MARMARA UNIVERSITY FACULTY OF ENGINEERING

COMPUTER ENGINEERING

CSE4197 – Analysis and Design Document

Title of the Project

Multi Agent Reinforcement Learning

Group Members

Hüseyincan ERBAYRAKTAR 150114015

Berk DAĞLI 150115039

Supervised by

Assoc. Prof. Borahan TÜMER

02.01.2019

1. Introduction

Reinforcement Learning (RL) is a learning approach similar to learning in real life. All mammals including humans take actions and learn from the feedback coming from their environments. Classic RL algorithms with a single agent proved promising in solving many complex problems where sequential decision making is involved. However, many tasks in the nature require teamwork and are mostly done under the adversary intentions of another team. This means that fulfilling such tasks require coming up with strategies involving cooperation of multiple agents within the same team and competition with another set of agents from an adversary team. Inspiring from such tasks in the nature solved by mammals, complex problems involving multiple agents necessitates devising solutions with multiple agents that both cooperate and compete with other agents, in respective order, from the same team and the other team. Multi-agent Reinforcement Learning (MARL) is one such paradigm that establishes the necessary framework. In MARL, there are multiple agents interacting with the environment. Those agents can also interact with each other, which means they can either be cooperative or competitive with each other.

1.1. Problem Description and Motivation

In RL, most of the time, the problem involves a single agent interacting with an static environment as seen in Figure 1. But in the real world problems, the environment is most of the time dynamic with multiple agents. Sometimes a single task can be too complex for a single agent to deal with. In that case they can handle it cooperatively. An example from nature is wolves hunting in packs. Wolves in a pack are individually same but their tasks during the hunt is different. By combining these different tasks, they can handle more difficult tasks. With this collaboration, they have become one of the most successful animals in hunting.

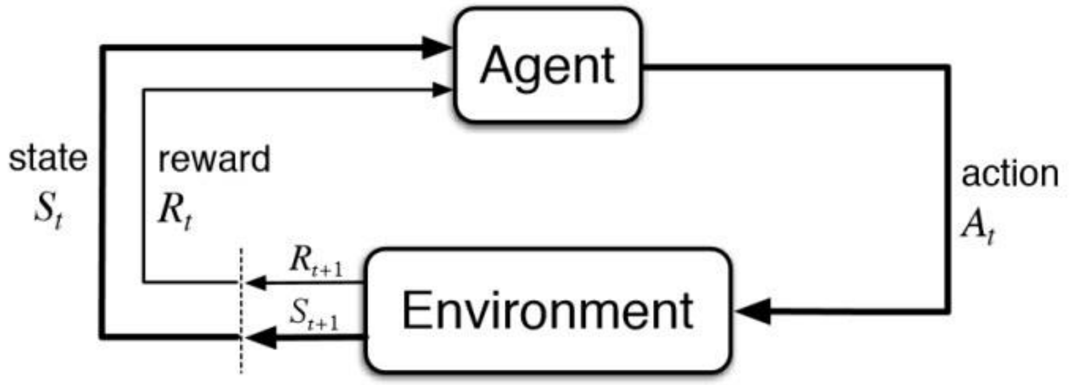


Figure 1: SARL diagram[12]

In multi-agent systems(MASs), it is considered that there are many agents interacting with the environment as seen in Figure 2. Therefore MASs contain agents that act cooperatively, competitively, independently or in a combination of these.

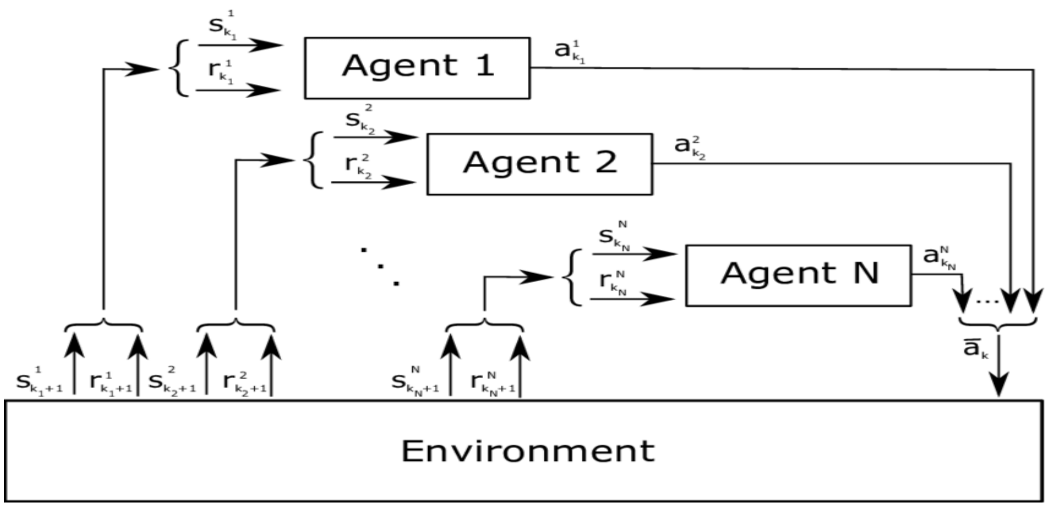


Figure 2: MARL diagram[11]

In order to apply RL approach to the real world problems, we must develop appropriate RL algorithms for it. As we stated above, there are lots of real world MASs for example manufacturing automation systems, air traffic control, medical diagnosis, routers in the internet etc[15]. In these kind of systems, agents can be developed independently of each other and can be later used as components of new systems. For example in medical diagnosis systems, different agents would specialize in different domains and then can be used together with each other[15]. If we want to solve a problem in any MAS, using MARL would be more appropriate. Because MARL takes cooperative and competitive behaviours into consideration.

In these systems, agents can benefit from game theory to select their actions. That means they consider the possible actions of the other agents in the environment.

		Pl 2	
		Left	Right
Pl 1	Left	4,2	6,2
	Right	3,3	2,5

Figure 3: Example of Game Theory

For example in Figure 3, its assumed that player 1 and player 2 are opponents. the numbers represent gain of player 1 and player 2 respectively. Player 1 assumes that player 2 would take right action because it gains 5 which is maximum for player 2 therefore player 1 will select left action to minimize player 2's gain and to maximize its own gain. This scenario is an example of competitive behaviour in a MAS.

In our work, we will try to show that MARL with interacting agent, which is our second approach, gives better results than the Single-agent RL where agents are independent, which is our first approach, in MAS. We will study these two approaches to RL using a soccer simulation as it is a good example where players in the same team behave cooperatively while opposing teams show competitive behaviour. Therefore a soccer simulation will be a suitable way to show the superiority of MARL.

In our work we plan to make the agents cooperate to accomplish tasks that can not be effectively handled with independent agents.

1.2. Scope of the Project

In our work we are just planning to show superiority of MARL approach only in soccer environment.

Our work is focused on soccer environment. The environment will be fully observable by all agents and deterministic, which means transition probability of an action is 1. Also the environment will be dynamic for the reason that there are multiple agents changing the environment. This will provide a dynamism.

The actions of the agents will be same in two approaches. The state space will be continuous for both approaches because there are lots of state action pairs. In both approaches, the state-action pairs will consist of state representation and action of the agent.

There will be a decision maker in MARL which will get top N actions from each cooperative agent in the team and determines all possible actions of agents in opponent team. It will evaluate all the combinations of actions from cooperative and competitive agents. Finally it determines cooperative agents action which maximizes the team reward.

The simulation that we are planning to use will be two dimensional. All the agents will have identical features. The agents will be represented with circles. In the simulation agents can turn anywhere, and can kick the ball onward where they are facing. An episode will end when one of the team reaches three goals.

In the both approaches, the number of agents will be fixed and we do not guarantee that it is applicable to the number exceeding that.

1.3. Definitions, acronyms and abbreviations

RL	Reinforcement Learning
MARL	Multi-agent Reinforcement Learning
SARL	Single-agent Reinforcement Learning
DQN	Deep Q Network
DDQN	Double Deep Q Network
MAS	Multi-agent System

2. Related Work

In the work of Ming Tan[4], author studied that cooperative agents could give better results than the agents that behave independently. Also in his work, he tried to answer what could be the cost of making the agents cooperate. He categorized them in three approaches which are (1) sharing sensation, (2) sharing episodes and (3) sharing learned policies. He stated that sharing sensation could be suitable if used effectively. He also stated that second and third approaches would speed up learning process at the cost of communication. Finally he analyzed that in multi agent systems the learning would be slow at the beginning.

In the work of Lauer and Riedmiller[5], they focused on distributed RL in cooperative multi-agent decision process. They stated that agents in single agent RL do not have information about behaviour of teammates but it should be exact opposite. Besides they proposed a model free distributed Q Learning algorithm on multi-agent decision processes. Moreover they showed that this algorithm can find optimal policy in deterministic environments. Also an interesting point in their work is that cooperating agents do not need any communication links.

In the work of Nagayuki et al.[6], authors stated that in multi-agent environments correctness of an agents behaviour depends on the behaviours of the other agents.

They told that in classical RL approaches only one agent changes the environment and the other agents' actions are ignored. Therefore they studied on a two-agent cooperation problem. In their work, one of the agents determines its action based on prediction of the other agents action. In order to predict, the agent looks to the internal model of the other agent. Finally the authors claim that by their approach they achieved a cooperative behaviour that works as intended.

In the work of DeepMind AI research team[7], researchers have developed a multi-agent environment where agents try to capture flag of the opponent team and get it to their own starting area. In their environment, teams are composed of two players. They have compared several situations like 2 agents vs 2 human players or 2 agents vs 1 agent and 1 human etc. After training about 180K games, the agents have become stronger than a strong human player and after 250-300K games, the strength between agents and strong human players have increased significantly.

In the work of Peter Stone[13], author studied about building artificially intelligent agents for complex multiagent control tasks. He chose robotic soccer as multiagent system. He tried to solve the problem with Layered Learning approach. Layered Learning approach is a hierarchical Machine Learning paradigm. This approach consists of three interconnected layers. In the first layer, agents learn individual skills. In the second layer, agents learn multiagent behaviour. In the last layer, agents learn team behaviour.

3. System Design

3.1. System Model

In our project, we assume that every team consists of two players. One of the teams agents will learn to play soccer using SARL approach, other ones agents will learn using MARL approach.

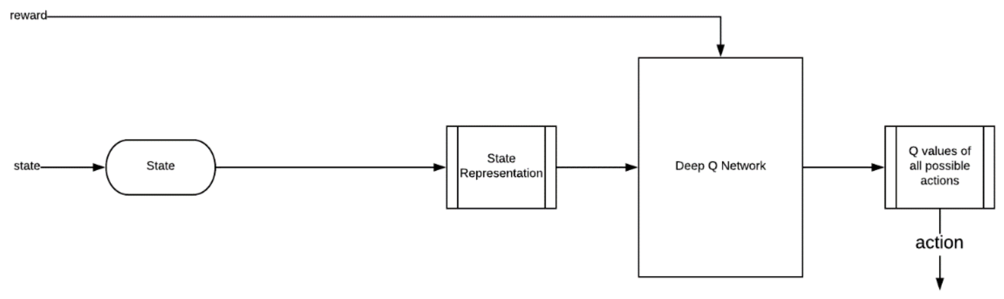


Figure 4: Internal structure of Agents 3 & 4

As seen in figure 4, Agent 3 and Agent 4 are in the team that uses SARL approach. They both get reward and state from the environment and select action according to highest Q value.

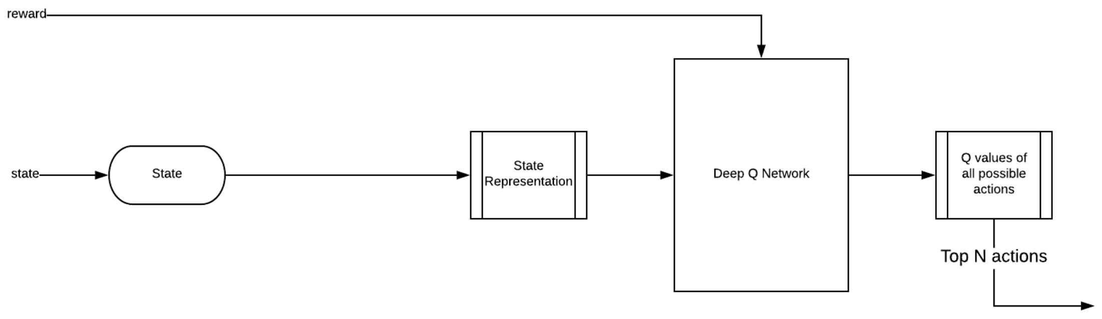


Figure 5: Internal structure of Agents 1 & 2

As seen in figure 5, Agent 1 and Agent 2 are in the team that uses MARL approach. In MARL team, there is a decision maker. It takes N actions of Agent 1 and Agent 2 which has highest Q values. Also it takes state information from the environment to determine final actions of Agent 1 and Agent 2.

Also for MARL team, there is a Decision Maker module as seen in figure 6 to provide cooperative and competitive behavior.

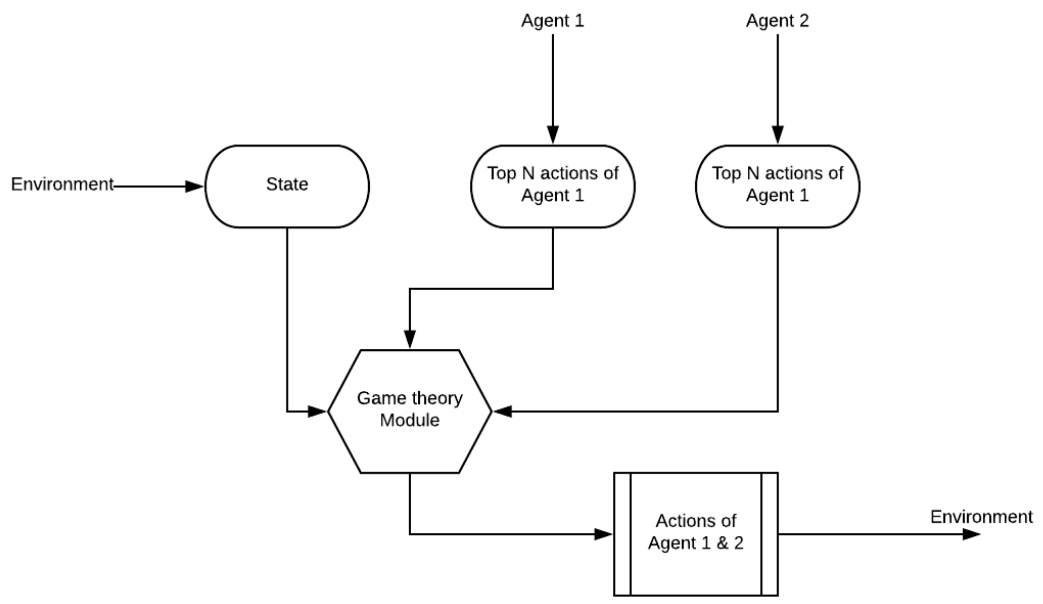
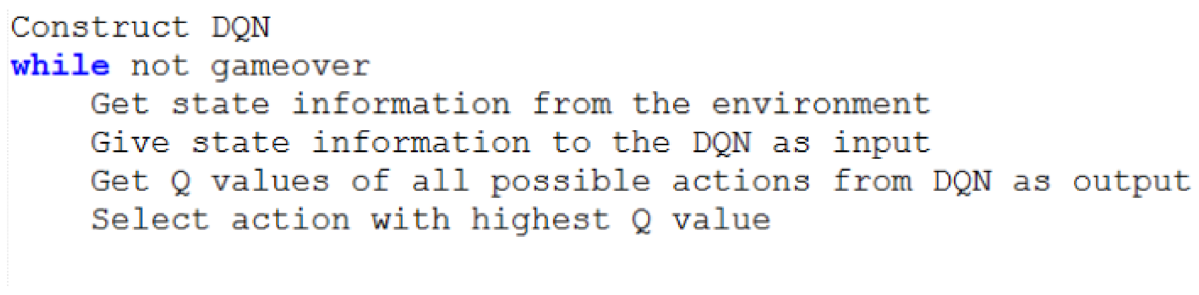


Figure 6: Internal structure of Decision Maker

3.2. Flowchart and pseudo code of proposed algorithms

Pseudocode for SARL agent:

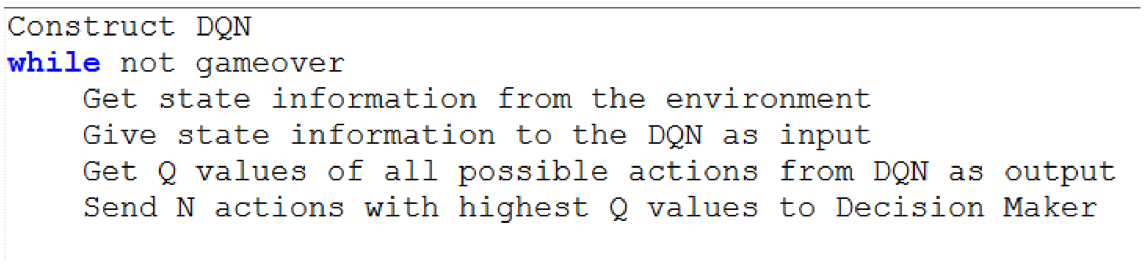


```
Construct DQN
while not gameover
    Get state information from the environment
    Give state information to the DQN as input
    Get Q values of all possible actions from DQN as output
    Select action with highest Q value
```

Figure 7: Pseudocode of SARL agent

As seen in the figure 7, state and reward information are retrieved from the environment. Then state information is parsed to obtain a suitable format for DQN. DQN gives Q values of all possible actions as output. Also, reward is used by DQN for training. Finally the agent selects the highest Q valued action as final action.

Pseudocode for MARL agent:



```
Construct DQN
while not gameover
    Get state information from the environment
    Give state information to the DQN as input
    Get Q values of all possible actions from DQN as output
    Send N actions with highest Q values to Decision Maker
```

Figure 8: Pseudocode of MARL agent

As seen in the figure 8, state and reward information are retrieved from the environment. Then state information is parsed to obtain a suitable format for DQN. DQN gives Q values of all possible actions as output. Also, reward is used by DQN for training. Then agent selects N actions that gives highest Q values and sends them to the decision maker.

Pseudocode for Decision Maker:

As seen in the figure 9, Decision Maker will get top N actions from each cooperative agent in the team and determines all possible actions of agents in opponent team from the state information retrieved from the environment. It will evaluate all the combinations of actions from cooperative and competitive agents. Finally it determines cooperative agents action which maximizes the team reward.

```
while not gameover
    Get state information from the environment
    Get top N actions from Agent1 and Agent2
    Determine possible actions of Agent3 and Agent4
    for all possible combinations of the actions of four Agents
        Give input of combination to the evaluation function
    Retrieve actions of Agent1 and Agent2 from the combination with highest team reward
    Send these actions to Agent1 and Agent2
```

Figure 9: Pseudocode of Decision Maker

<u>Pseudocode and explanation of DQN:</u>

We decided to use Q Learning algorithm as a structure of the problem. Q Learning learns how good the action is at a particular state and it is represented as Q(s,a). In this algorithm, firstly, we build a table for Q(s,a) values called as Q-Table and we have a reward table which includes rewards of the actions according to current state. Agents action is determined from Q-Table. In other words, agent takes action with respect to highest Q value among actions of current state. Q values are updated according to the figure 10 below:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Bigg)}^{\text{learned value}}$$

Figure 10: Update formula of Q value[1]

Pseudo code of Q Learning is like in figure 11:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Figure 11: Pseudocode of Q-Learning[1]

For our environment, simple Q learning is not sufficient because state space is continuous. Therefore, we need a function approximator for Q value and DQN is a function approximator of Q values of state-action pairs. Briefly, DQN takes state information as an input and gives Q values of actions of the input state as output as seen in figure 12.

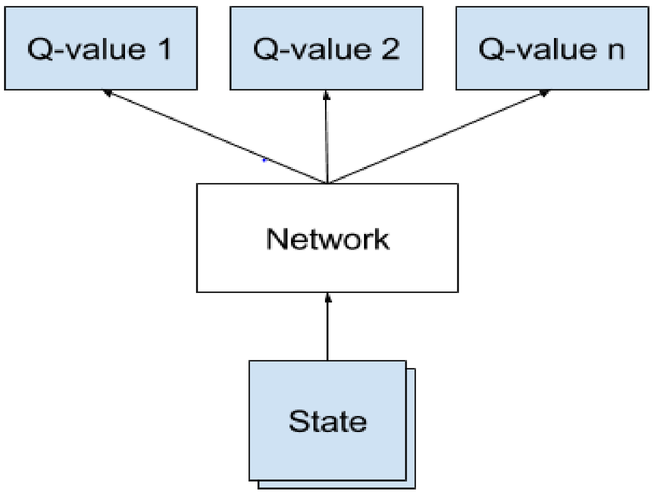


Figure 12: Structure of DQN[16]

The network is trained with current state(s), action(a), reward(r) and next state(s')(also called experience). If we train network after taking each action, the network will not converge because training data are correlated. Training data must be independent from each other for feasibility of learning. For this problem, there is a solution named “Experience Replay” . In this approach, after taking each action, experience is stored in a memory. After a few steps, mini batch is created randomly from the memory. Then, the network is trained with this mini batch. In figure 13 can be seen pseudocode for DQN with Experience Replay. Also, agent should take actions sometimes randomly, otherwise agent can not explore a new path.

```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
      end for
  end for
```

Figure 13: Pseudocode for Deep Q Learning[14]

However, we have one more problem. While we train the DQN, first step is feed forward step, then we calculate gradient of loss function and update the weights with weight - alpha * gradient as seen in figure 14. The problem is, target and prediction are dependent on networks parameters(Q^i). In other words, when DQNs parameters(weights) are updated, not only predicted Q Value is changed also target Q Value is changed.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right].$$

Figure 14: Gradient of loss function of DQN[14]

The solution for this problem is using 2 DQNs for target and prediction separately, which is called Double Deep Q Network(DDQN). In this approach, we use target DQN for retrieving target Q values of actions, prediction DQN for updating parameters in training. After a few iterations, we must synchronize target DQN parameters with prediction DQN parameters.

With both experience replay and DDQN, we have more stable input and output to train the network and they behave more like supervised training.

3.3. Comparison metrics

We will compare MARL and SARL approach in training and testing phases.
In training phase we will compare the learning speeds. In order to compare, we will determine a threshold value for learning. This threshold shows whether learning happened or not. We will compare which one reaches to that threshold in minimum number of episodes. We will assume that the one that reaches in minimum number of episodes, learns faster. However, this comparison only shows the speed of learning and not the quality of learning. Therefore in testing phase we will compare the quality of learning by making 100 matches between them. The team with highest number of wins would have higher quality of learning.

3.4. Data sets or benchmarks

We will not use any data sets because we will use RL which does not require any data sets.

4. System Architecture

Our projects architecture is shown in figure 15:

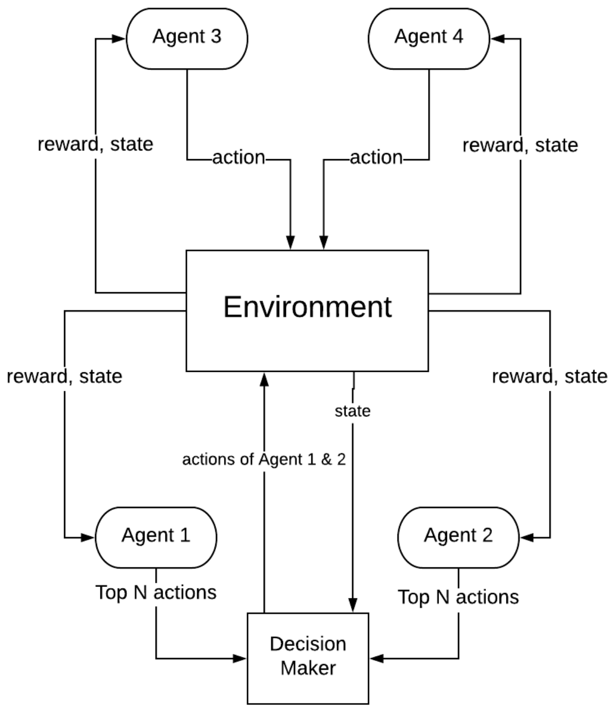
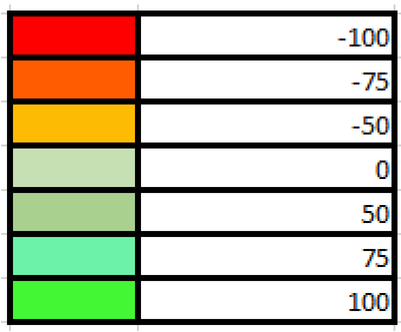


Figure 15: System architecture of our project

Environment:

The environment is responsible for getting actions from the agents and returning the reward to the agents according to these actions. State information is composed of position of all agents and the ball. Therefore the environment is fully observable. Also the environment returns different reward to each agent. We used a heuristic function for rewarding. This heuristic function takes agents' and balls position and the agent which is closest to the ball into consideration. Agent's team that is closest to ball, gets rewarded according to the table, in the colored area in the figure. To explain more clearly, when a team has the control of the ball, its reward increases as gets closer to opponents field. If a team does not have the control of the ball but the opponent has the control of the ball, rewarding will be according to two factors. First and most important factor is the distance of the agents to the ball. If the distance gets smaller, reward becomes larger. The second factor is positions of the agents. As they get closer to their field, reward increases. If both teams does not control the ball, distance of the agents to the ball will be rewarded. Also there will be a positive reward of controlling the ball.



🟥	-100
🟧	-75
🟨	-50
🟩	0
🟩	50
🟩	75
🟩	100

Figure 16: Reward table

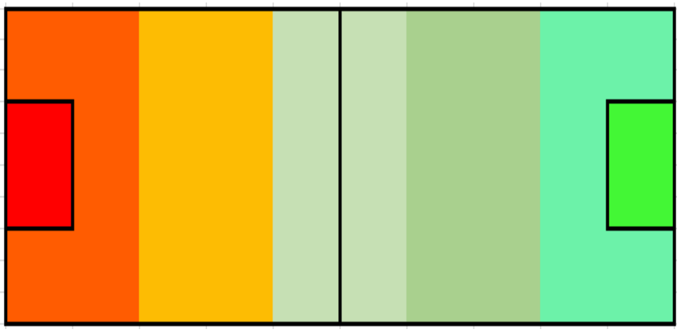


Figure 17: Color representation of rewards in soccer

As seen in the figure 17 and 16, we divided the field into seven areas. At attacking, the highest reward is gained when the attacking teams agents are closest to the opponent teams penalty area. Therefore, the team that controls the ball tends to attack. At defending, the values in the table are reversed. Higher reward is gained when defending agents are closer to the their own penalty area. However, also the distance to the ball is important. Otherwise defending agents would not tend to grab the ball.

5. Experimental Study

We have only tested our DQN implementation on Lunar Lander environment from OpenAI Gym so far. We have managed to successfully make agent learn. In soccer environment, we have not yet configured it to work with our DQN, that is why we have not tested on it yet.

6. Tasks Accomplished

6.1. Current state of the project

We have completed DQN implementation part of the project. Also we have managed to make contact with the soccer simulator using UDP socket, but we have not yet integrated our algorithms with it. We are now working on the heuristic function of the environment.

6.2. Task Log

During this semester, we had meetings with our advisor on Mondays each week. In these meetings, we discussed about the progress and current state of the project. Also we got advises from our advisor when we encountered problems and got advises about the problems that we may encounter in the future.

6.3. Task Plan with Milestones

Our first task is to configure the soccer simulator to work with our algorithms. Second task is to implement heuristic function of the environment. Third one is to integrate SARL and MARL approaches with the soccer simulator. The final task is to compare these two approaches according to the comparison metrics which we explained in the comparison metrics part.

7. References

[1] Sutton, R.S. & Barto, A.G., Reinforcement Learning: An Introduction, 445, 2017.

[2] Watkins, C.J.C.H. & Dayan, P. Mach Learn (1992) 8: 279

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou,

H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[4] Tan, Ming. (1993). Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents.. 330-337.

[5] Lauer, M., and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In Proceedings of the Seventeenth International Conference in Machine Learning.

[6] Nagayuki, Yasuo & Ishii, Shin & Doya, Kenji. (2000). Multi-Agent Reinforcement Learning: An Approach Based on the Other Agent's Internal Model.

[7] "Capture the Flag: the emergence of complex cooperative agents | DeepMind", *DeepMind*, 2018. [Online]. Available: https://deepmind.com/blog/capture-the-flag/. [Accessed: 20- Oct- 2018].

[8] Chincoli, Michele & Liotta, Antonio. (2018). Self-Learning Power Control in Wireless Sensor Networks. Sensors. 18. 375. 10.3390/s18020375.

[9] "Applied Deep Learning - Part 1: Artificial Neural Networks", *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6. [Accessed: 25- Oct- 2018].

[10] "Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond", *Medium*, 2018. [Online]. Available: https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df. [Accessed: 25- Oct- 2018].

[11] "Modern Game Theory and Multi-Agent Reinforcement Learning Systems", *Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/modern-game-theory-and-multi-agent-reinforcement-learning-systems-e8c936d6de42. [Accessed: 25- Oct- 2018].

[12] "Multi-agents Self-Driving Mario Kart with Tensorflow and CNNs", *Medium*, 2018. [Online]. Available: https://medium.com/@aymen.mouelhi/multi-agents-self-

driving-mario-kart-with-tensorflow-and-cnns-c0f2812b4c50. [Accessed: 25- Oct-2018].

[13]P. Stone, *Layered Learning in Multiagent System*, 1st ed. The MIT Press, 284, 2000.

[14]*Cs.toronto.edu*, 2019. [Online]. Available: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf. [Accessed: 01- Jan- 2019].

[15]*Csc2.ncsu.edu*, 2019. [Online]. Available: https://www.csc2.ncsu.edu/faculty/mpsingh/books/MAS/MAS-Springer/1.pdf. [Accessed: 02- Jan- 2019].

[16]"Guest Post (Part I): Demystifying Deep Reinforcement Learning - Intel AI", Intel AI, 2019. [Online]. Available: https://ai.intel.com/demystifying-deep-reinforcement-learning/. [Accessed: 02- Jan- 2019].