Maze DRL

The goal for this project was to create an autonomous AI agent capable of navigating and solving mazes using Deep Reinforcement Learning.

Data Analysis

Reinforcement Learning deals with the state of the environment. Given that fact for this project a maze generator script, a simple game engine (for visualization), and a maze environment class were created.

A maze consists of a N x N size, with one or more solutions to a goal, a starting position at the left top corner and a goal position at the bottom right, there are no traps. Padding was added to some mazes. Here is an example of maze 5x5 with no padding:

1 1 1 1 1

1 3 0 0 1

1 1 1 0 1

1 0 0 4 1

1 1 1 1 1

0 = Free Space, 1 = Wall, 3 = Start, 4 = Goal

The maze environment class played an important role in this project, the reward system would help us navigate the maze more effectively. This system consisted in a big positive reward for reaching the goal, a small positive reward for doing a step, a small positive reward for moving closer to the goal (used manhattan distance), a small negative reward for moving farther from goal, a negative reward for reaching the max allowed steps, a negative reward for hitting a wall, and a negative reward for repeating the same spot.

Technical Approach

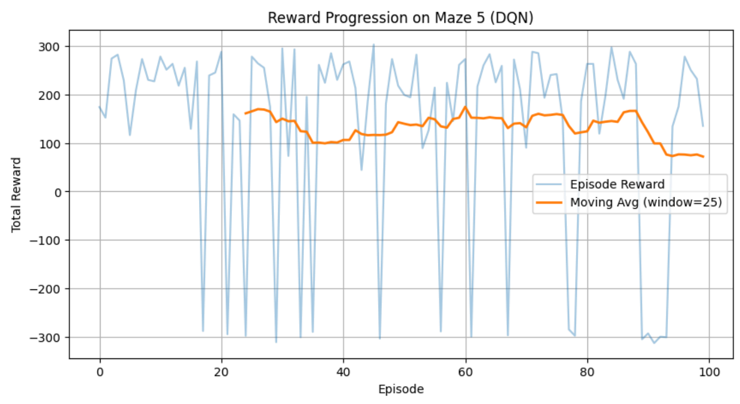
The Reinforcement Learning lectures and lab were used as a background for this project. The DQN and PPO approaches were used to try to solve this problem, and served as a foundation for a curriculum learning approach using PPO.

The Deep-Q-Network (DQN) approach consisted in creating an agent that would approximate Q-values through the usage of a local and a target network leveraging experience replay. This approach would select actions using an epsilon-greedy exploration strategy. The network architecture consisted in 4 linear layers with an activation function in between, the input consisted in batches of a flatten state with size of if is its original size squared times 2. A hidden size of 128 nodes and an output 4 representing each action. It follows an architecture of a simple fully connected network.

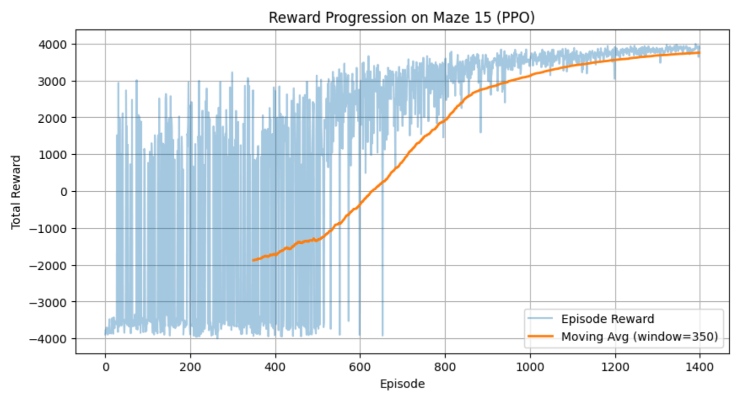
Proximal Policy Optimization (PPO) was the second approach used in this project. The PPO agent includes a policy and value network, each fully connected, the policy network consists of 6 hidden layers with a ReLU activation function and outputs a softmwax over the actions, the value network consists of 5 hidden layers with a ReLU activation function. The PPO algorithm updates the policy using a clipped surrogate objective to ensure stable learning, and the value function is trained with mean squared error loss. Training data is generated on-policy through agent-environment interaction, with performance evaluated using recent success rates. For this approach we tried learning a single maze and also progressively learning mazes with higher orders of complexity, we called this curriculum learning approach.

Results

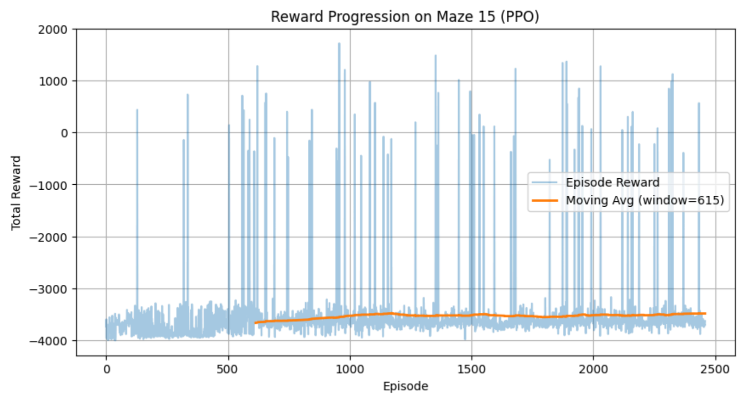
DQN on maze size 5, had trouble with bigger mazes.



Single training with PPO for maze size 15, success rate of 100%



Curriculum training for maze size 15, had a success rate of 0%



For the curriculum learning padding was added to the mazes, in fact it was aimed to learn mazes up until size 20, but it had trouble with size 15. It was capable of solving mazes from sizes 5 - 11, with the curriculum training.

Notebook: PPOCurriculum.ipynb

Time Log

Date	Time	Task
March 11	4 hrs	Create Maze Factory
April 12	2 hrs	Build Maze Environment class
April 15	4 hrs	DQN AI Agent class and usage of maze environment in training
April 15	3 hrs	Improved, debugged, and tested DQN
April 16	5 hrs	PPO AI Agent class
April 17	7 hrs	Debugged, train, tested PPO single agent
April 18	2 hrs	Achieved results with single agent
April 18	4 hrs	Curriculum learning development
April 19	5 hrs	Debugged, train, tested curriculum learning approach