Half Field Offense – Final Q-agent

The goal was to alternate test how the agent evolves while training. So I would **train the agent for 500 games, test it for 50 games**, train it again, test it. Until reach **10k train games**. Although, to make sure that the results were not "luck", I would **repeat this process 30 times**.

Notes: The reward is sparse, +1 if it wins, -1 otherwise. The agent is playing single against a dumb goalkeeper.

1. Even Ronaldo is not perfect :(

While training the agent I noticed that the **training process was really unstable**, sometimes it would behave like Ronaldo, but other times like Éder. As the results bellow shows:

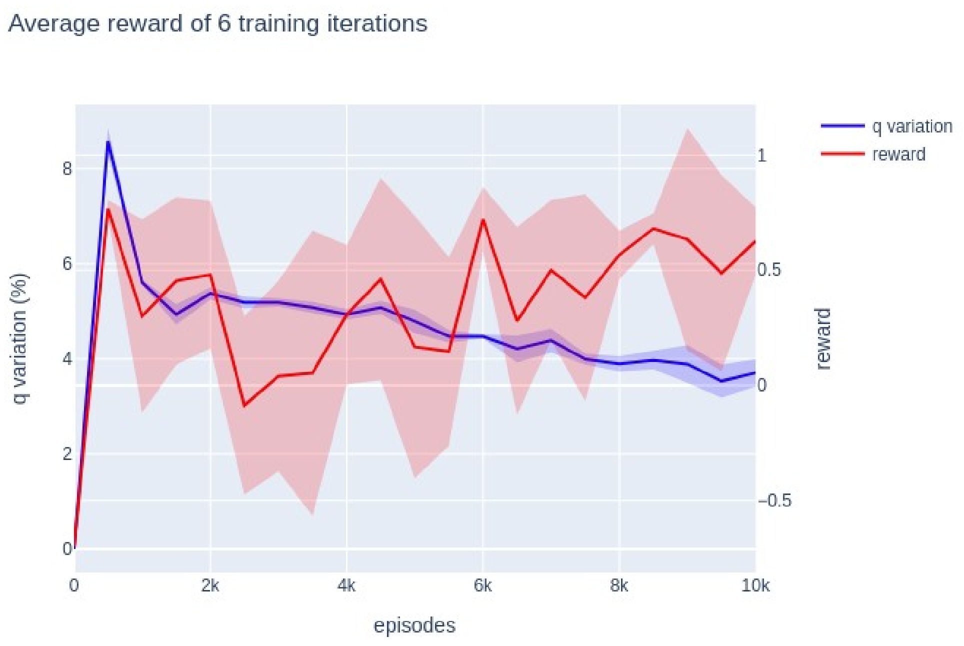


Fig 1 – The average results of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes.

To get more into the training process. The agent had a **learning rate of 0.1**. The **epsilon would start at 0.8 and end up at 0.3**. And the **q_table started as zero**. The epsilon changes as shown bellow:

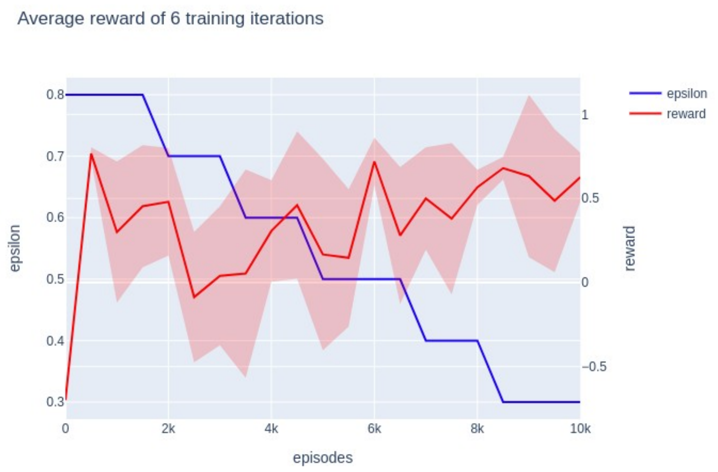


Fig 2 – The average reward of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes.

I felt that **something had to be wrong with my agent**, so I decided to **test everything** that could be going wrong:

1. **The environment is responding wrongly?** So I created some exceptions, in case the agent was not able to perform more than 5 actions per game. Although, this part shown that the agent was able to select actions, usually around 60 actions per game.

   ☺ No the environment is working okay!
2. **Am I losing the pointer to the agent** while alternating between train and test?

   ☺ No, the agent was the same and was correctly being updated (however it happened in the past)
3. **Is the Q learning function wrong?**

   ☺ No, for some iterations I checked all updates (sometimes I would calculate it by hand)

So, I could finally be in peace with myself, the code was working okay. Although, the agent was really unstable. So now the question **"Was I modeling the problem wrongly"?**

I trained the agent again without opponent, and as expected, he quickly started to archive great results. Although, sometimes would "unlearn", and learn again (still unstable). Also, removed most of the features, but results stayed almost the same.

To sum up, I assumed different possibles errors, which ended up led me to the conclusion that there isn't any kind of bug producing this results. Although some things contribute to the bad results:

- "**Sometimes actions do not produce the expected result"**. 95% of the times an action do what is expected, such as dribble the ball or kick to goal. Although sometimes, due to the dinamic aspect of the game, sometimes actions Fail, the agent could lose the ball while dribbling, or shoot to a random direction. These kind of random events make it harder for the qlearning to stabilize;

- **“Features could be better”**. The features used make sense, although, lots of states are rarely or never visited, or even do not give any special information to the agent. For example, the ball position would be useful if the agent was able to learn how to catch it, although it seems to be an hard task, partly because I do not give any incentive. Other features like the proximity of the opponent and goal angle, which are boolean features, do not give much information right now;
- **“Reward function is too simple”**. In the end of the game the agent receives 1 or -1, depending on scoring goal or not. Usually it takes 50 actions, until the agent receives the reward;

2. Try new stuff

Note: Now on, when I show the test results, on the left side will be the results of the initial approach, presented previously. And on the right side, the new results. The first results were only runned for 6 times, since each train would take around 20-30 minutes.

2.1. Initial Random Q Table

Until now the initial q table was just zeros. So using the initial conditions (1 opponent). Just changing the initial q table:

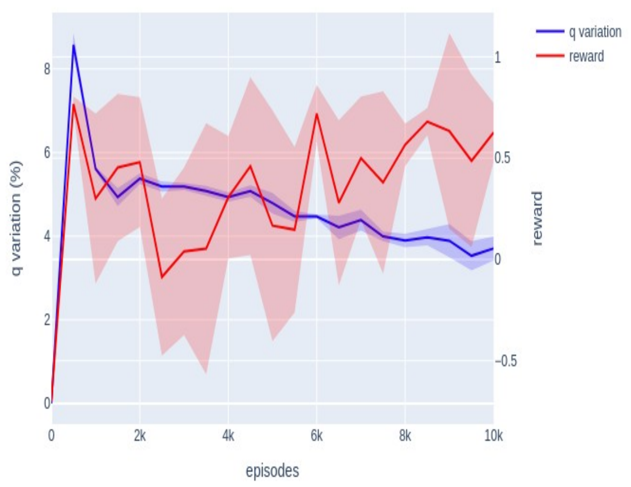
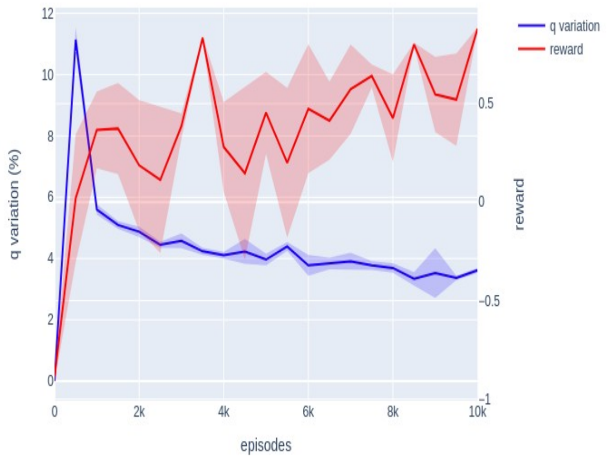
Old results (average of 6 training processes)	**New results** (average of 6 training processes)
	
• Initial Q table – zeros;	• Initial Q table – random;

Fig 3 – The average results of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes. *(Random Q table)*

As expected the initial variation of the q_table is higher, although it end ups becoming a bit more stable than the initial approach.

2.2. Epsilon reach 0.1

In the end of the training, the agent would keep exploring a lot. I assumed that it could be part of the problem. So now the epsilon goes from 0.8 to 0.1:

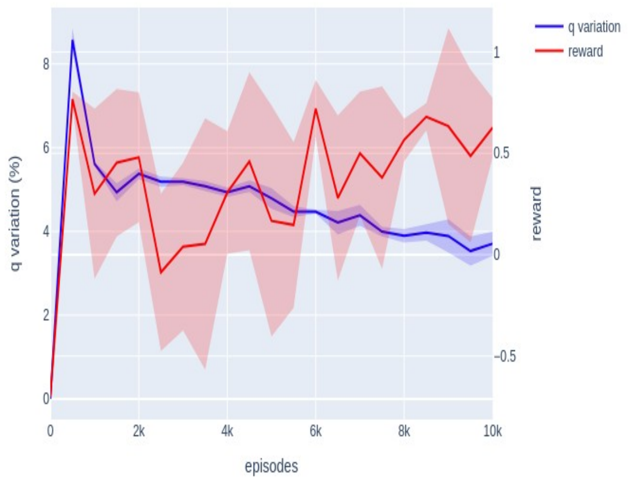
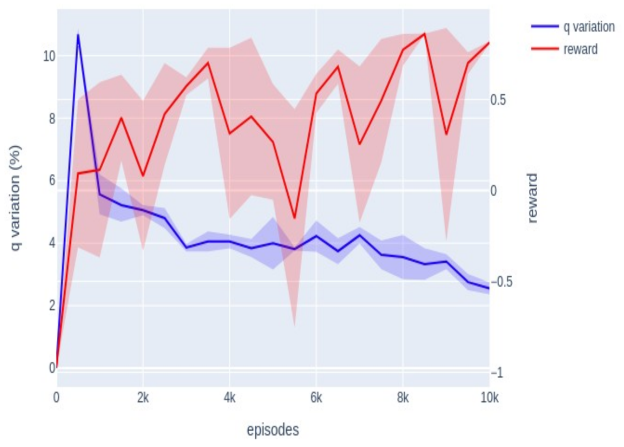
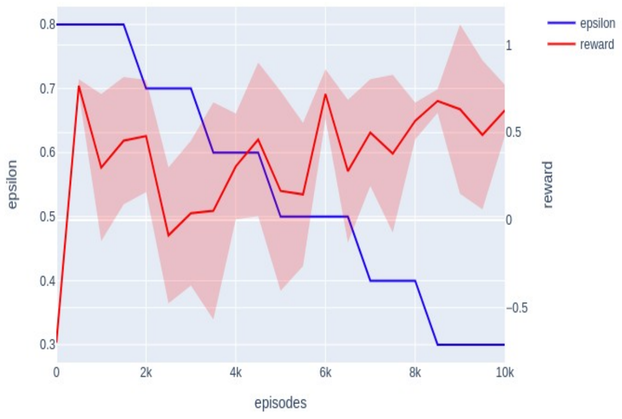
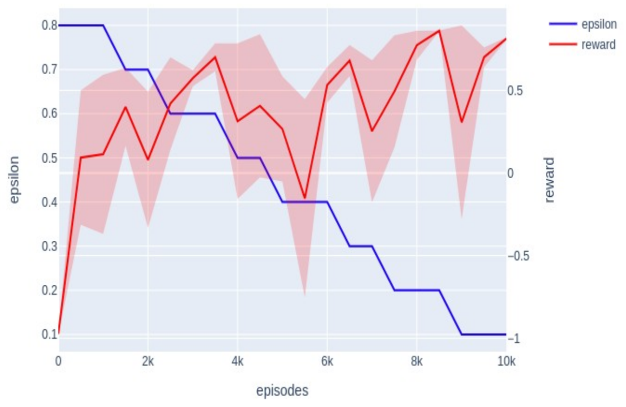
Old results (average of 6 training processes)	**New results** (average of 6 training processes)
	
	
• Initial Q table – zeros; • Epsilon – [0.8 – 0.3];	• Initial Q table – random; • Epsilon – [0.8 – 0.1];

Fig 4 – The average results of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes. *(Lower Epsilon)*

The results showed a lower variance, as expected.

2.3. Learning rate changes (old epsilons)

In the beginning of the training is important for the agent to learn fast. Althought in the end we want our agent to stabilize. So, in the first part of the training my agent has an learning rate of 0.1, but it the other half it has a learning rate of 0.05:

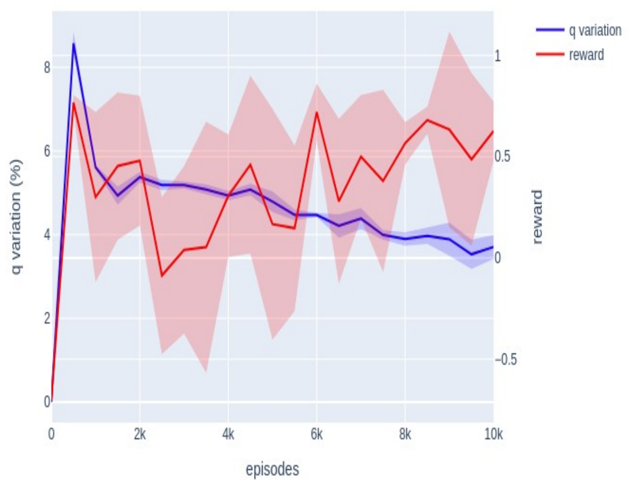
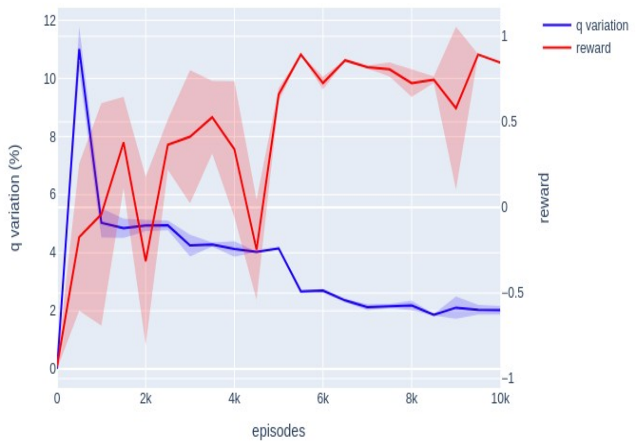
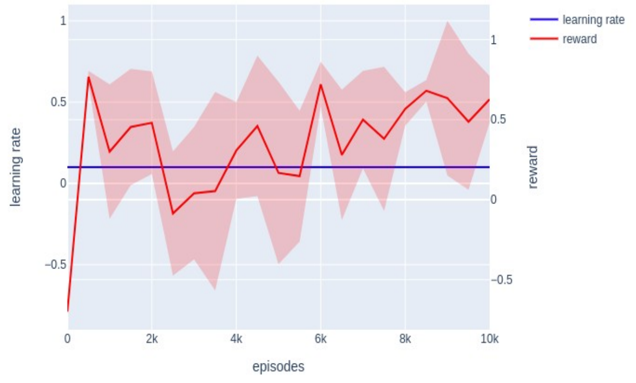
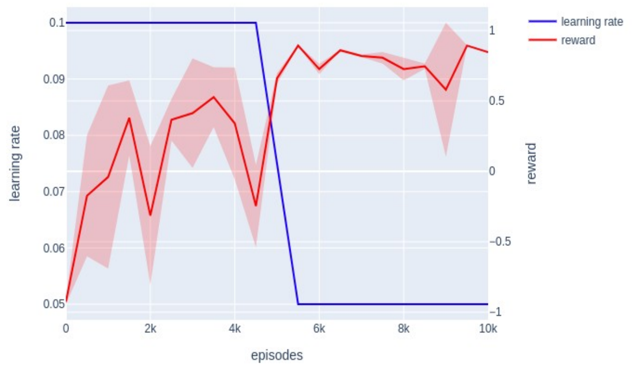
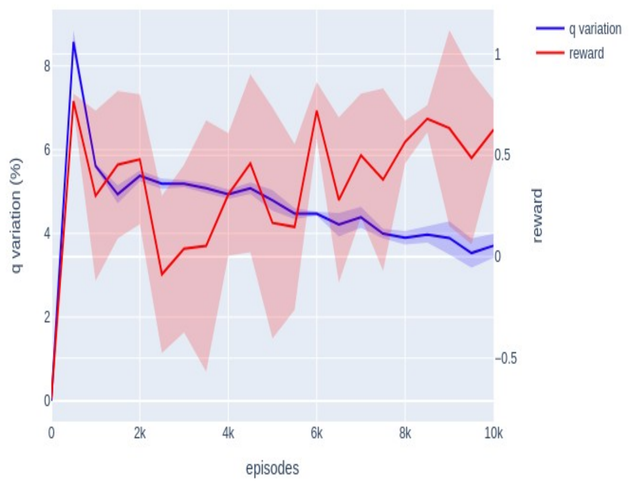
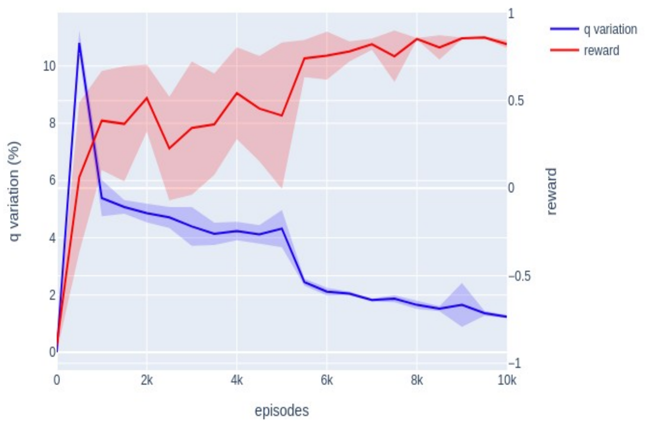
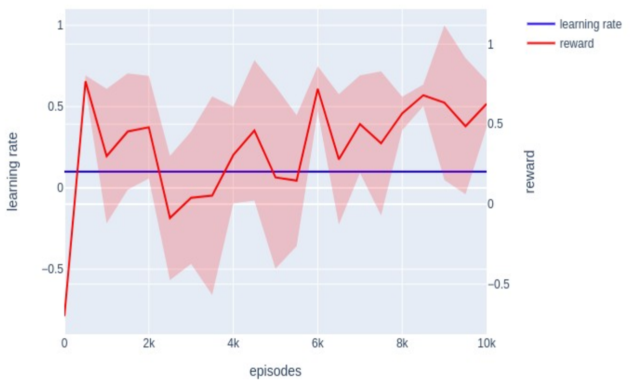
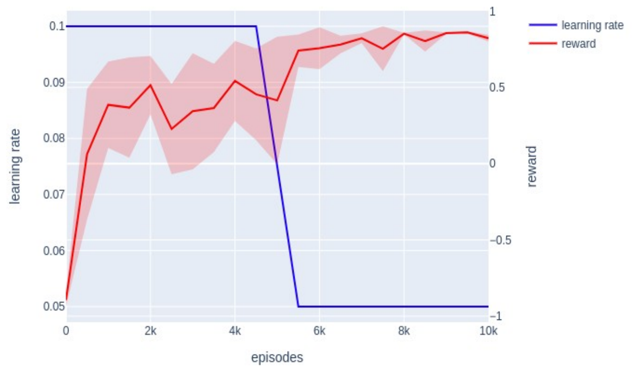
Old results (average of 6 training processes)	**New results** (average of 6 training processes)
	
	
• Initial Q table – zeros; • Epsilon – [0.8 – 0.3]; • Learning rate – 0.1;	• Initial Q table – random; • Epsilon – [0.8 – 0.3]; • Learning rate – [0.1 – 0.05];

Fig 5 – The average results of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes. *(Changing learning rate)*

The agent was able to reach better results and stabilize there.

2.4. Learning rate changes && new epsilon (Final agent)

This is the last version of my agent. So I trained it for 32 times, to make sure that the results are not bias (took me 12 hours):

Old results (average of 6 training processes)	**New results** (average of **32** training processes)
	
	

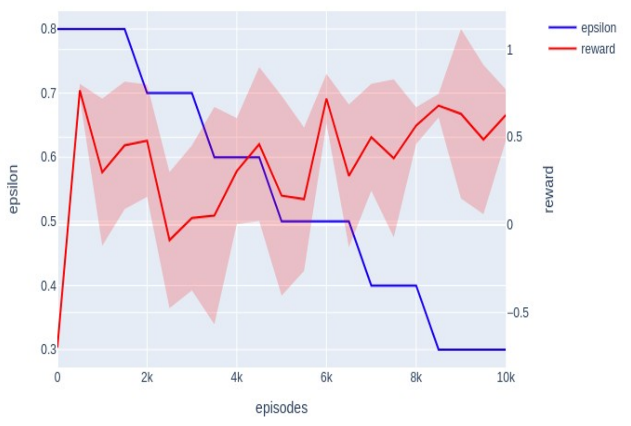
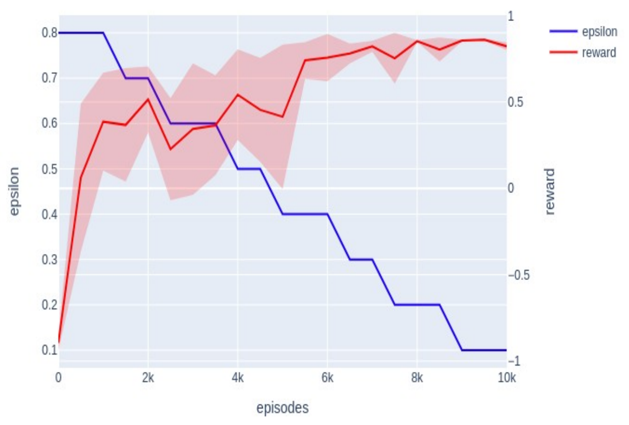
	
• Initial Q table – zeros; • Epsilon – [0.8 – 0.3]; • Learning rate – 0.1;	• Initial Q table – random; • Epsilon – [0.8 – 0.1]; • Learning rate – [0.1 – 0.05];

Fig 5 – The average results of 50 games after x episodes of training. The shaded part is the variance. These were the average results of 6 training processes. *(Changing learning rate)*

With all the new changes, the training process ended up becoming more stable, and allowed the agent to reach better results.

3.Future Work

- I feel tempted to go for an “hierarchical approach” (as João suggested). The agent learn to do smaller tasks, for example catch ball, score goal…
  Challenges: For now it make sense, however in the future, while playing with other agents, sometimes “go for the ball” is not best approach, since our teammate could be “counting on us” to go near the goal and wait for the ball; Or even in order to learn to score, I would have to introduce more shoot actions, with different angles;
- The agent only learns and decides by himself when he has the ball. When the agent do not has the ball, he would do fixed actions;

Game / Environment Setup

1. Game parameters:

- Number of **teammates: 0;**
- Number of **opponents: 1;** (Dumb Goalie)
- Number of **episodes: 10k train;**

2. Q-Learning Agent

2.1. Q learning parameters:

- Learning rate: [0.10, 0.05];
- Epsilon values: [0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1];
- Discount factor: 0.9;
- **Q learning table dim:** 120 environment states * **9** number of actions;

2.2. State Features

1. **Position** – int [0, 6], subdivided the field in 6 regions, each integer signalizes one of these regions;
2. **Should Shoot** – int (0,1). Agent's goal opening angle > 20%;
3. **Opponent is close** – int (0,1). Opponent is close;
4. **Ball Position –** int [0, 4]. Five states {0: "Player Has Ball", 1: "Up", 2: "Right", 3: "Down", 4: "Left"}

2.3. Actions

1. **Kick to Goal –** Shoot to the goal. This command only works when the agent has the ball.
2. **Short Dribble/Move Up –** The agents goes up 10x. If it has the ball, it dribbles the ball up, otherwise just moves up.
3. **Short Dribble/Move Down** – The agents goes up 10x. If it has the ball, it dribbles the ball up, otherwise just moves up.

4. **Short Dribble/Move Right –** The agents goes up 10x. If it has the ball, it dribbles the ball up, otherwise just moves up.
5. **Short Dribble/Move Left –** The agents goes up 10x. If it has the ball, it dribbles the ball up, otherwise just moves up.
6. **Long Dribble/Move Up –** The agents goes up 20x. If it has the ball, it dribbles the ball up, otherwise just moves up.
7. **Long Dribble/Move Down –** The agents goes up 20x. If it has the ball, it dribbles the ball up, otherwise just moves up.
8. **Long Dribble/Move Right –** The agents goes up 20x. If it has the ball, it dribbles the ball up, otherwise just moves up.
9. **Long Dribble/Move Left –** The agents goes up 20x. If it has the ball, it dribbles the ball up, otherwise just moves up.

2.4. Rewards

- - 1 – the game ends without scoring goal;
- + 1 – score goal;

3. Goalkeeper Agent

The agent presents 2 simple behavior:

```
If ball is on the upper side of the field:
    the goalkeeper moves to the goal top corner;
Else:
    the goalkeeper moves to the goal bottom corner;
```