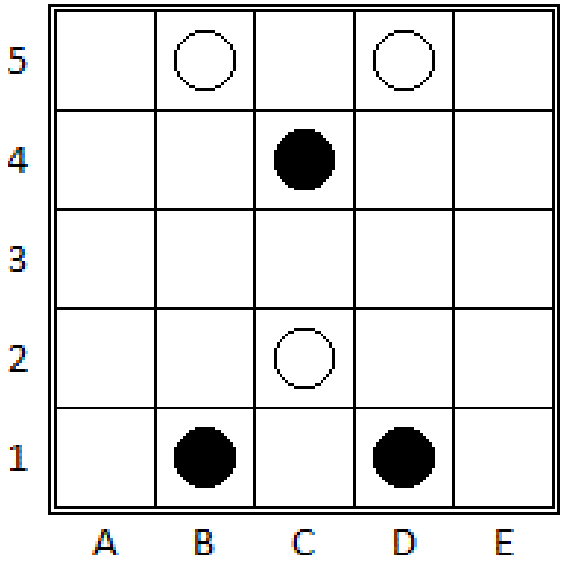
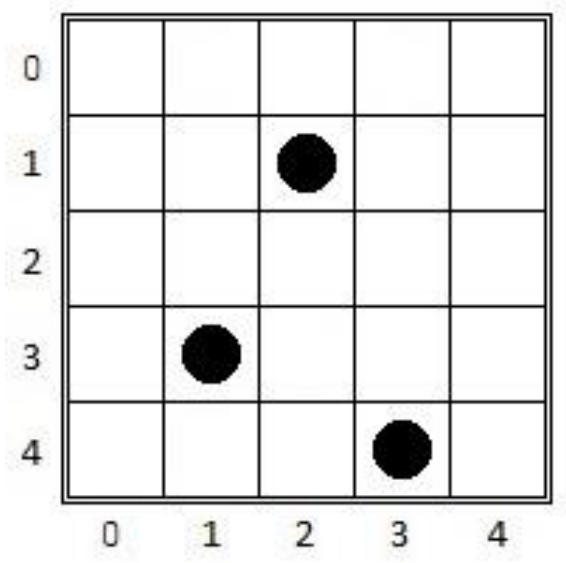




Simplified Neutreeko using Reinforced Learning

André Gomes - 201806224
Gonçalo Teixeira - 201806562
Luís Recharte - 201806743

Specification of Reinforced Learning Problem

Game Logic

- Create Starting Condition
- Possible moves
- Manage player turns
- Perform a move and update the board
- Verify end game and tie conditions

Environment

- Step – Apply the Agent action to the environment
- Reset – Reset the env
- Render – Print a representation of the env
- Close – Finish the episode
- Done – Check if episode is done

Agents

- **Random choice** Agent
- **Q-learning** Agent
- **SARSA Agent**
- Monte Carlo Agent (not implemented)

Related Work & References

Board games with OpenAI Gym:

- Abalone – https://github.com/towzeur/gym-abalone
- Go – https://github.com/aigagror/GymGo

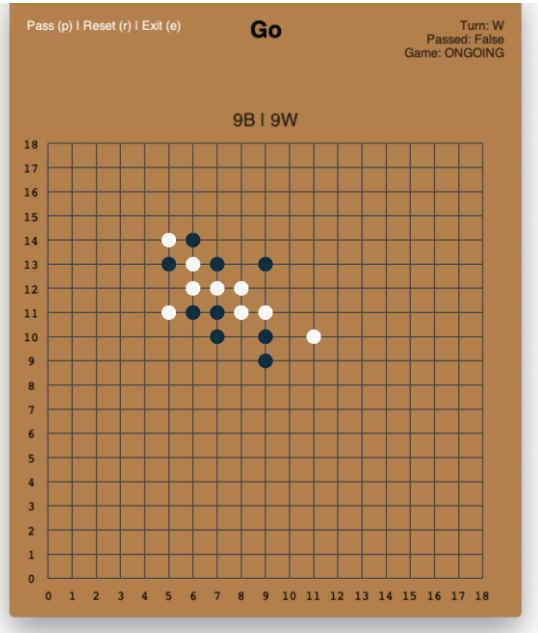
Spaces' definition – https://github.com/openai/gym/tree/master/gym/spaces

Create an environment – https://github.com/openai/gym/blob/master/docs/creating-environments.md

Table of environments – https://github.com/openai/gym/wiki/Table-of-environments

States, Observation and Action Spaces in Reinforcement Learning - https://medium.com/swlh/states-observation-and-action-spaces-in-reinforcement-learning-569a30a8d2a1

Q-Learning Agent - https://medium.com/swlh/introduction-to-q-learning-with-openai-gym-2d794da10f3d



Tools and Algorithms

Anaconda environment – Python 3.8

JetBrains IntelliJ

Gym, Numpy, Matplotlib

Q-Learning and SARSA

Work carried out

1. Started the structure of a OpenAI Gym project
2. Implemented the game logic
3. Implementation of the game loop and the environment. Random Agent to play the game
4. Define the rewards and their values. Implement the remaining agents
5. Transpose code to a Jupyter notebook

Neutreeko valid board positions

Sinalizar para seguimento.

JH

Jan Kristian Haugland <admin@neutreeko.net>
sáb, 22/05/2021 12:37
Para: André Gomes

Hi,

I am not entirely sure how I arrived at 3,450,515 (this would date back around 19 years). I have probably assumed that the Next player (N) does not already have three in a row, which gives them C(25, 3) - 48 = 2252 possible positions for their pieces, where C(n, k) denotes the binomial coefficient (n over k). The Previous player (P) should then have C(25 - 3, 3) = 1540 possible position for each of them, for a total of 2252 x 1540 = 3,468,080 positions. Most likely I have subtracted positions that are impossible to reach because there is no position from which the Previous player could have made a legal move and reached the current position. Two examples are

00000
0P0P0
00000
NN0P0
0N000

and

00000
0NPPN
00000
N00P0
00000

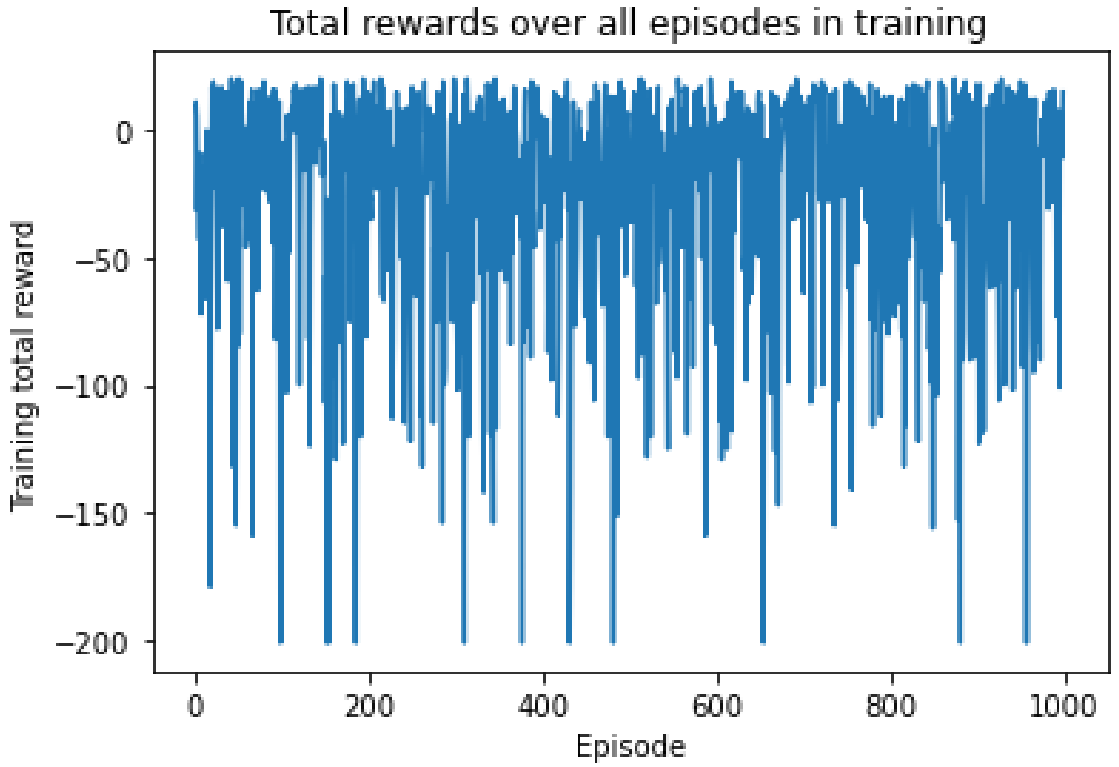
But I guess for this purpose, this is not extremely important, and we can use all the 3,468,080 positions.

I am not sure what would be a useful way to map each position to a unique number. Perhaps one could maintain a list of the C(25, 3) = 2300 ways to place three pieces of one colour, for a total of 2300^2 = 5,290,000 positions with both colours, and then sift out those that collide, if that is feasible. This is what my program does. By the way, on my web site

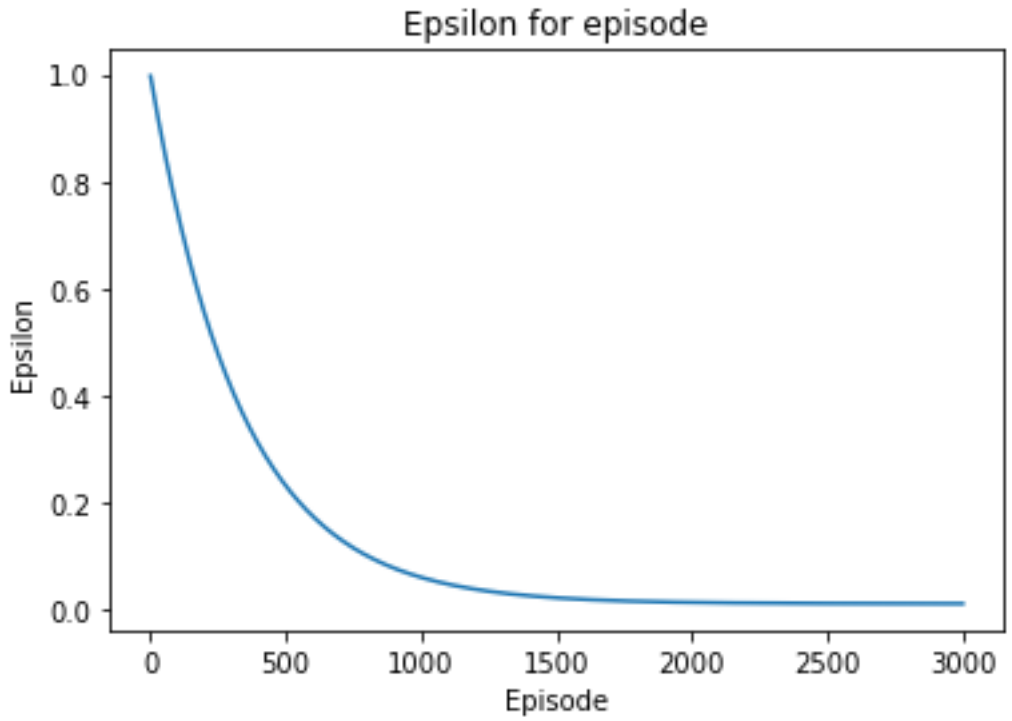
https://www.neutreeko.net/neutreeko.htm

Rewards:
Win: 20
Step: -1

Random
Agent

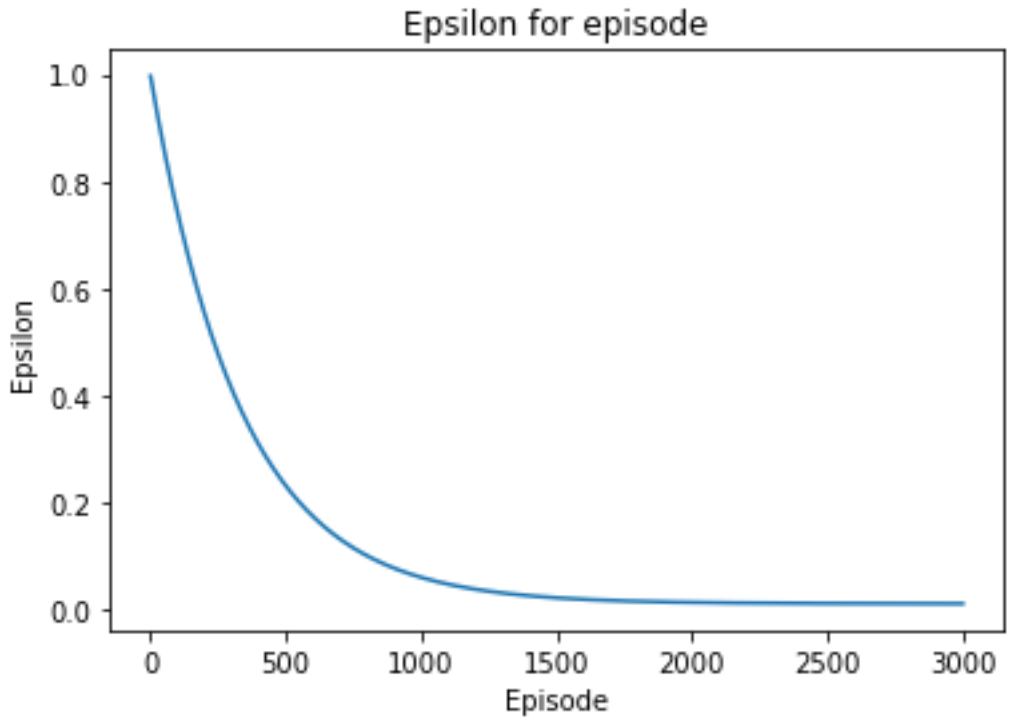


Q-Learning Agent



SARSA Agent


Total rewards over all episodes in training
Training total reward
25
0
−25
−50
−75
−100
−125
−150
−175
0
500
1000
1500
2000
2500
3000
Episode


Epsilon for episode
Epsilon
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
3000
Episode