Autonomous Agents 2023 (INF 412)
Smart Connect 4: Developing an Intelligent Game-Playing Agent through Reinforcement Learning

Nikolaos Angelidis

May 2, 2024

1 Introduction

This project focuses on the game of Connect 4, a classic two-player connection game where the objective is to form a line of four consecutive discs of one's own color, vertically, horizontally, or diagonally. The primary goal is to develop an AI agent that can effectively play and excel at the game. This combines artificial intelligence through Reinforcement Learning. This agent balances tactical moves and strategic planning within a large but finite decision space of the game.

2 Reinforcement learning

2.1 How it works

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The fundamental components of any RL system are:

- Agent: the entity that performs actions and learns from the environment
- Environment: everything the agent interacts with and includes all the states
- States: represents the current situation or configuration of the environment
- Actions: the choices that the agent can make in any given state
- Rewards: positive or negative feedback, that helps the agent learn which actions are beneficial and which are not towards achieving a goal
- Policy: a strategy that the agent follows to decide which actions to take
- Value Function: estimates how good it is for the agent to be in a given state
- Learning and Optimization: the agent uses its experience in the environment to improve its policy

2.2 How RL is applied to Connect4 (Q-Learning)

Q-learning is a form of reinforcement learning, specifically a type of value-based method that enables an agent to learn the optimal action based on the current state from past experiences. The aforementioned fundamental components of an RL system in our case are:

- Agent: the player
- Environment: the connect4 board and the disks of each player
- States: represents the current configuration of the board
- Actions: placement of a disc in one of the columns of the game board
- Rewards: positive or negative reward can be given depending on a win, a loss, or a draw
- Policy: Q-values, which represent the expected future rewards for a state-action pair

- Value Function: the main goal in Q-learning is to find the optimal Q-values (more on the Q-values calculation later
- Learning: update of the Q-values based on its past experiences
- Optimization: adjusting parameters, so the policy to improves learning performance and speed up convergence towards optimal play

2.3 Implementing Q-Learning

The training of our agent is done through playing many games against against a random playing bot, updating Q-values, which is a combination of the next immediate reward and the sum of all the following ones, is also called the Q-value.

$$Q^{new}(S_t, A_t) = (1 - \alpha) \cdot Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \cdot maxQ(S', a))$$

Depending on the outcome and with the use of the ϵ-greedy policy to balance between exploration and exploitation. The value of is typically set to start high and decay exponentially over time, allowing for more exploration at the beginning of the training and more exploitation as the agent learns about the environment. Over time, the agent develops a solid strategy and can play a Connect4 game at a good level against various opponents.

The Q-learning strategy may seem practical for training our agent, although due to the number of these combinations that can reach millions, making it impractical to maintain a single table with all this information. In Connect 4, there are 4,531,985,219,092 (over 4 quadrillion) possible game states to account for. Storing such massive Q-table would exceed the storage capabilities of most modern computers. Additionally, even after extensive training periods, we cannot ensure that the agent has been exposed to every possible scenario within a game because of the vast amount of different possible states that the game can be in. Therefore, it is crucial for the agent to be able to handle new situations that were not encountered during its training. A way of dealing with this is a variant of the traditional Q-learning method, called Deep Q-learning.

3 Deep Q-Learning

In Deep Q-learning, a neural network is employed to approximate the Q-value for each action based on that state. The state of the board is input for the Deep Q-Network (DQN) and the Q-values for all possible actions are produced as outputs, then our agent picks the best action from the ones that the DQN returned. Deep Q-learning is essentially an extension of Q-learning that scales better to complex scenarios but requires careful management of neural network training techniques.

Later, an enhanced version of the standard Deep Q-Network (DQN) is employed, aimed at reducing the overestimation bias of action values that can occur in traditional DQN which helps in stabilizing the learning process and often leads to better policy evaluation and more reliable training outcomes.

3.1 Training the Q-Network

Firstly, the training of the Q-network starts by giving it random weights. As the agent continues to interact with the environment, it gathers experiences for every state and stores them at each time step, (s, a, r, s'), in a replay buffer, where, s is the current state, a is the action taken, r is the reward received, and s' is the new state after the action. A Replay Buffer, as mentioned before, stores the experiences that the agent observes, but with an extra functionality. In addition to allowing us to reuse this data later, we can also sample a batch of experiences from it randomly. That results in the experiences that build up a batch being uncorrelated, thereby enhancing learning stability and variety of our agent.

For each batch sample, the agent calculates the target Q-value. This is done using the reward for the experience plus the discounted maximum Q-value for the next state as predicted by a target network. The target network is a copy of the Q-network that is updated less frequently to stabilize training.

$$target = r + \gamma \cdot maxQ_{target}(s', a')$$

We want to update the Q-network by minimizing the loss between its predicted Q-values and the target Q-values. This is typically done using the mean squared error (MSE) of the loss, where:

$$loss = (target - Q(s, a))^2$$

3.2 Reward System and (Hyper)Parameter tuning

As mentioned above, every reinforcement learning environment should implement a reward system to motivate the agent for making moves that lead towards the desirable outcome and demotivate when they do not. More specifically, in the game of Connect4, the agent should be granted a reward when they do something good, mostly win or prevent the opponent from winning. On the other hand, when losing, they should be granted a negative reward. Moreover, the precise implementation of the hyper parameters of our model, such as the learning rate (α), discount factor (γ), and the exploration rate (ϵ), is of high importance for the correct training of the agent.

3.3 Application of DQN to Connect4 Agent

In the implementation of DQN for the training of the Connect4 Agent, the network input layer is the observation space for the Connect4 board, which means $columns \cdot rows$, 42. The output of the network are the estimated Q-values for each possible action for the current state. In the case of Connect4, these are the Q-values for placing a disc in each of the seven columns.

The training process of the model agent, starts by the agent playing games against a random bot, a player that picks a random available column. Later, the model agent is trained against a more experienced opponent, implements the strategies that were learned this far, and is also building new ones. During this process, the agent stores the state, action, reward, and next state (s, a, r, s') of each episode in the Replay Buffer. As a result, the agent learns the basic rules of the game and starts developing good habits, going for winning moves, and preventing the opponent from winning when possible. Subsequently, the agent samples a batch of experiences from the Replay Buffer to train the DQN. For each experience in the batch, the DQN calculates the loss between the predicted Q-value and the target Q-value. Later, the weights of the network are updated to minimize this loss, using back-propagation and an optimization algorithm.

Lastly, during the training process, the performance of the agent is measured, such as the win rate and the average reward against each opponent and also the balance between exploration and exploitation.

3.4 Double Deep Q-Network (DDQN)

In the standard DQN, the same network is used to estimate the Q-value used to select the best action and to evaluate that action. This dual role can lead to significant positive bias because the same estimates that are noisy and possibly with high variance are used to choose and evaluate an action. This issue is addressed by implementing a DDQN. More specifically, the networks used are:

- Training network: selects which action to take
- Target network: evaluates the action chosen by the training network

During the training phase, the training network proposes which action appears best and the Q-value of this selected action is evaluated by the target network. Thus, the target Q-value equation for the Double DQN implementation is as follows:

$$Target_{Q-value} = R_{t+1} + \gamma \cdot Q_{target}(s', argmax_{a'} Q_{training}(s', a'))$$

As a result, the overestimation of Q-values is reduced, leading to more stable and reliable agent learning.

4 Performance Evaluation

4.1 Experimental Setup

The environment in which the Connect4 agent is trained, is provided by Kaggle, a subsidiary of Google with an online community of data scientists and machine learning engineers. There, the agent will train against the 'random' agent for 20000 episodes. The large number of episodes against the 'random' agent is due to the accelerated computation time for the move that the random player will play. Later, the model will train against the 'negamax' agent of the Kaggle environment, which is a more experienced and competitive opponent, for 5000 games. The whole training process takes about 2.5 hours on the Google Colab online notebook. A crucial part of the training is rewarding the agent when blocking the opponent from making a Connect4, that way we encourage the agent during the course of the game and not just at the end depending on the outcome to get the reward of the actions taken.

4.2 Measurement Comments

The most direct way to measure the performance of our agent is the win percentage, indicating the percentage of games won versus games each type of opponent. During the training of model agent the win-rate should steadily increase, as a result of our agent learning and developing winning strategies. We expect the win-rate against the 'random' agent to be higher than the one against the 'negamax' agent of the Kaggle environment. Moreover, over time our agent should optimize their reward gaining policy, therefore we should also observe a steady increase at the average reward gaining over the last 100 episodes. Lastly, the plotting of the ϵ parameter is important to show the balance between, exploration and exploitation during our agents training with each type of opponent agent.

4.3 Training Plots

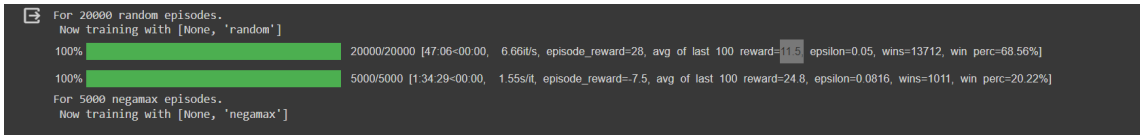


Figure 1: The final training time, wins, win-rate and average reward of 100 episodes

Figure 2: Win-rate for 100 matches vs the random agent, 81% showing that the agent learned

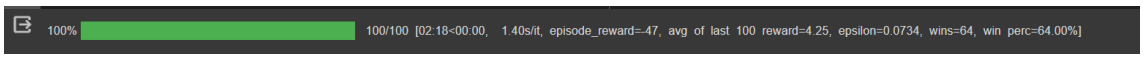
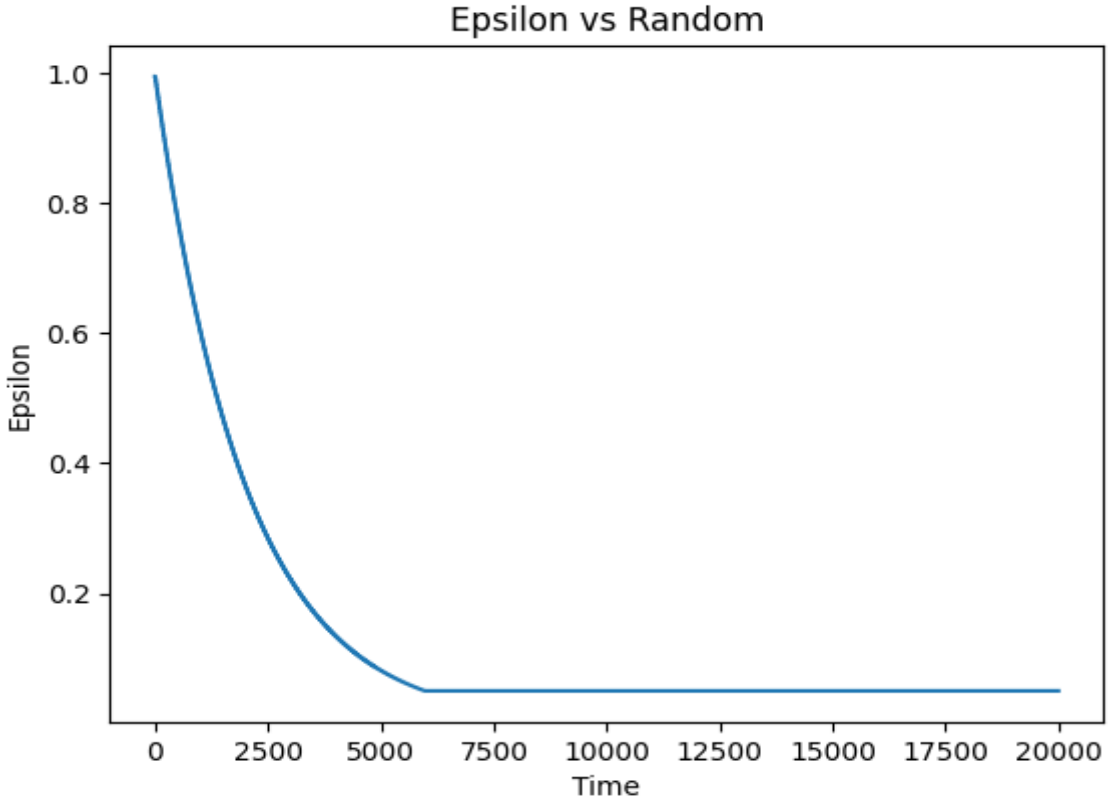


Figure 3: Win-rate for 100 matches vs the negamax agent, 64% showing that the agent learned

4.4 Measurement Plots



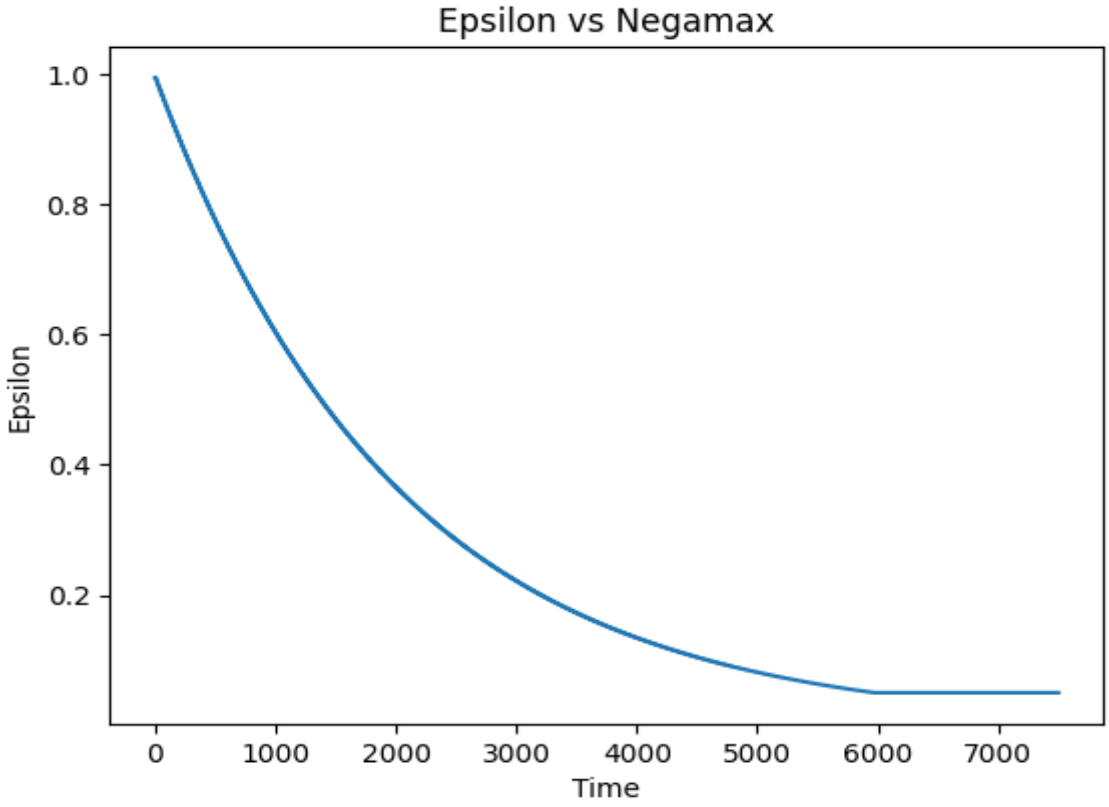


Figure 4: ϵ variable decay through training against the random and the negamax agent, indicating a balance between exploration and exploitation

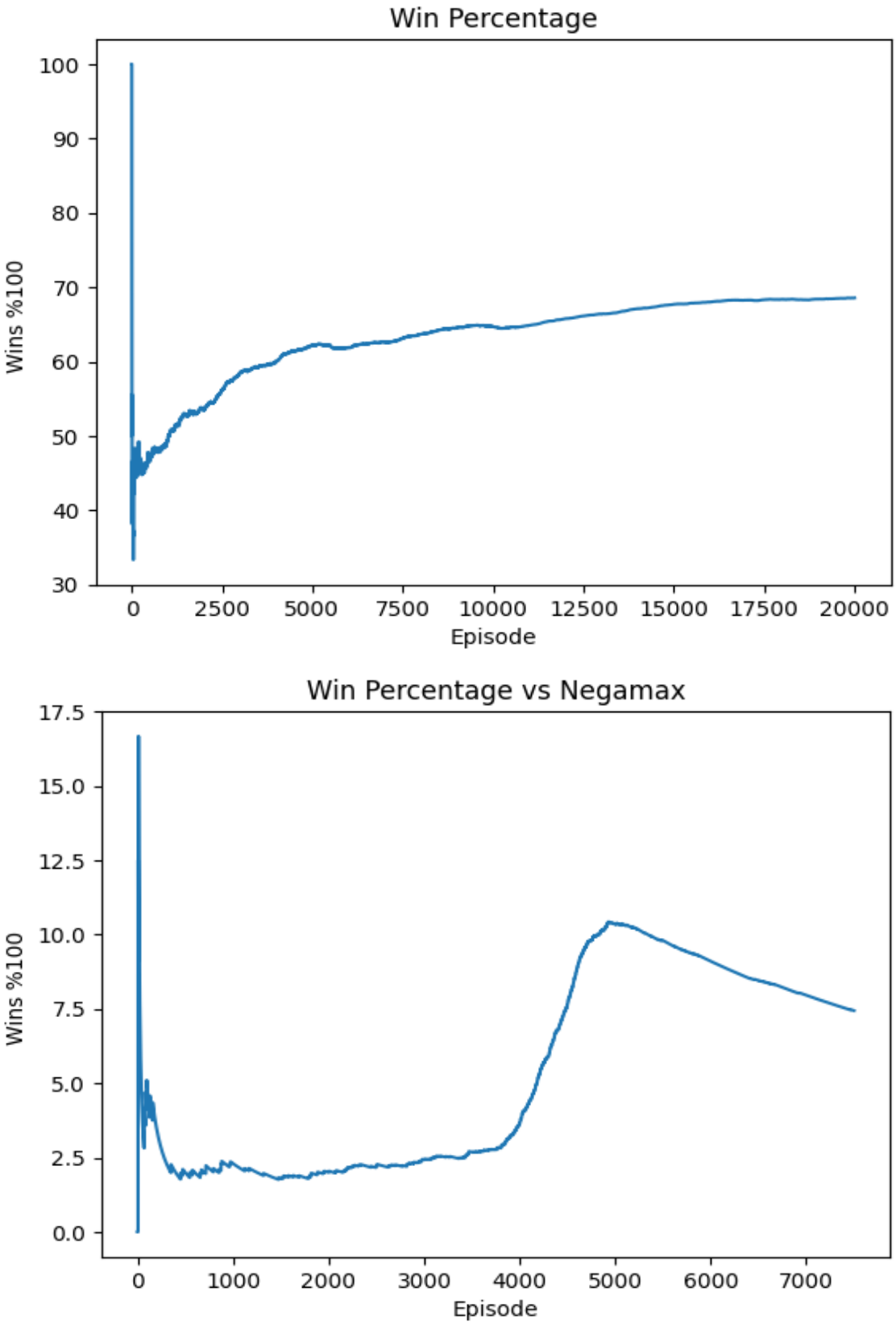


Figure 5: During the training process the win percentage is shown to be rising, meaning the agent is learning and is continuously improving

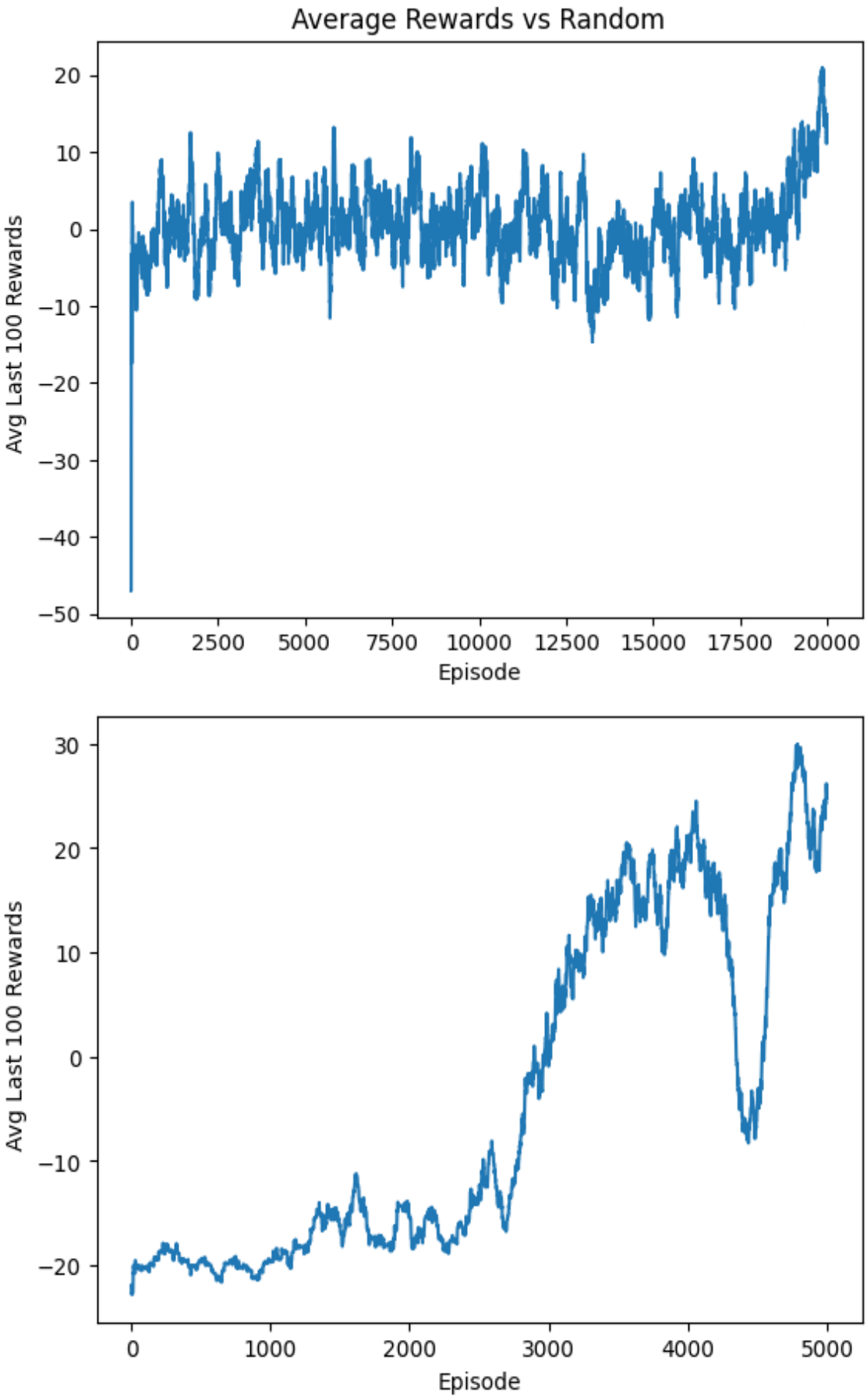


Figure 6: In addition to win-percentage being a solid indicator that the agent is learning, the measurement of the average reward for the last 100 episodes, rising of this measurement is also demonstrated

5 Graphical User Interface (GUI) for Connect4

A custom graphical user interface was designed, it serves as a practical tool allowing the user to play against another human, the random player and the trained agent model, demonstrating the capabilities of the agent, offering real-time game-play experience. Although, the

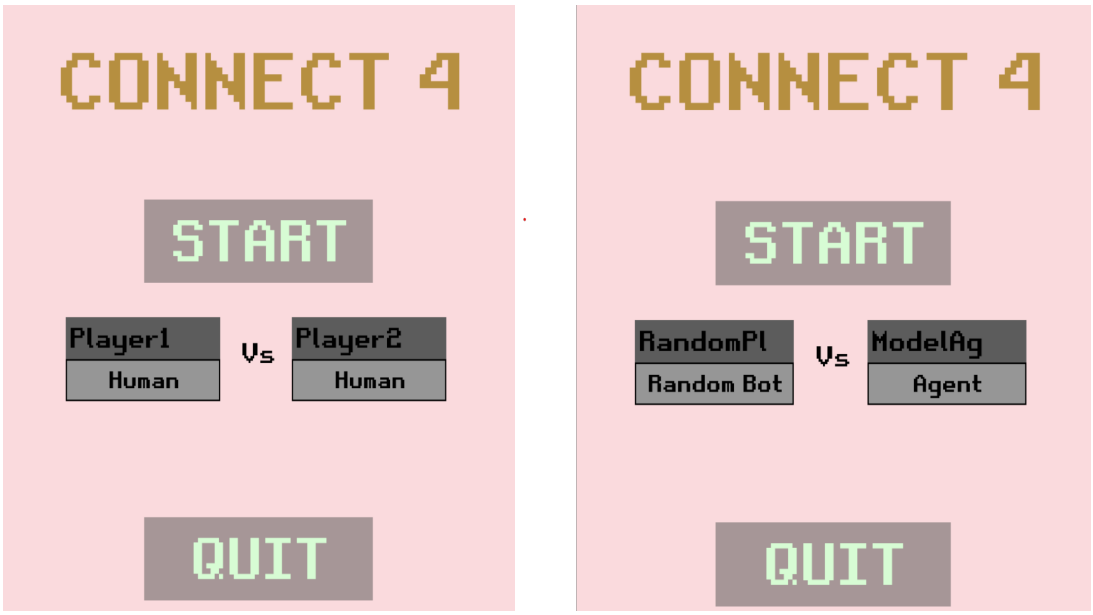


Figure 7: Setup Menu Screen with custom names and player types

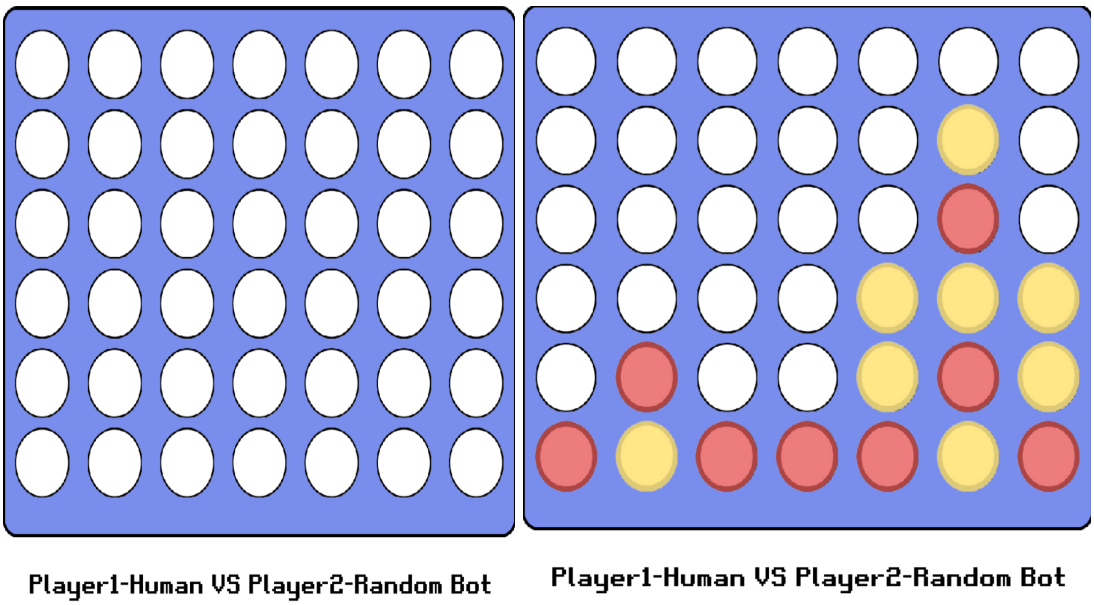


Figure 8: Game Screen that displays the name and the type of the players

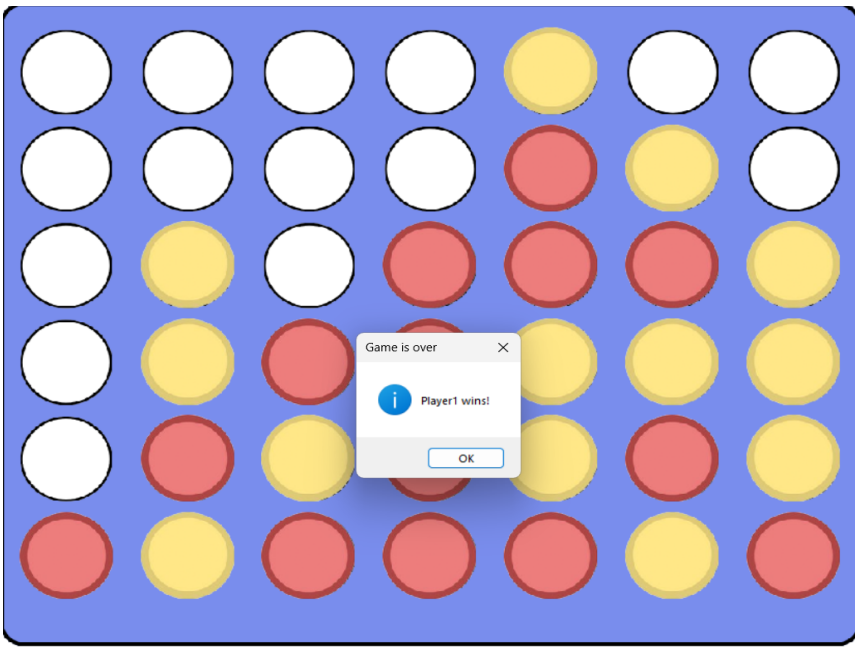


Figure 9: Ending Screen with the name of the player than won.

References

https://en.wikipedia.org/wiki/Q-learning