Problem Statement

Two RL agents, player A and player b (capital A denoting the player has the ball) compete in an 8*8 grid environment, having 5 discrete actions (up, down, left, right, and stationary) to score a goal at the coordinate defined in the environment.
The agents not only have to deal with maximizing their reward in the environment but also have to deal with another RL Agent (multi-agent problem in RL).

State-space :

We would like to add the following variables in the state space: Ball Owner, x coordinate of player, y coordinate of player, x coordinate of the opponent, y coordinate of the opponent, (distance of the player from goal post in the x-axis), (distance of the player from the goal in the y axis), score of our bot, score of opponent .

Reason - We want our agent to know who is the ball owner, and the distances from the goal post throughout the entire training process so that it can move to the goal post faster. Also the problem of boundaries, that is (if our agent reaches an edge and the action sampled from our neural network forces it to move out which is nullified by the environment, the action is wasted in this case. This problem is also addressed by the last two variables in the state space that calculates the distance of the agent from the goalposts). x and y coordinates are given as usual for the training process for offense and defense maneuvers.

Action space

We are using the actions which are possible in the environment as the action space for the problem, namely 0 - Stationary, 1 - move up, 2 - move right, 3 - move down, 4 - move left

Reward :

We are treating them as values that need to be tuned for training our bot. We will just explain the rewarding system in detail here.

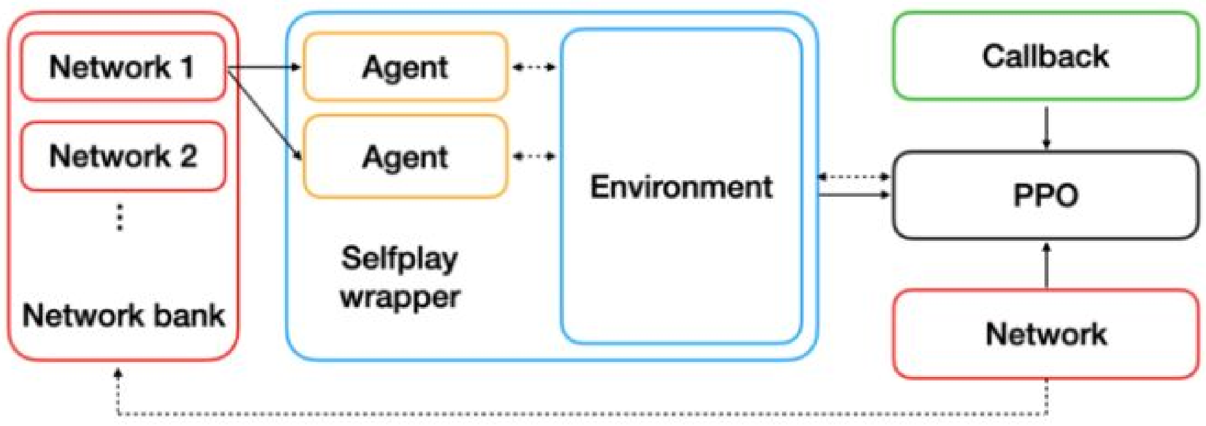
Reward - For Goal
Penalize - For losing the ball to the opponent, if our opponent is the ball owner, our bot has done self-goal, our opponent scoring goal.

Reasons for above penalizing and rewards :

Reward - We expect our agent to maximize its expected return from the environment which is done by scoring more goals. We would also like to reward the agent for having the ball as a means of telling the agent to not give the ball to the opponent
Penalize - Passing the ball to the opponent is a move that our agent cannot afford to do, and self-goal is discouraged too so that our agent is not encouraged to move backward and treat that as a goal

Model

For the model, we are using Proximal Policy Optimisation: Clipping version



The figure shows in block diagram on how we are going to use the ppo model to train the agent to play the game in the multiplayer mode
The blocks in the block diagram are explained below
The Network Bank. This store's previous versions of the agents to pull into the environment as opponents.
The Proximal Policy Optimisation (PPO) engine. PPO, developed by OpenAI in 2017, is a state-of-the-art RL algorithm. It updates the current version of the **network** being trained and fires a **callback** that saves the network to the bank if the current version has outperformed previous versions.**The Environment.** The base environment is any multiplayer game logic that you can dream up.
The Self-play Wrapper. This converts the multiplayer base environment into a 1-player environment that can be learnt by the PPO engine.

The Self-play Wrapper

The self-play wrapper performs two key functions:
Handling opponents - On each reset of the environment, it loads random previous versions of the network as the next opponents for the agent to play against. It takes the turns of the opponents, by sampling from their policy networks.
For example, the base opponent is loaded initially as Player 1 and Player 3 and the agent plays as Player 2. Player 1's turn is taken automatically by sampling from the policy network output before handing back to the agent.
Delayed Rewards
It delays the return of the reward to the PPO agent, until the other agents have taken their turn.

Suppose you are playing a 3-player card game such as whist. If you are the first player to play in the trick, then your action (playing a card) doesn't immediately create any reward. You need to wait to see what the other two players play in order to know whether you won the trick or not.

The self-play wrapper handles this delay, by following up the PPO agent action with all other required actions by opponents before returning any reward to the PPO agent being trained.

References

Self training

https://medium.com/applied-data-science/how-to-train-ai-agents-to-play-multiplayer-games-using-self-play-deep-reinforcement-learning-247d0b440717\

https://youtu.be/gqX8J38tESw?t=2999

PPO

https://arxiv.org/abs/1707.06347

https://openai.com/blog/openai-baselines-ppo/

https://www.youtube.com/watch?t=14m1s&v=gqX8J38tESw&feature=youtu.be