approximating expected return for given state and action

θ = parameters of value network

From the above equation, we can say that we take some sample trajectories and calculate their returns. Then, the gradients of log probabilities at each time step is added for a trajectory and multiplied by the return of that trajectory.Averaging this quantity for some samples of our policy nearly approximates the true gradient of the expected return which is used to update the policy.

Formula clearly shows that, the positive return for a trajectory makes the probability of taking these actions more probable and negative return for a trajectory takes a step in the direction of decreasing the probability of the trajectory and hence making it less probable. The reinforce-

ment of the positive reward trajectory suppressing of the negative reward trajectory makes the agent get high expected returns.

3.3 Value based methods

Value based methods don't directly give us policy. They give us certain value related to our action on a particular state. A higher value represents that the decision on the action based on the observation was good.

Q-learning is most used value based learning. In Q-learning, try to estimate the Q-value i.e $Q_\theta(a_t|s_t)$ which is expected return after taking action a_t in a particular state s_t. Q-value is also called action value. This gives us how valuable is a particular action a_t in a particular state s_t. So, strategy in Q-learning is to choose the action a_t with the highest Q-value in a particular state s_t.

The real problem is how do we estimate q-value for a particular action in a state. And the solution is TD-estimate. We define a network that gives Q-value of all possible actions given a state i.e $Q^i_\theta(s_t)$. And we calculate the target by adding the reward and Q-value of next state scaled by γ. This value is expected to be the actual Q-value. So, we find the error by subtracting this value from actual Q-value our network predicted and update network by backpropagating this error.

$$target = r(a_t|s_t) + \gamma max_i(Q^i_\theta(s_{t+1})) \tag{2}$$

$$TDerror = (target - Q_\theta(a_t|s_t))^2 \tag{3}$$

$$\theta = \theta - \alpha \nabla_\theta (TDerror) \tag{4}$$

3.4 Actor Critic Methods

Actor Critic Methods were introduced in order to improve the policy gradient algorithm. Since we use Monte-Carlo approach in policy gradient algorithm, every sample we take might be considerably different from each other. As a result, there is high variability in log probabilities as well as return of the trajectory. That results in high variance in gradients. High variance in gradients implies bad learning. And hence we can use actor critic to decrease the high variance in returns which results in decreasing variance in gradients.

There are two networks here. One is called Actor and the other is Critic. Actor is the one that outputs action given a state. Critic is the one that reduces variance of the return of trajectory. The critic network takes state and outputs the state value i.e how valuable is the given state. The output of the critic is also called baseline. The critic network is updated by TDerror. The

actor is updated by policy gradient except that this time return with high variance is replaced by difference of return and baseline with low variance.

The gradient is calculated like this in actor critic:

$$\nabla_\theta E(\theta) = \frac{1}{m}\sum_{\tau=0}^{m}\sum_{t=0}^{H}\nabla_\theta \log(Pr(a_t|s_t))\ (R(\tau) - b(s_t)) \tag{5}$$

where $b(s_t)$ may be obtained from a separate network.

3.5 Off Policy Learning

RL agents learn on the basis of experience. Algorithms like policy gradient algorithms learn without storing the experience in replay buffer. They just learn from those experiences at the time of exploring the environment. This method is called On policy learning.

Algorithms like Q-learning collect experiences into replay buffer, sample the experiences from the buffer and learn from it. Learning from the experiences that were collected and stored is Off policy learning. This has advantage of using same experience multiple times so that there is very efficient use of experience. There is better convergence while training a function approximator with this approach.

3.6 Deep Deterministic Policy Gradients(DDPG)

DDPG is a actor critic algorithm that uses actor network,critic network and respective target networks. Target network is used to find the target in the TDerror. Local networks are used to explore and exploit the environment.

First of all, experiences are collected by exploring the environment and stored in replay buffer. DDPG works by updating the critic network to properly give the Q-value i.e $Q_c(s_t, a_t)$ given a state s_t and action a_t. Then the actor network is updated such that a_t calculated by actor network has high value of $Q_c(s_t, a_t)$. That means the gradient of $Q_c(s_t, a_t)$ w.r.t actor network parameters is calculated and actor network is updated so that actions with high $Q_c(s_t, a_t)$ is chosen by actor. And finally target networks are updated either by hard update every n updates or by soft update every time we update the local network.

critic update :

$$target = r(s_t, a_t) + \gamma \cdot dQ_{\theta_{ct}}(s_{t+1}, \pi_{\theta_{at}}(s_t)) \tag{6}$$

$$TDerror = (target - Q_{\theta_{cl}}(s_t, a_t))^2 \tag{7}$$

$$\theta_{cl} = \theta_{cl} - \alpha \nabla_{\theta_{cl}} TDerror \tag{8}$$

Actor update :

$$a_t = \pi_{al}(s_t) \tag{9}$$

$$\nabla_{\theta_{al}}(Q_{\theta_{cl}}(s_t, a_t)) = \nabla_{a_t}(Q(s_t, a_t)) \cdot \nabla_{\theta_{al}}(a_t) \tag{10}$$

$$\theta_{al} = \theta_{al} + \alpha \nabla_{\theta_{al}} Q_{\theta_{cl}}(s_t, a_t) \tag{11}$$

Since DDPG is specifically designed for continuous action spaces, performance on discrete action spaces may not be as good as in continuous action spaces.

3.7 Multi-Agent Deep Deterministic Policy Gradients(MADDPG)

MADDPG is a multi agent version of DDPG. We have multiple agents and each agent has local and target actor critic networks. We add observations and actions of all agents to the input of critic network here so that there is stationarity of environment. If actions and states of all agents are not added then, the agents are bound to misunderstand the reward they got resulting in erroneous behavior i.e non-stationarity. MADDPG can be implemented by following steps:

Sample a random minibatch of S samples $(s^l a^l, r^l, s'^l)$ from replay buffer, D
Set $y_l = r_i^l + \gamma Q_i^{\mu'}(s'^k, a'_1, ..., a'_N)|_{a'_k = \mu'_k(a_k^j)}$
Update critic by minimizing the loss $\mathcal{L} = \frac{1}{S}\sum_j (y^j - Q_i^{\mu}(s_j, a_1^j, ..., a_N^j))^2$
Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j) \nabla_{a_i} Q_i^{\mu}(s_j, a_1^j, ..., a_i, ...a_N^j)|_{a_i = \mu_i(s_i^j)} \tag{12}$$

3.8 Self Play

A common approach to train agents in adversarial scenarios is via self-play. The idea behind self-play is that between two teams, only one of the teams is learning(is trained) while the other acts on copies of past versions of the actors of learning team. Self play promotes exploration and exploitation because if the opponent is too weak or too strong learning becomes difficult.

Self play is implemented by maintaining a network bank, B of fixed length. This network bank holds periodic copies of the actor networks$(\mu_i, ...\mu_{N/2})$ of the learning team. The rate of saving copies to the bank is determined by a parameter, we will term $save - network - step$. Also, according to a different parameter, $swap - network - step$ the current actor networks of the non learning-team is changed to a random sample of the network bank. The network bank operates like a queue and discards the oldest entry of networks when new entries are added to a

full network bank.

4 Methodology

4.1 System Block diagram

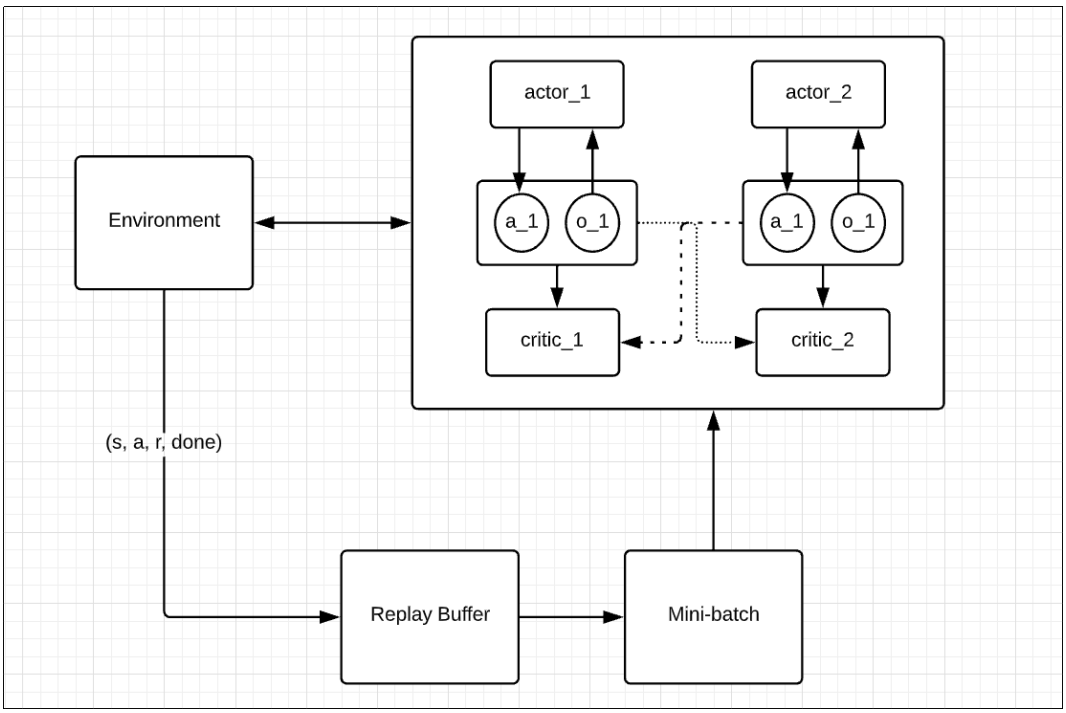

Figure 2: System Block Diagram

The system consists of an environment with four reinforcement learning agents that perceive and interpret its environment, take action and learn through the reward obtained. Each agent has a set of observations(state) and actions. The state of the environment observed by an agent is passed through two neural networks - actor neural network and critic neural network. The actor takes in a state and returns an action to take at that state while the critic takes in an action and a state and returns a q-value that determines how well it's doing. The state of the agent, the action it took and the reward obtained at each time step are stored on a replay buffer. A random batch of experience is sampled from the replay buffer and passed to the actor critic network. The loss is calculated and policy is optimized to minimize the loss.

4.2 Observation and Action Space

4.2.1 Environment

We used the dodgeball environment provided by Unity and edited it to create the curricula for self learning. The size of the environment was reduced to reduce the amount of exploration

required to solve the environment. The reward structure and size of observations were also changed as per needed at each step of curriculum.

4.2.2 Observation Space

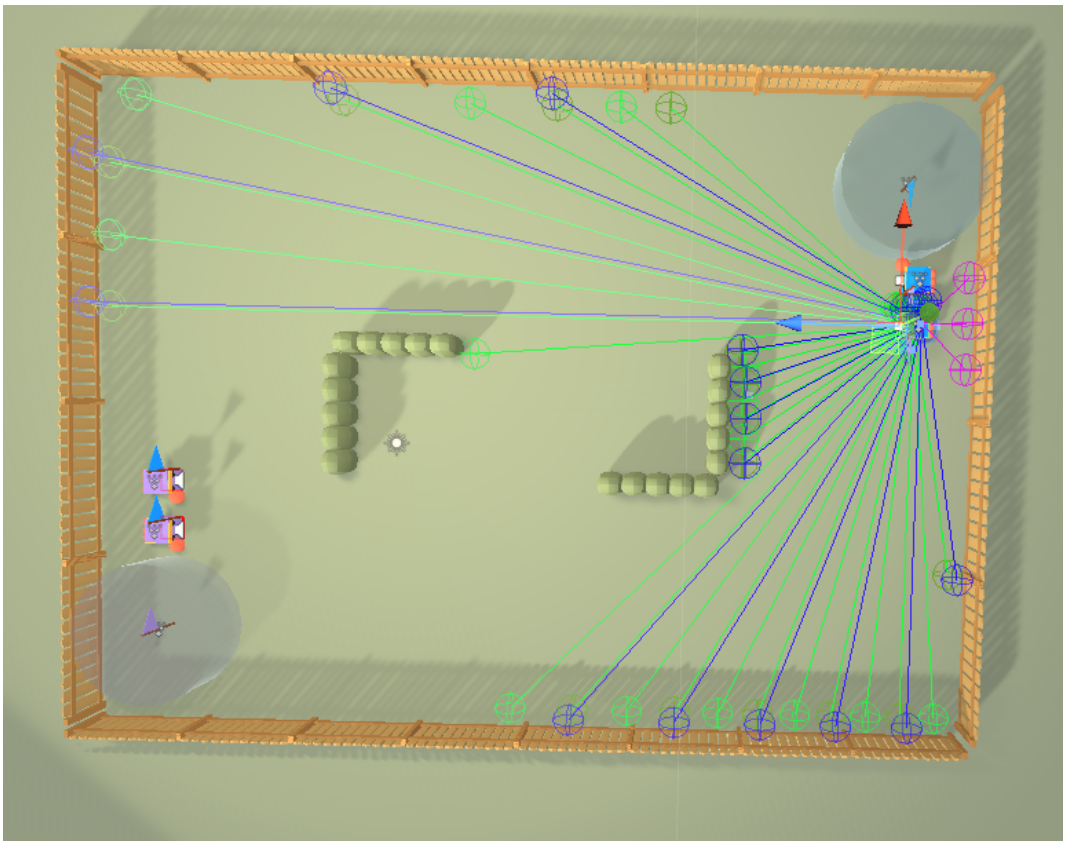

Figure 3: Ray perception in 3D

A number of rays are shot from an agent position at various angles and if a ray hit a target(detectable tags) successfully then the hit information is is written to a subset of float arrays.There are various observation spec that returns hit information of a ray.

- AgentRayPerceptionSensor: It is responsible for detecting 4 detectable tags- purpleAgent, blueAgent, purpleAgentFront, blueAgentFront and has a shape of(256,).
- BallRayPerceptionSensor: It is responsible for detecting 2 detectable tags- dodgeBallActive, dodgeballPickup and has a shape of(84,).
- RayPerceptionSensorBack:It is responsible for detecting 2 detectable tags- wall, bush and has a shape of(12,).
- WallRayPerceptionSensor:It is responsible for detecting 4 detectable tags- wall, bush, purpleFlag, blueFlag and has a shape of(126,).

Each element in the rayAngles array determines a sublist of data to the observation. The sublist contains the observation data for a single cast.The list is composed of the following:

- A one-hot encoding for detectable tags. If DetectableTags.Length = n, the first n elements of the sublist will be a one-hot encoding of the detectableTag that was hit, or all zeroes otherwise.
- The 'numDetectableTags' element of the sublist will be 1 if the ray missed everything, or 0 if it hit target (detectable or not).
- The 'numDetectableTags+1' element of the sublist will contain the normalized distance to the object hit, or 1.0 if nothing was hit.

Each agent also has a buffer sensor named 'BufferSensorAgents' of shape(2,8) where its state information as well as other agent's state information are stored. Its state information includes the position of the agent, the number of balls it currently holds, the number of hits it can take before being eliminated, as well as information about the flags. Agents use this information to strategize and determine their chances of winning. Other's agent information includes position and health of the agent's teammates, and whether any of them are holding a flag.

An agent also has a vector sensor named 'VectorSensor_size20' with shape of (20,). Its store information of whether an agent is stunned or not, the agents remaining hit point, location to base, whether it has enemy flag, the remaining cooldown, the balls currently active.

The total observation size for one agent is 504 upon concatenating all observations.

4.2.3 Action Space

There are a total of 5 actions that an agent can perform.This environment makes use of hybrid actions, which are a mix of continuous and discrete actions. There are two discrete action and 3 continuous actions. Each discrete action has two branch and allows an agent to throw a ball or to dash as the ball approaches. The 3 continuous allows an agent to move forward, move sideways, and rotate at certain angle. The range of values for continuous actions is (-1,1), the sign representing direction and magnitude linear or angular velocity.

4.3 Network Architecture

Actor network consists of a series of fully connected layers that map an agent's observations to actions. The observations of an agent are passed to a first fully connected layer of dimension 504 and standardized output using batch normalization. Then its output is passed to the second

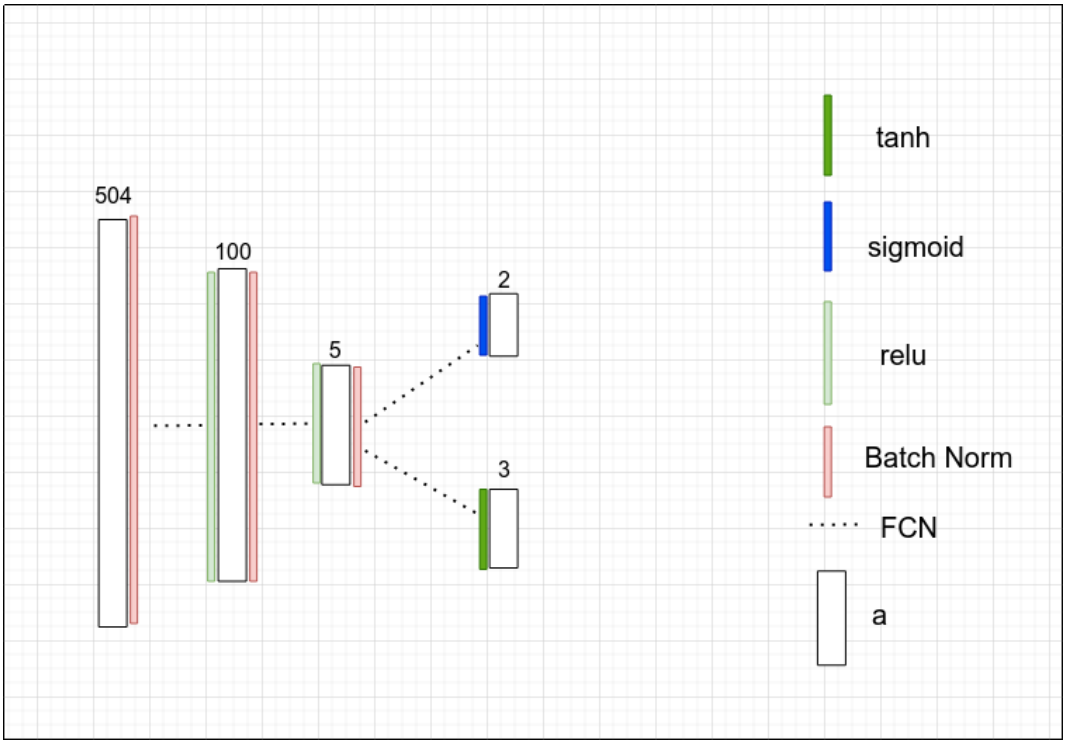

Figure 4: Actor Network Architecture

connected layer of dimension 100 with relu activation. Then we will have an output of dimension 5. The first two output with sigmoid activation gives the discrete action and the remaining three with tanh activation gives the continuous action.

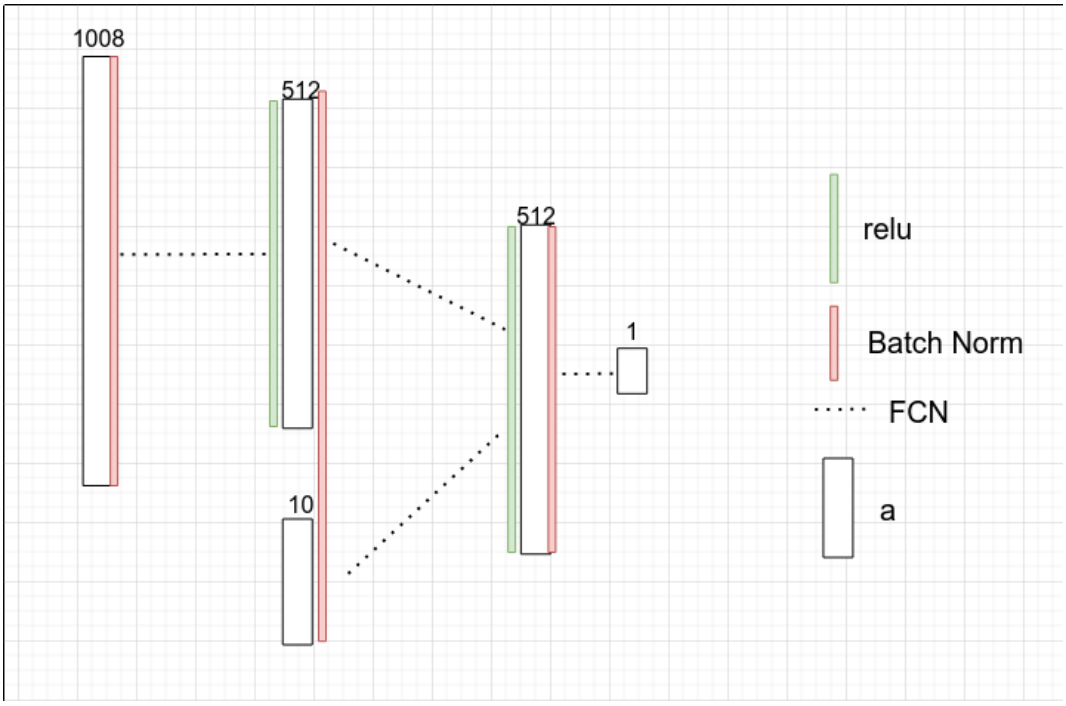

Figure 5: Critic Network Architecture

Critic network consists of a series of fully connected layers that map an agent's concate-

nated observation and action pair to a q-value. The agent's observations are passed to a first fully connected layer of dimension 1008 and batch normalized. Then it's output is passed to the second connected layer of dimension 512 with relu activation. The actions are also batch normalized. Then the output is concatenated and the concatenated agent's observations and actions are passed to the third layer with relu activation and batch normalization is used on the result. A single output obtained gives the q-value.

4.4 Algorithm

MADDPG, or Multi-agent DDPG, extends DDPG into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. This algorithm deals with decision making process of multiple agents at the same time. The pseudocode of the variation of MADDPG used in this project is as described in Algorithm 1 below.

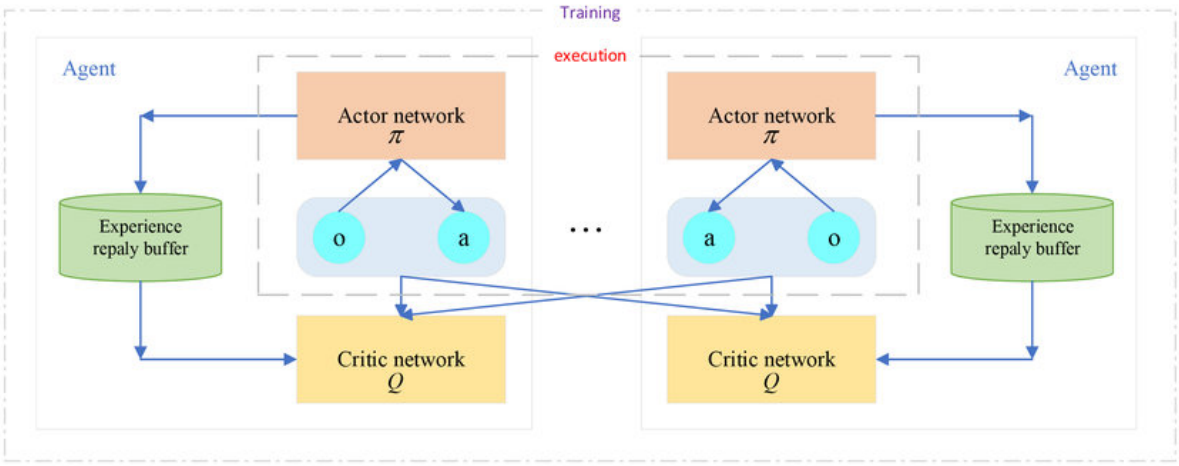

Figure 6: Schematic of MADDPG algorithm

The critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner.

Algorithm 1: Self Play $MADDPG$

for *episode* $= 1$ *to* M **do**
- Set team 1 as learning-team and team 2 as opponent-team
- Initialize a random process $\mathcal{N}$ for action exploration
- Initialize empty network bank B
- Receive initial state s
- **for** $t = 1$ *to max-episode-length* **do**
  - **for** *each agent* i *in learning-team* **do**
    - select action $a_i = \mu_{\theta_i}(s_i) + \mathcal{N}_t$ w.r.t the current policy and exploration
  - **end**
  - **for** *each agent* j *in opponent-team* **do**
    - select action $a_j = \mu_{\theta_j}(s_j)$ w.r.t the current policy
  - **end**
  - Execute actions $a = (a_i, ..., a_N)$ and observe reward r and new state s'
  - Store (s, a, r, s') in replay buffer D
  - $x \leftarrow s'$
  - **for** *agent* $i = 1$ *to* $N/2$ *in learning team* **do**
    - Sample a random minibatch of S samples $(s^l a^l, r^l, s'^l)$ from D
    - Set $y_l = r_i^l + \gamma Q_i^{\mu'}(s'^k, a'_1, ..., a'_N)|_{a'_k = \mu'_k(a_k^j)}$
    - Update critic by minimizing the loss $\mathcal{L} = \frac{1}{S}\sum_j (y^j - Q_i^{\mu}(s_j, a_1^j, ..., a_{N/2}^j))^2$
    - Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(s_i^j)\nabla_{a_i} Q_i^{\mu}(s_j, a_1^j, ..., a_i, ...a_{N/2}^j)|_{a_i = \mu_i(s_i^j)}$$

  - **end**
  - Update target network parameters for each agent i in learning-team:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

- **end**
- Update elo rating based on result of episode
- **if** *episode* $\%$ *save-network-step* $== 0$ **then**
  - Enqueue current actor networks $(\mu_1, ..., \mu_{N/2})$ to network bank B
- **end**
- **if** *episode* $\%$ *swap-team-step* $== 0$ **then**
  - Sample $(\mu_1, ..., \mu_{N/2})$ from network bank B and assign it as actor networks of opponent $(\mu_{N/2}, ..., \mu_N)$
- **end**

end

where,

$\mu =$ actor network

$\theta =$ parameters of network

$Q =$ value(critic) network

$o_i = s_i =$ observation of i^{th} agent

$N/2 =$ total number of agents in a team

$\tau =$ soft-update parameter

4.5 Training Agents

4.5.1 Self Play

Because of the symmetric and adversarial nature of the environment, self play was employed for training, where one of the teams(purple) was trained while the other team(blue) acted based on periodic copies of the actor networks of the learning team.

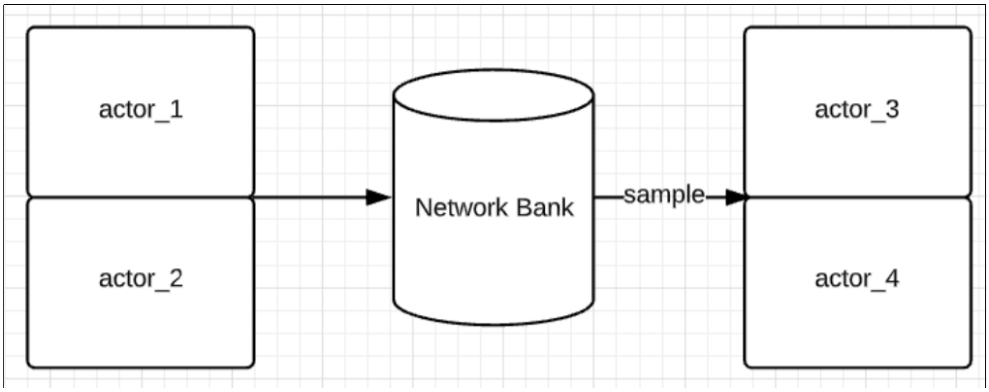

Figure 7: Self Play

4.5.2 Curriculum Learning

The agents were trained successively on three different environments with increasing difficulty. The touch zone environment rewarded agents of a team if a member successfully touched opponent's zone. The touch flag environment changed that reward to when a team member successfully touched opponent's flag inside the opponent's zone. The return flag environment rewarded agents for successfully picking up(touching) opponent's flag as well as bringing it back to own's zone. Any agent was given a small reward for successfully hitting an opponent with a ball.

Rewards/ Environment	Hit opponent	Touch enemy zone	Touch enemy flag	Return enemy flag	Lose Game
Touch Zone	0.02	2	-	-	-
Touch Flag	0.02	0	2	-	-
Return Flag	0.02	0	2	10	-5

Table 1: Reward Structure

4.5.3 Action Generation

During training, the outputs of the actor network for continuous actions are treated as means of gaussian distributions with standard deviations equal to a parameter defined as the noise rate. This is required to introduce exploration into the training. The noise rate was decreased exponentially upto a certain limit and then kept constant. The outputs corresponding to the discrete actions are treated as probability values of a bernoulli distribution. The sampled action value is then stored in the replay buffer. During evaluation, the outputs for continuous actions are used in a deterministic manner.

$$\pi(s_i) \sim \mathcal{N}(\pi(s_i), \textit{noise\_rate}) \text{ for continuous actions} \tag{13}$$

$$\pi(s_i) \sim \mathcal{B}(1, \pi(s_i)) \text{ for discrete actions} \tag{14}$$

4.6 Evaluation Metrics

4.6.1 Maximum Average Return

The overall learning metrics can be evaluated by comparing the performance of agents on the target task after following the complete curriculum, versus performance following no curriculum (i.e., learning from scratch). One of the popular method is maximum average return where returns of the each team is compared and average of the maximum value is considered which when follows the increasing trend is said to be in progress. The maximum average return is expected to rise steadily and then settle at the maximum possible return in an episode.

4.6.2 Elo Rating

Elo Rating is a method of comparing the relative skill of the agents in multi-agent environment. The difference in the rating of two team serves as the probability of the outcome of a match. Suppose a team having lowest rating wins the game, its rating increases exponentially as compared to the opponent losing team and if it loses, its value decreases only by the small proportion. Approximately equal rating of both team ensures competitive behaviour in the Reinforcement learning. If a player and its opponent has a elo rating of R_A and R_B respectively, then expected score of both team will be given as,

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}} \tag{15}$$

$$E_B = \frac{1}{1 + 10^{\frac{R_A - R_B}{400}}} \tag{16}$$

Now, Given the actual score S_A and S_B and maximum possible adjustment per game K, the update equation for elo rating can be calculated as,

$$R'_A = R_A + K.(S_A - E_A) \tag{17}$$

$$R'_B = R_B + K.(S_B - E_B) \tag{18}$$

4.6.3 Episode Length

Episode length is the another metrics that evaluates how good is the performance of the agents in its environment. It determines how fast the agents learn during the training. A steady decrease in episode length means that the trained agents are getting faster at performing the task.

5 Results

5.1 Training

While training multi agents using MADDPG for about 1500 episodes, the results were pretty satisfactory for the curriculum learning rather for learning from scratch. Due to the diverse rewards structure, it takes about 10% more time than that of curriculum learning for training up to same episode length. All the evaluation metrics were satisfied to the criteria. With the increasing time steps, average maximum rewards and Elo rating for both agents goes on increasing while the average episode length decreases with more and more intelligence behaviour seen on the agents. During curriculum learning, network parameters from current environment is copied to its successor having an increasing difficulty to ensure its faster convergence. So here, only the most difficult environment is taken to maintain the same reward structure for both learning methods. Since the learning from scratch needs more exploration, initial noise rate is also taken larger. All presented results are based on comparison between curriculum learning and learning from scratch.

Fig 15 and Fig 16 shows the win rate for curriculum learning to be greater than learning from scratch as it needs a lot of time to maximize its win rate and also we can see that the average return of those of curriculum learning is stabilized more and also it converges faster which can also be illustrated from actor loss as shown in Fig 17 and 18.

During the evaluation steps, both the team competitively acts upon each other with the aim

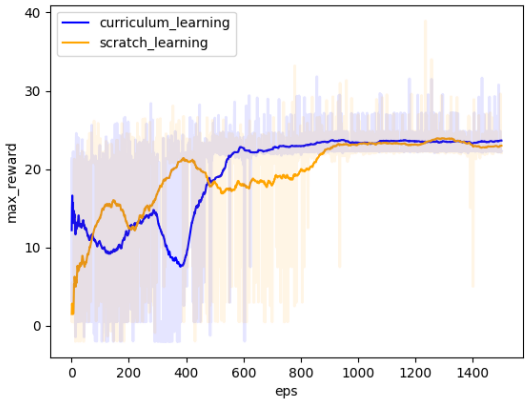

Figure 8: Maximum average returns

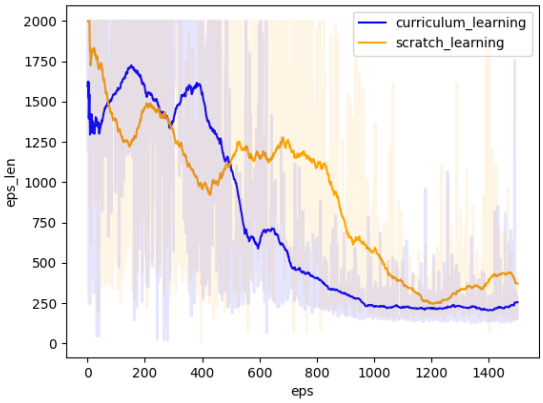

Figure 9: Mean episode length

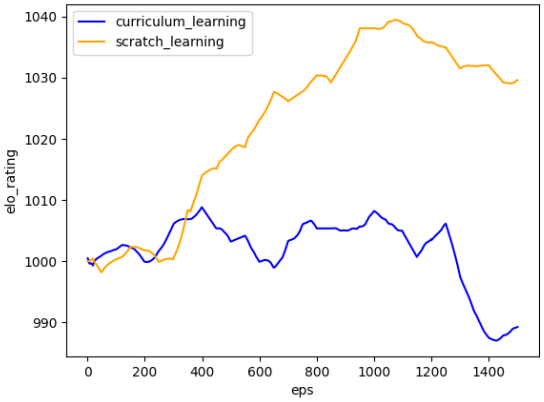

Figure 10: Average elo rating for blue team

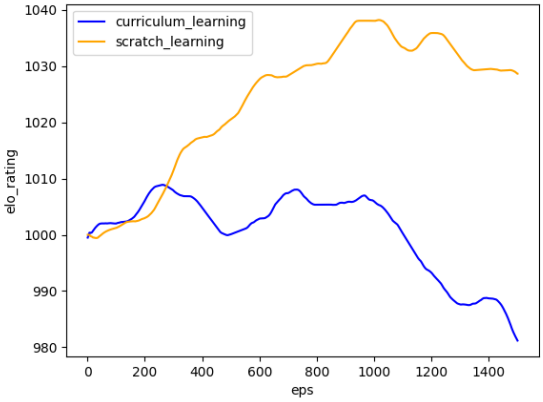

Figure 11: Average elo rating for purple team

to maximize the rewards. It includes a two main action i.e shortest time to return the flag to home and hitting opponent team with a maximum amount of ball. Since, even for the human,hitting the moving opponent team needs a lot of skill and training, the results for that is limited to about only 2, 3 hit per 10 episodes. But the skill to return flag is relatively much good in compared to learning from scratch for curriculum learning.

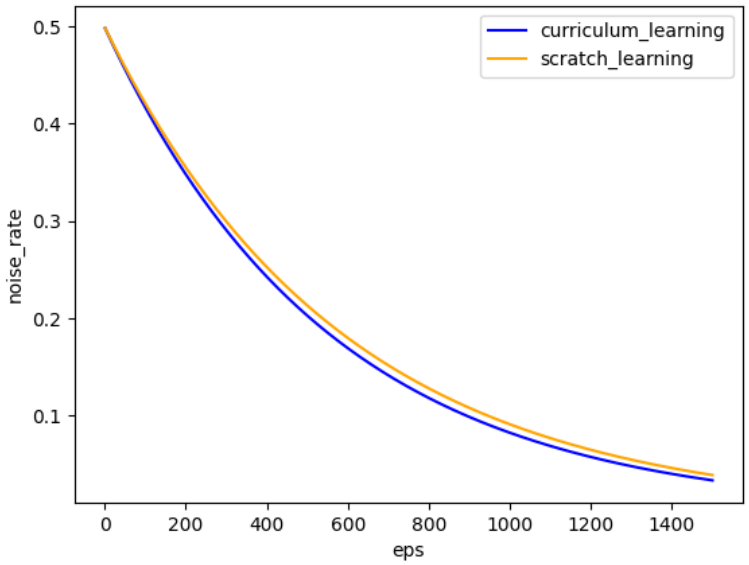

Figure 12: Noise rate decrease with increasing episodes

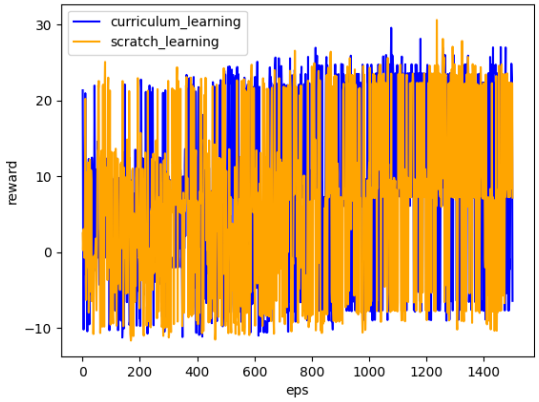

Figure 13: Return for blue team

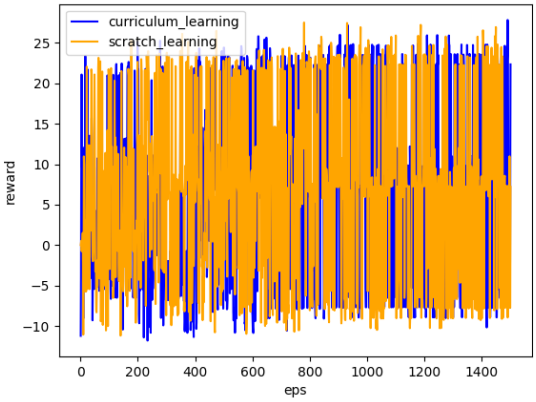

Figure 14: Return for purple team

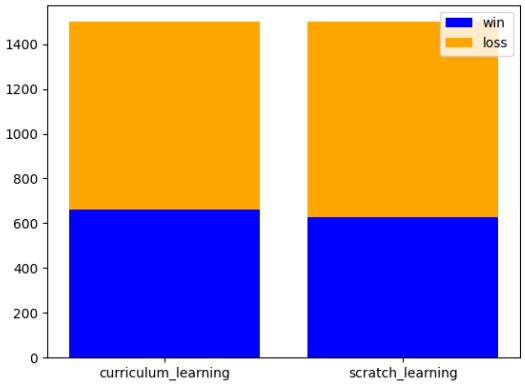

Figure 15: win rate for blue team

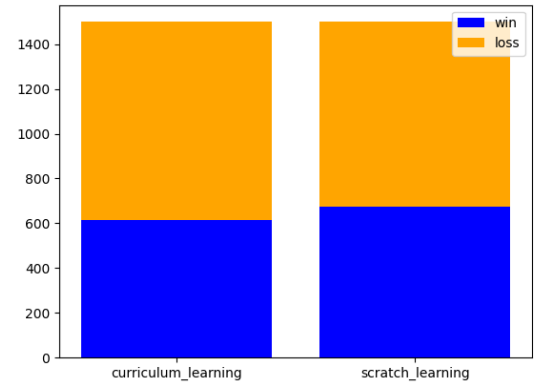

Figure 16: win rate for purple team

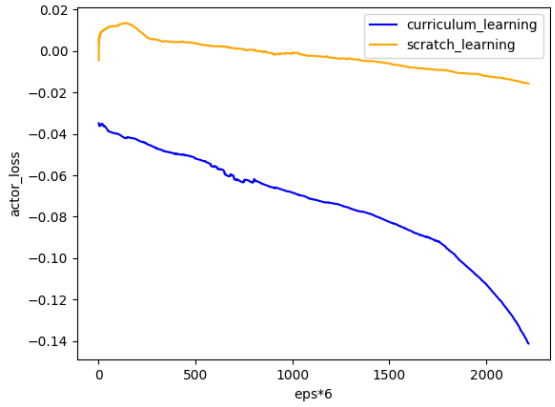

Figure 17: actor loss for purple $agent_1$

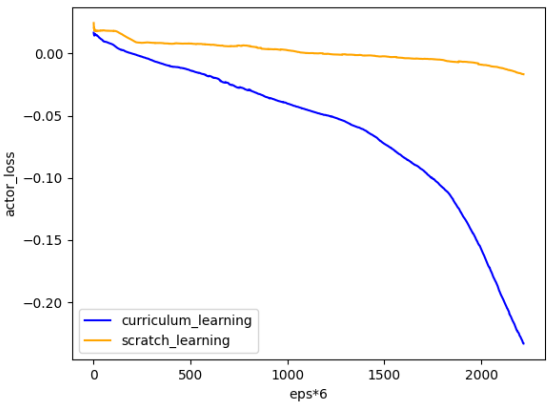

Figure 18: actor loss for purple $agent_2$

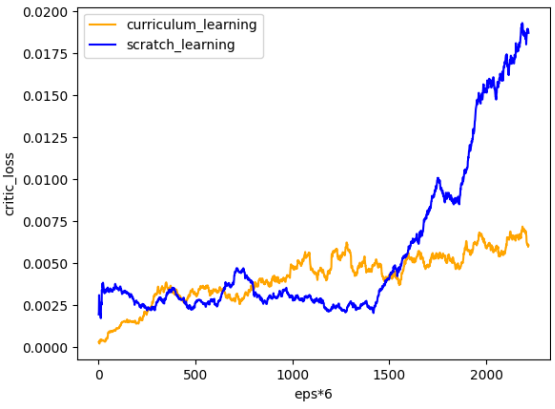

Figure 19: critic loss for purple $agent_1$

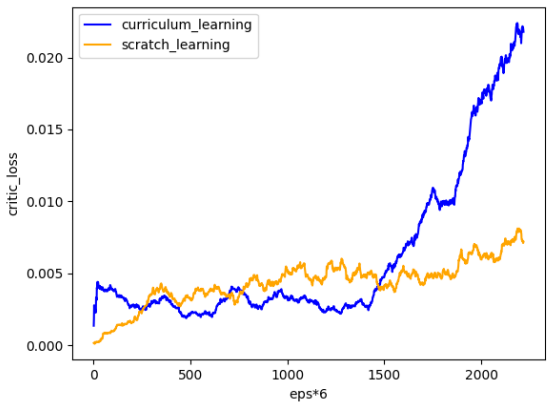

Figure 20: critic loss for purple $agent_2$

5.2 Testing

After training session was completed,the trained actor and critic network was used to test the performance of the agents for about 200 episode. Self play is integrated in the blue team which reduces a lot of time to train itself from the network of other agents. There is not much difference in the graph of win rate for both learning methods. After analyzing the results, win rate for curriculum learning and learning from scratch are found to be 100% and 99.5% respectively which can also be illustrated from 23 & 24.

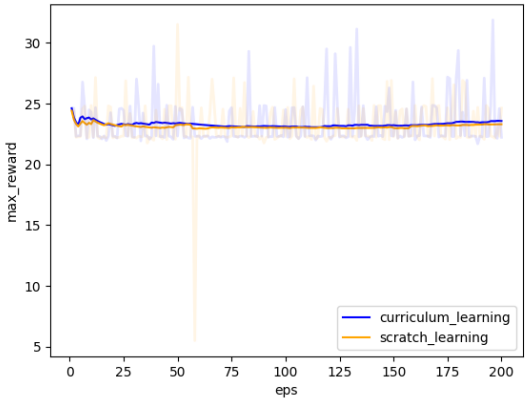

Figure 21: Maximum average return

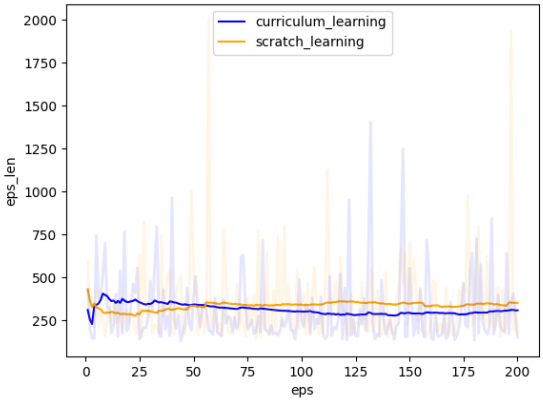

Figure 22: Average episode length

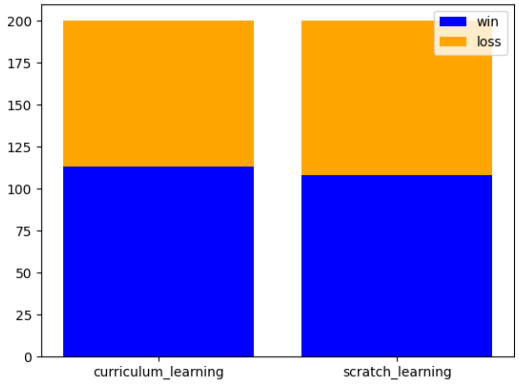

Figure 23: win rate for blue team

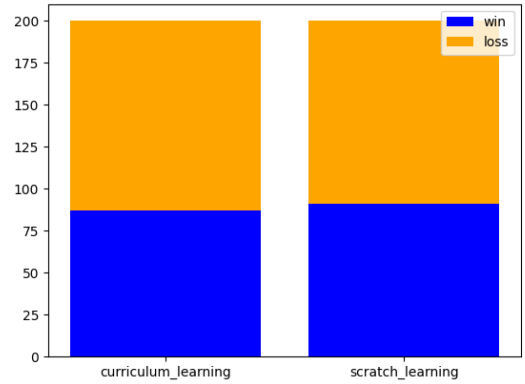

Figure 24: win rate for purple team

6 Conclusion

We have proposed the methods of deep RL having centralized critics based on the observation and actions of all agents which does not need complete observability, rather uses only the local observation to learn the optimum policy. On the other hand, curriculum learning along with self

play provides faster convergence than that of learning from scratch in the return_flag environment. Agents also learn to cooperate and coordinate with each other in order to maximize their objectives.

6.1 Limitation

Although, self play with curriculum learning performs quite well, there are series of challenges that limits the overall performance of the system. The design of curricula is often the most difficult part which must ensure that each sample presented to the agent is suitable given its current ability [14]. It can be automatically generated by ordering samples from the target tasks or iteratively choosing the intermediate tasks with increasing difficulty. Another major issues in the most multi agents system is the problem of the non-stationarity which changes the overall setting of the agents in the environment. On the other hand, scalabilty also affect the computational time and memory which even results in inefficiency of such algorithms as described in [2].

6.2 Future Enhancements

MADDPG algorithm is widely used multi agent RL technique which is also equally applicable to the real world environment where the agents are to accomplish certain tasks in multi-agent scenarios. In spite of that, there are still a lot of limitations and flaws in this method of learning. This method of learning is not invariant to the number of agents in the enviroment. In an environment where the number of agents may change dynamically, MADDPG performs poorly. A solution to this is to include a recurrent layer at the input of critic network to handle a variable number of actors.Another approach is to use decentralized setting with networked agents in order to reduce scalabilty.Even so, a large number of agents makes the learning memory and computationally expensive. Another approach might be to use a fixed number of actors per critic in a locally networked manner where the actors are close enough to introduce non-stationarity if not handled by the same critic.

References

[1] Per-Arne Andersen, Morten Goodwin, and Ole-Christoffer Granmo. Deep rts: A game environment for deep reinforcement learning in real-time strategy games. pages 1–8, 08 2018.

[2] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.

[3] Deepmind. Deepmind ai reduces google data centre cost (https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40). 2017.

[4] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[5] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning, 2016.

[6] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, and et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, May 2019.

[7] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas, 2017.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[9] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[10] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. *AAAI 2012*, 02 2013.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 12 2013.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[13] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations, 2018.

[14] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

[15] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games, 2017.

[16] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[17] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation, 2016.

Hyperparameters

Hyperparameters	Value
Max. episode length	2000 steps
Learning rate(actor)	0.0001
Learning rate(critic)	0.001
Starting noise rate	0.5
γ	0.99
τ	0.01
Buffer size	200000
Batch size	1000
Training rate	100 steps
Initial Elo rating	1000
Size of network bank	10
Swap team every	50 episodes
Save learning team actors every	20 episodes
Probability to select latest opponent	0.5

Table 2: Hyperparameters used in the project