19th September 2024

MIGe *Departement*
University *of Studies* ***of Trieste***

Deep Reinforcement Learning methods for Google Research Football RL Environment

Student:
Gabriel Masella [SM3800048]

Exam project for:
Deep Learning

Table of Contents:

1.

RL Environment

Brief introduction of the environment used in this project, what characterizes it and how it is described

1.a. Google Research Football Environment

Google Research Football (GRF) is a *reinforcement learning environment*, created by the Brain Team of Google, where agents can be trained to play football in an advanced, physics-based 3D simulator.[1]

In the Reinforcement Learning context, that's the case where there is an *high observability* but limited knowledge of the environment.
So, we will look for **model-free algorithms**.

1.b State, Action and Reward

In this context, the term *"state"* refers to the *comprehensive data set returned by the environment in response to the completion of specific actions*.

- The *floats representation* provides a compact encoding and consists of a *115-dimensional vector*, summarizing many aspects of the game, such as:
  - coordinates (x,y) and directions of players
  - ball position and direction
  - one hot encoding of ball ownership (none, left, right)
  - one hot encoding of game_mode [1]

This is the chosen representetion for this experiment, because of its efficiency

1.b State, Action and Reward

That's the **set of possible actions** that a single-agent can choose from in this RL Environment:

- Standard move actions (in 8 directions)
- Different ways to kick the ball (short and long passes, shooting, high passes).
- Players can sprint, try to intercept the ball with a slide tackle or dribble if they possess the ball.
- Moving and sprinting actions are sticky and continue until an explicit stop action is performed.[1]

Top	Bottom	Left	Right
Top-Left	Top-Right	Bottom-Left	Bottom-Right
Short Pass	High Pass	Long Pass	Shot
Do-Nothing	Sliding	Dribble	Stop-Dribble
Sprint	Stop-Moving	Stop-Sprint	—

1.b State, Action and Reward

For every action made by the agent at each timestep, there is a corresponding reward

- Two reward functions, implemented by GRF: **SCORING** and **CHECKPOINT**.
  - **SCORING**: *Each team obtains a +1 reward when scoring a goal, and a -1 reward when conceding one to the opposing team.*
  - **CHECKPOINT**: *The agent gets an extra reward if they move the ball close to the opponent's goal.* The agent's team gets an extra reward of +0.1 to +1 when they have the ball in each checkpoint region. Checkpoint rewards are given once per episode.[1]

2.

Solution Proposed

Introduction to Deep Reinforcement Learning and its advantages and a description of the Deep Q-Network algortim used in this project

2.a Deep Reinforcement Learning

With *non-linear function as approximator* we obtain **Deep Reinforcement Learning** methods when we use deep neural networks to approximate any of the following components of reinforcement learning:

- **Value Function** $\hat{v}(s;\theta)$
- **Action-Value Function** $\hat{q}(s,a;\theta)$
- **Policy Function** $\pi(s|a;\theta)$
- **Model** (Reward Function or State Transition Function)

where θ are the **network's weights**.[4]

2.b Deep Q-Learning Algorithm

Q-Learning is an off-policy control method to find the optimal policy.

It learns **action-value function** $Q(s,a)$ and the **Bellman Equation** gives the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Although theoretically speaking, an agent based on Q learning can memorize all state-action pairs from a table, also known as a *Q table*.

But practically, it is computationally expensive when state and action space is enormous.

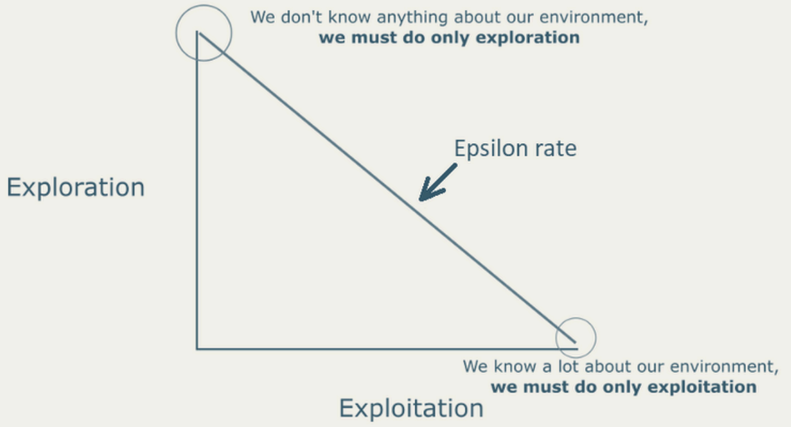
To deal with this problem, **one can use a function approximator in order to approximate Q-values**.[3]

2.b Deep Q-Learnin Algorithm

Deep Q-Learning comes under reinforcement learning and is an **improvement** and **extension** of **Q-Learning** by introducing a ***Deep Neural Network for action-value function estimation***, ***Replay Buffer*** and an ***Exploration Method***.

With ***experience replay*** the **agent's experiences are stored** at each time-step $e_t = (s_t, a_t, r_t, s_{t+1})$ into a *replay memory* and random samples of experience tuples are drawn out during the Q-learning updates.

After the execution of experience replay, the agent selects and executes an action according to an **ε-greedy policy**, designed to achieve a **balance between exploration and exploitation**.[2]



2.b Deep Q-Learnin Algorithm

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 ϵ 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

2.b Deep Q-Learnin Algorithm

This approach has **several advantages** over **standard online Q-learning**:

- each **step of experience is potentially used in many weight updates**, which allows for greater data efficiency.
- learning directly from consecutive samples is inefficient, due to the strong correlations between the samples, **randomizing the samples breaks these correlations and therefore reduces the variance of the updates.**
- By using experience replay **the behavior distribution is averaged over many of its previous states**, smoothing out learning and avoiding oscillations or divergence in the parameters. [2]

2.c Curriculum Learning

Training machine learning models in a meaningful order, from the easy samples to the hard ones, using curriculum learning can provide performance improvements over the standard training approach based on random data shuffling.
In this project 5 levels of training are created to train and test our agent:

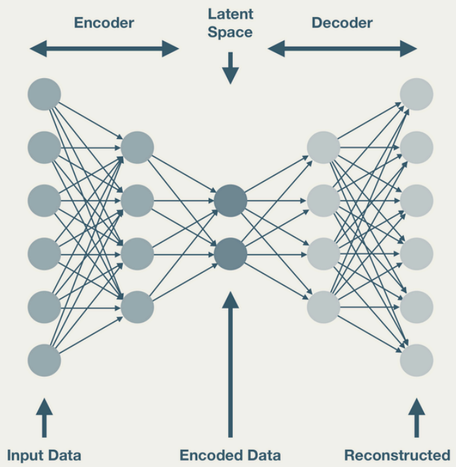
- **Level 0**: the Agent has to score in an **empty goal**.
- **Level 1**: the Agent has to score a goal, but there is the **goalkeeper**.
- **Level 2**: the Agent has to score a goal but there is a **defender** in front of him.
- **Level 3**: **2v1** where the agent is tagged by a defender and there is a free team-mate.
- **Level 4**: **3v2** situation with the agent tagged and another team-mate on the right tagged, the one on the left is free.

3.

Implementation and Training

Explanation of the designed architecture, hyper-parameters chosen and training procedure

3. Implementation and Training



In this project, the chosen architecture is a **deep autoencoder-like** for the implementation of the **Deep Q-Network**, incorporating:

- **Layer Normalization:** Helps with training stability and is suitable for varying batch sizes.
- **Dropout:** Prevents overfitting by randomly dropping neurons during training.
- **ReLU6 Activation Function:** Helps prevent activation values from becoming too large, which can improve training dynamics.

This network is designed to process the 115-dimensional input state, compress it, reconstruct it, and output raw Q-values for each possible action.

```
self.model = nn.Sequential(

            # Encoder Part
            nn.Linear(*input_dims,
fc1_dims),  nn.LayerNorm(fc1_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            nn.Linear(fc1_dims, fc2_dims),
            nn.LayerNorm(fc2_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            nn.Linear(fc2_dims, fc3_dims),
            nn.LayerNorm(fc3_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            # Decoder Part
            nn.Linear(fc3_dims, fc2_dims),
            nn.LayerNorm(fc2_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            nn.Linear(fc2_dims, fc1_dims),
            nn.LayerNorm(fc1_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            nn.Linear(fc1_dims,
*input_dims),nn.LayerNorm(*input_dims),
            nn.ReLU6(),
            nn.Dropout(p=dropout_rate),

            # Output Layer
            nn.Linear(*input_dims,
n_actions))
```

3. Implementation and Training

In addiction to the **Deep Q-Network there are a maximum size of 10,000 tuplets as replay memory**, and a **linear decay of the epsilon of 0.00001**.

As an **off-policy method**, during the inner loop of the algorithm, **Q-learning updates are applied with minibatch updates**, with a **batch size of 128**.

```
##########################################
# Hyperparameters
##########################################

LR = 1e-5                       # learning rate
GAMMA = 0.999                   # discount factor
BATCH_SIZE = 128                # batch size

EPSILON = 0.9                   # starting value of epsilon
EPSILON_DECAY = 1e-5            # decay rate of epsilon
EPSILON_MIN = 0.01              # minimum value of epsilon

MAX_SIZE = 5000                 # max size of the replay buffer

FC1_DIMS = 256                  # number of neurons in the first layer
FC2_DIMS = 128                  # number of neurons in the second layer
FC3_DIMS = 64                   # number of neurons in the third layer

TARGET_UPDATE_FREQ = 2500       # frequency to update target network
```

3. Implementation and Training

The **agent**, designated in the code as CR7*, **is trained through five training scenarios** of increasing difficulty.

The initial five levels have a maximum of **400 time steps**; however, this can be terminated prematurely if the objective is attained or the ball is removed from play.

Each scenario is comprised of **1,000 episodes** of training, followed by **100 episodes** of testing to assess the agent's learning outcomes.

*= nickname of the famous player Cristiano Ronaldo, known for his great achievements through hard training

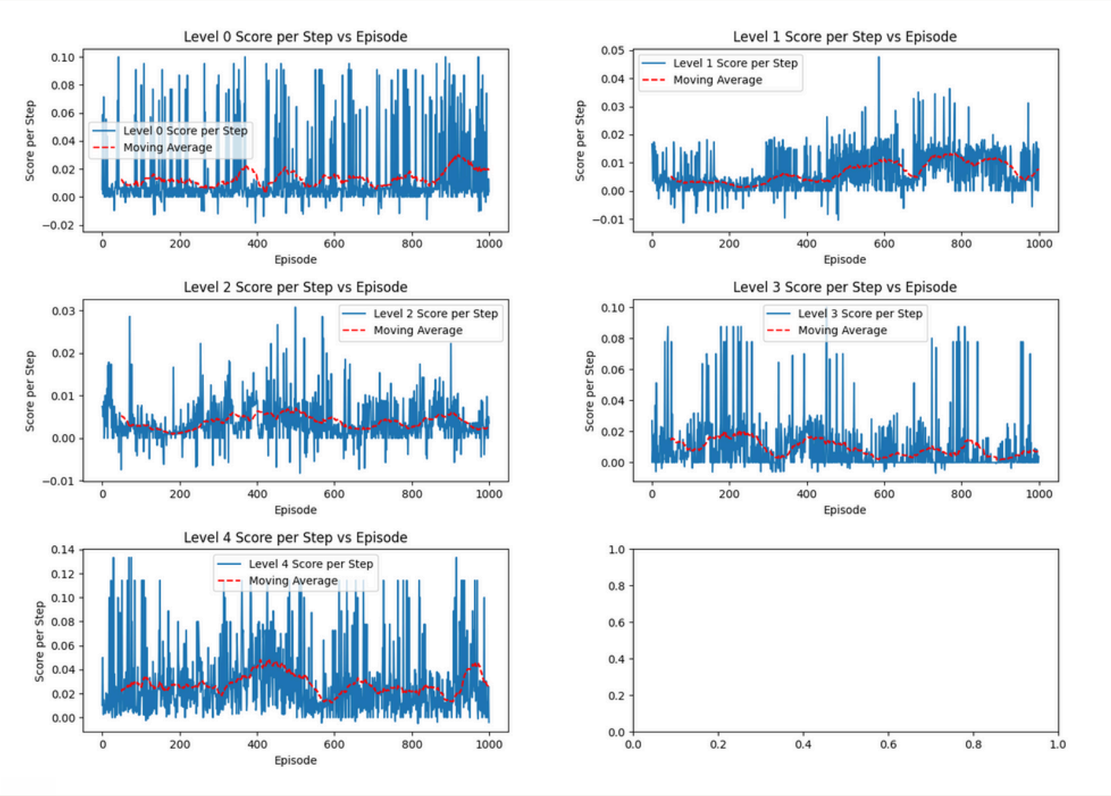
4.

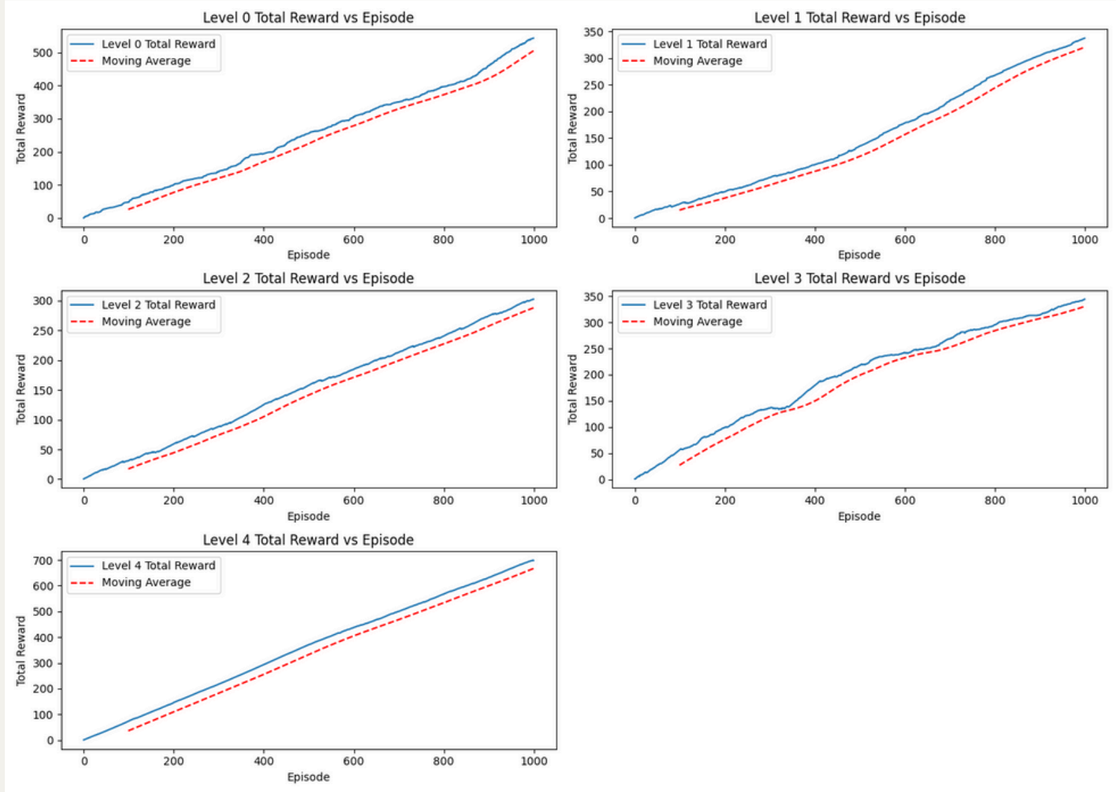
Test Results

How the CR7 agent performed? Does it learn an optimal policy to solve different scenarios?

4. Test Results

Prior to examining the knowledge acquired by agent CR7, it is instructive to consider the following graph, which illustrates the evolution of the **score ratio on # step** as a function of training level and **cumulative total reward** during the training.

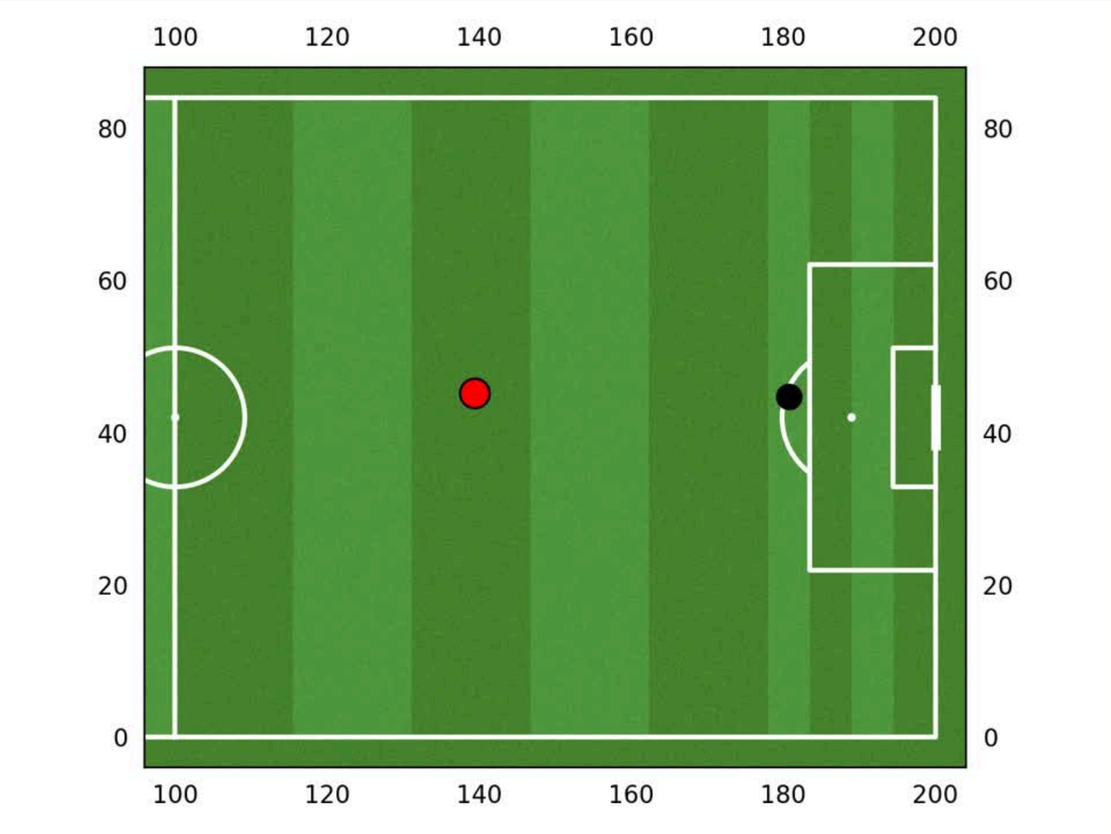


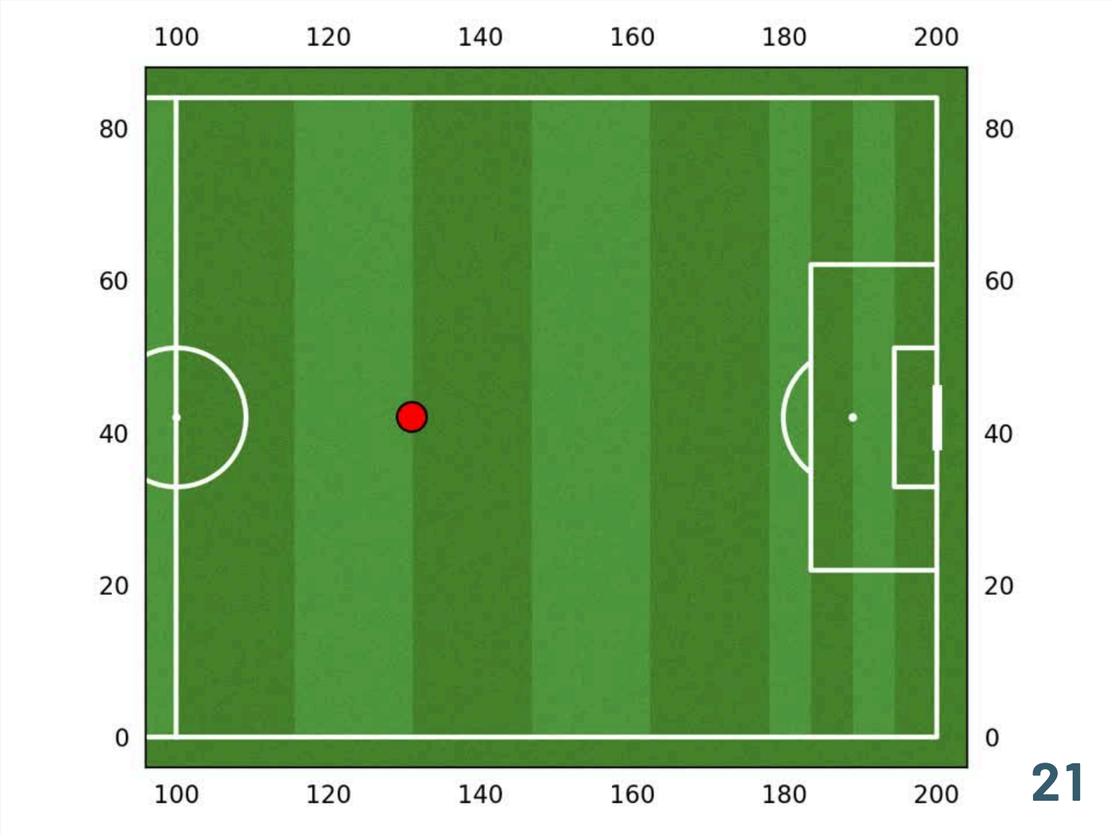


4. Test Results

The following visualization illustrates the policy that he acquired during the training. It is not the optimal policy, but he was able to identify a method for achieving his tasks.

That’s what the Agent learned during **level 0**:

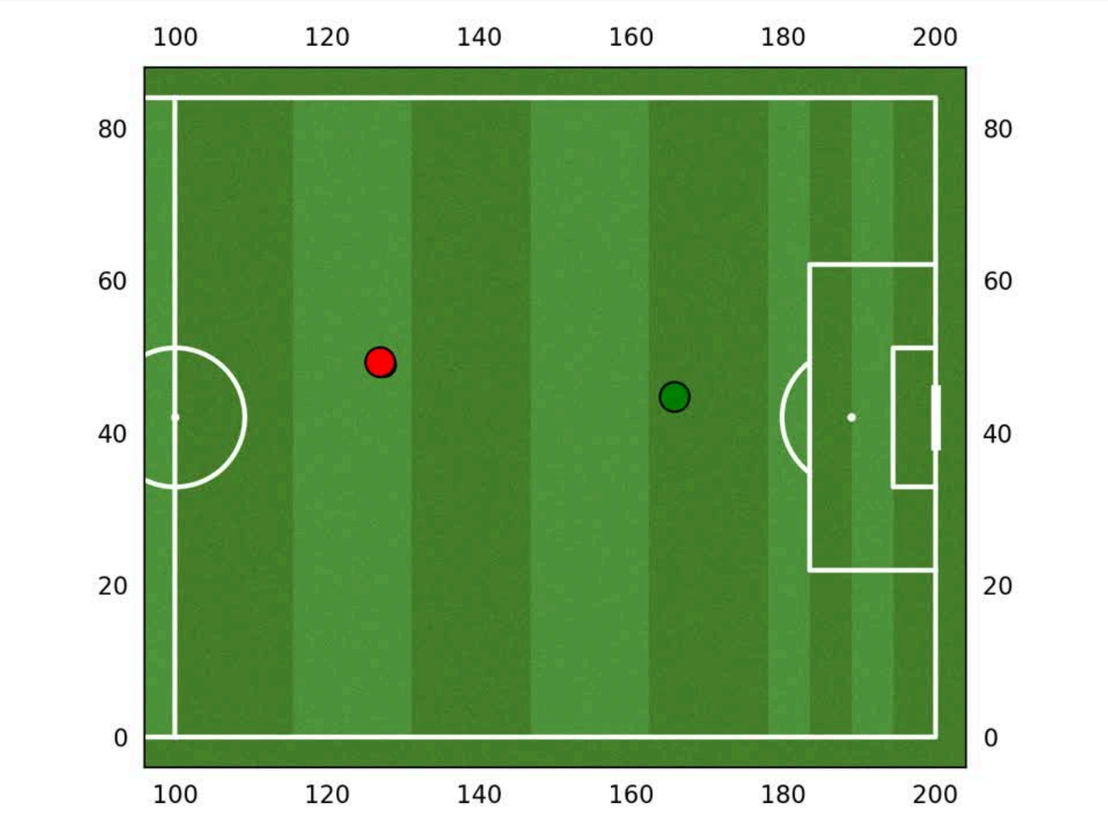


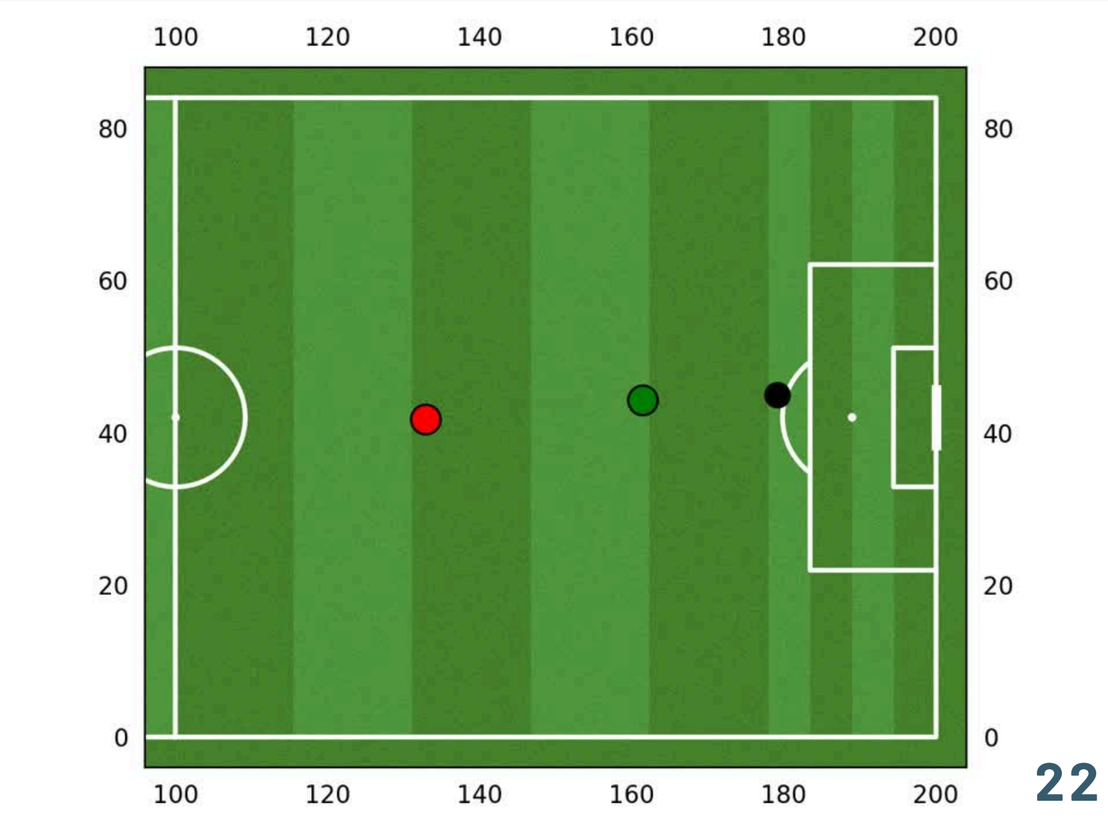


4. Test Results

The following visualization illustrates the policy that he acquired during the training. It is not the optimal policy, but he was able to identify a method for achieving his tasks.

That's what the Agent learned during **level 1**:

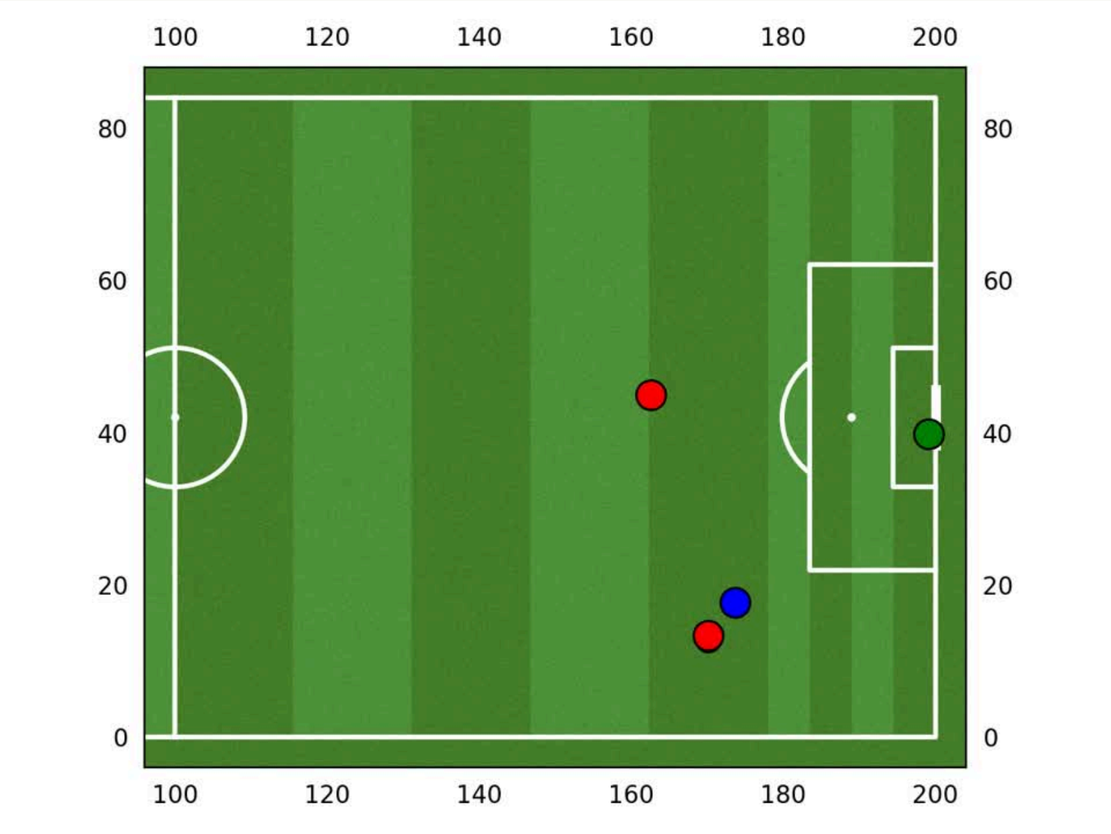


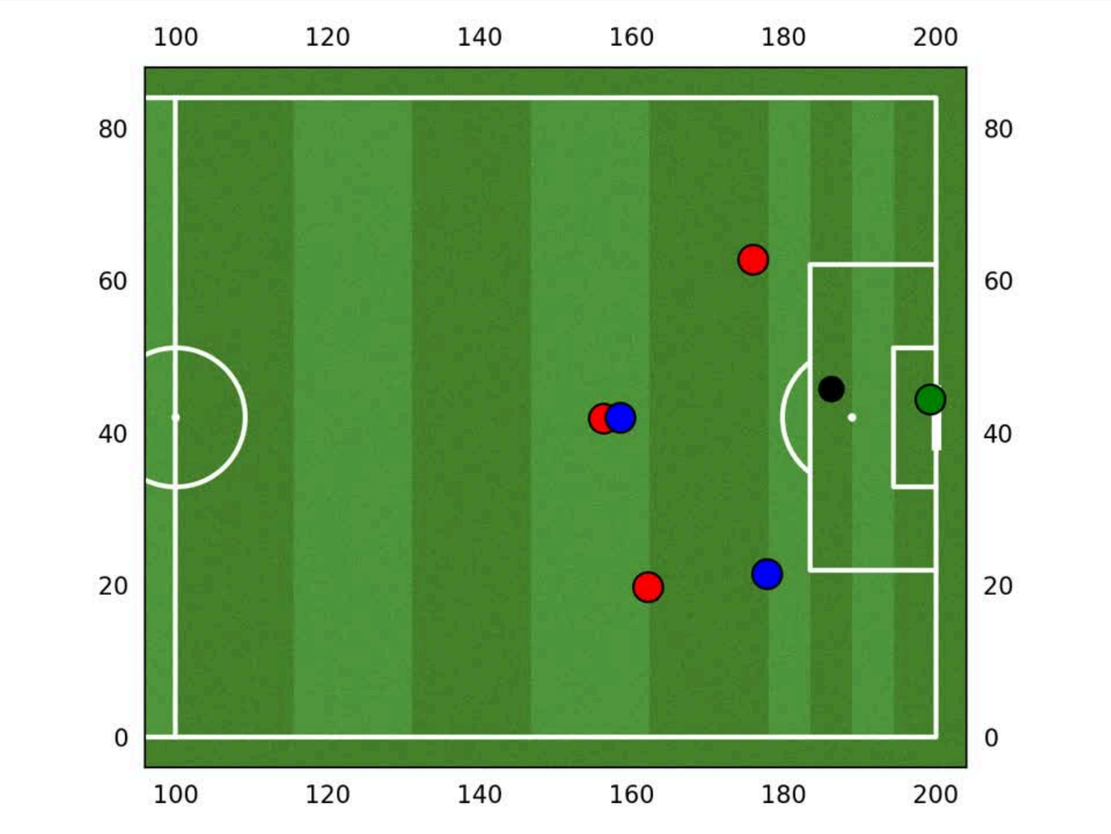


4. Test Results

The following visualization illustrates the policy that he acquired during the training. It is not the optimal policy, but he was able to identify a method for achieving his tasks.

That's what the Agent learned during **level 3 and 4**:





4. Test Results

The following table presents the test accuracy of the experiment conducted on agent CR7. For the initial five levels, we provide the number of times the agent scored goals as a percentage.

Training Scenario	Test Accuracy:
Level 0	**52%**
Level 1	**10%**
Level 2	**3%**
Level 3	**1%**
Level 4	**1%**

5.

Further Improvements

How the agent could do better, where the flaws are, and what to improve in the model.

5. Further Improvements

It is evident that the model is not operating at its optimal level for a number of reasons. However, there are **numerous potential approaches** that could be taken to enhance the performance of the agent:

- *A hyperparametrization-tuning could be performed.*
- *Probably for harder tasks, a higher number of training episodes is necessary* to let the agent understand better and adapt its policy.
- The use of different state representations, such as **pixels**, and the subsequent use of a **convolutional neural network** (as employed by Google in the original paper) could also be considered.
- Uniform sampling gives equal importance to all transitions in the replay memory. However, a more sophisticated sampling strategy might emphasize transitions from which we can learn the most, similar to **Prioritized Experience Replay**.

References

1. Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. (2020). **Google Research Football: A Novel Reinforcement Learning Environment.**
2. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. (2013). **Playing Atari with Deep Reinforcement Learning.**
3. Gupta, K., Dahiya, N., Dabas, M., and Pushparaj, P. (2022). **Playing football game using AI agents**. In *2022 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*(pp. 84-88).
4. Yuxi Li. (2018). **Deep Reinforcement Learning: An Overview.**