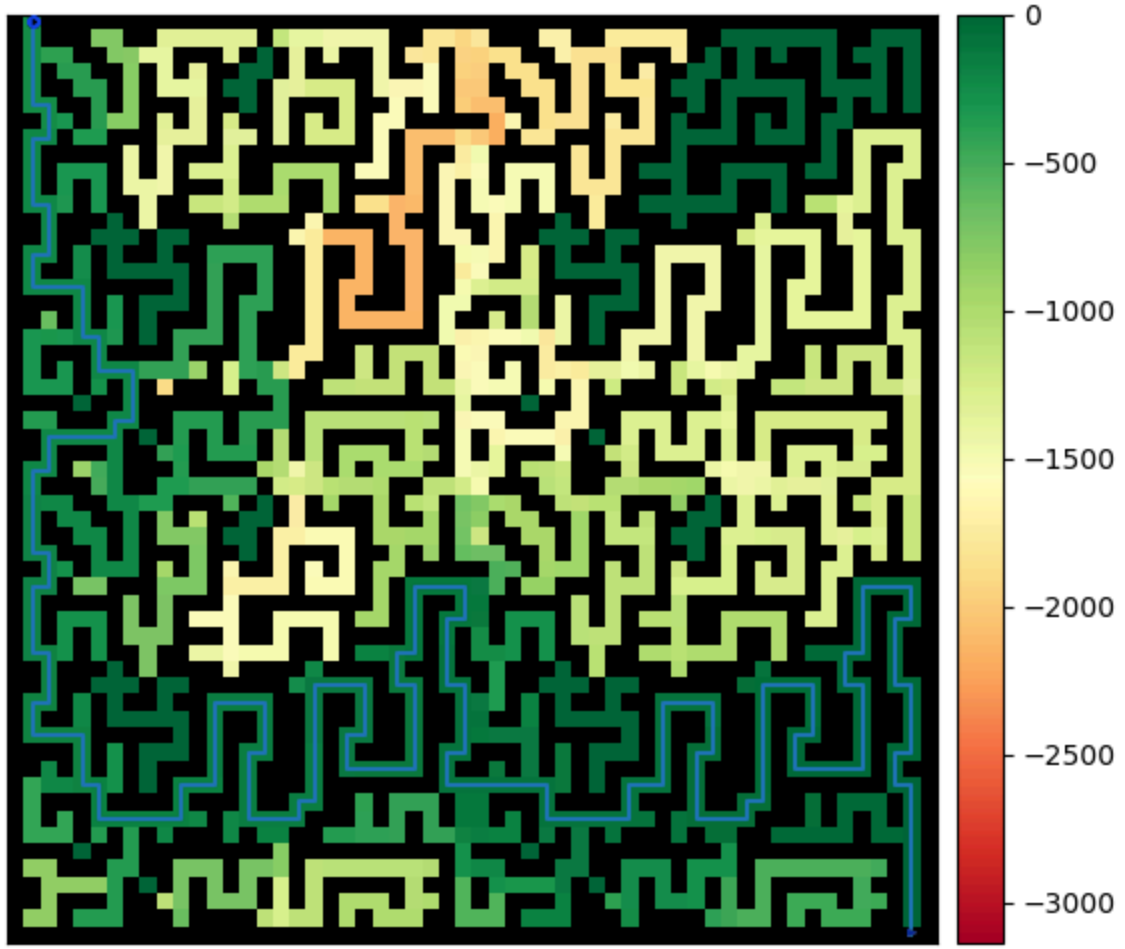
Solving a Maze - Reinforcement Learning



Contents

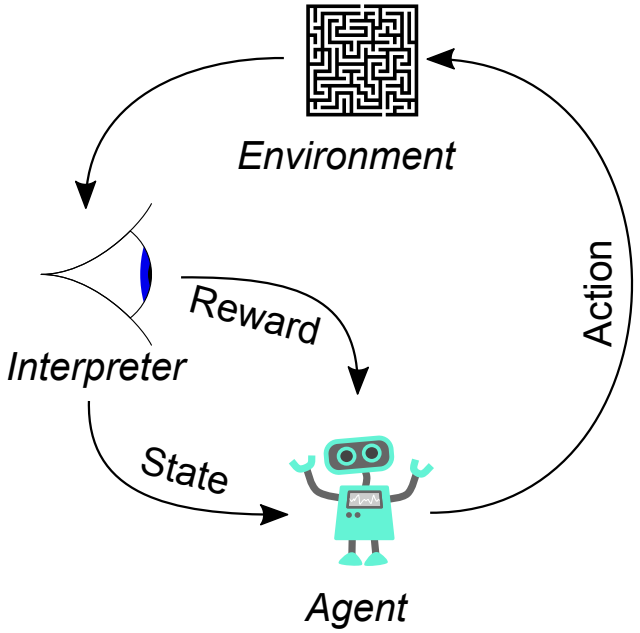
Reinfocement Learning

(Un)supervised

- Need of a training set
  - May be labeld (supervised)
  - Or unlabeld (unsupervised)

Reinforcement

> [...] intelligent agent ought to take actions in a dynamic environment in order to maximize the *cumulative reward*.
> -- *Wikipedia*



The Environment

- Provide a system, with which the agent can interact
- Implement game logic

```
class Maze():

  def get_moves():
  def move(...):

  def get_state_and_reward():

  def is_game_over():

  # just for plotting
  def get_steps()
```

- Stores:
  - Actual maze
  - Player position
- Provides to agent:
  - Possible moves
  - Rewards
- Ends game

The Agent

- Robot which learns to solve the maze

```
class Agent():

  def __init__(...):
    """set learning/exploration rate"""

  def choose_action(...):

  def store_rewards(...):

  def learn():
```

- Initilization with specific
  - learning rate
  - exploration rate
- Choose from possible moves
- Store reward for given positon
- update reward table from rewards collected during one game (learn)

Reward System

Environment

```
def get_state_and_reward(self):
  return self.pos, -1
```

- penalize each step with -1

Agent

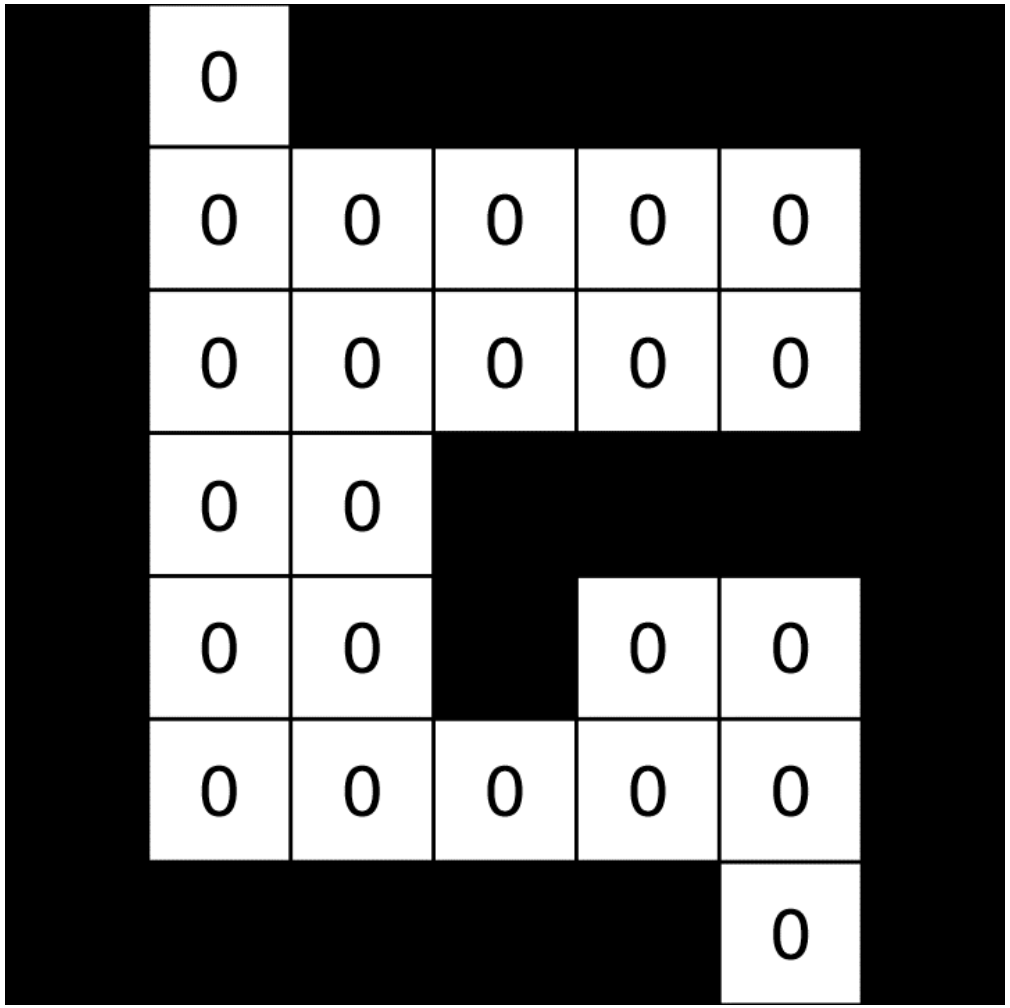
- During the game

```
def store_reward(self, pos, reward):
  self._pos_history.append(tuple(pos))
  self._rew_history.append(reward)
```

- After the game `learn()`

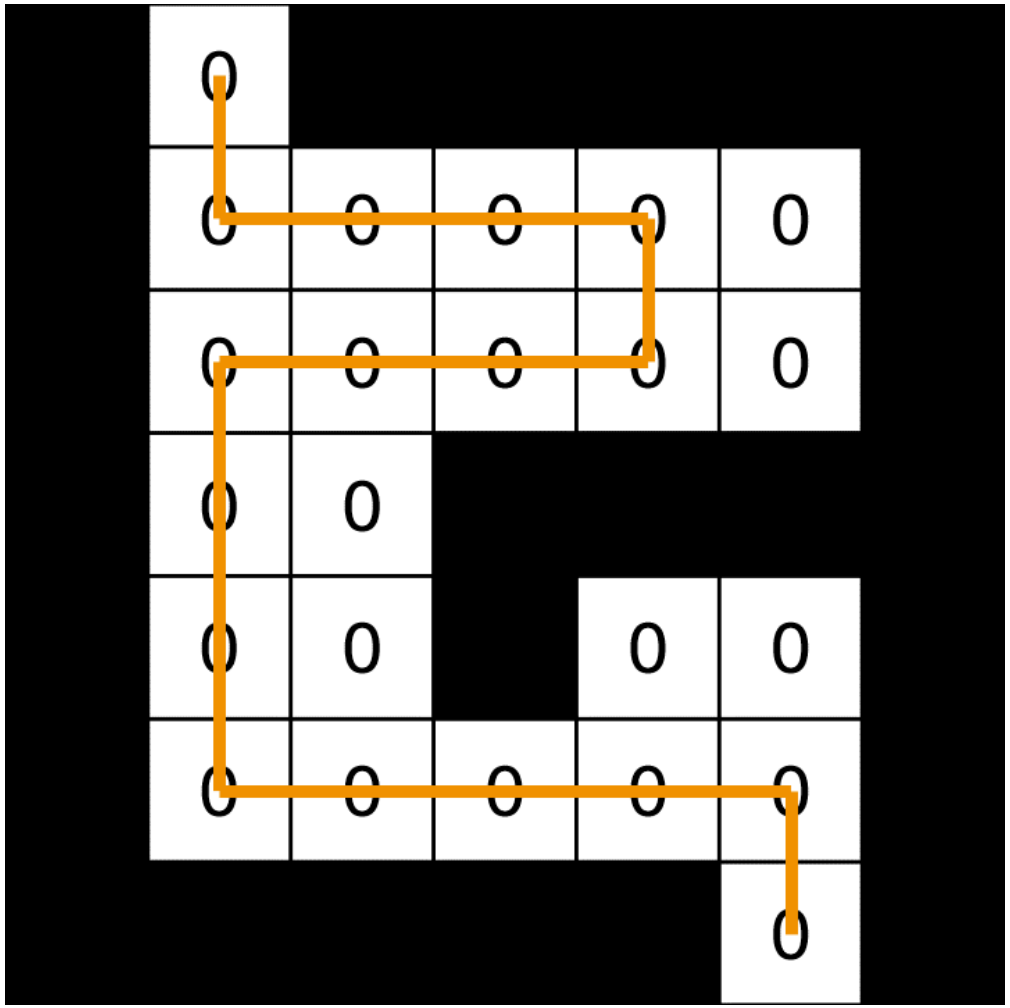
Reward System - *Agent*

- Agent stores a reward matrix
  - tecnically **G table** *(cumulative)*
- initiallized to zero



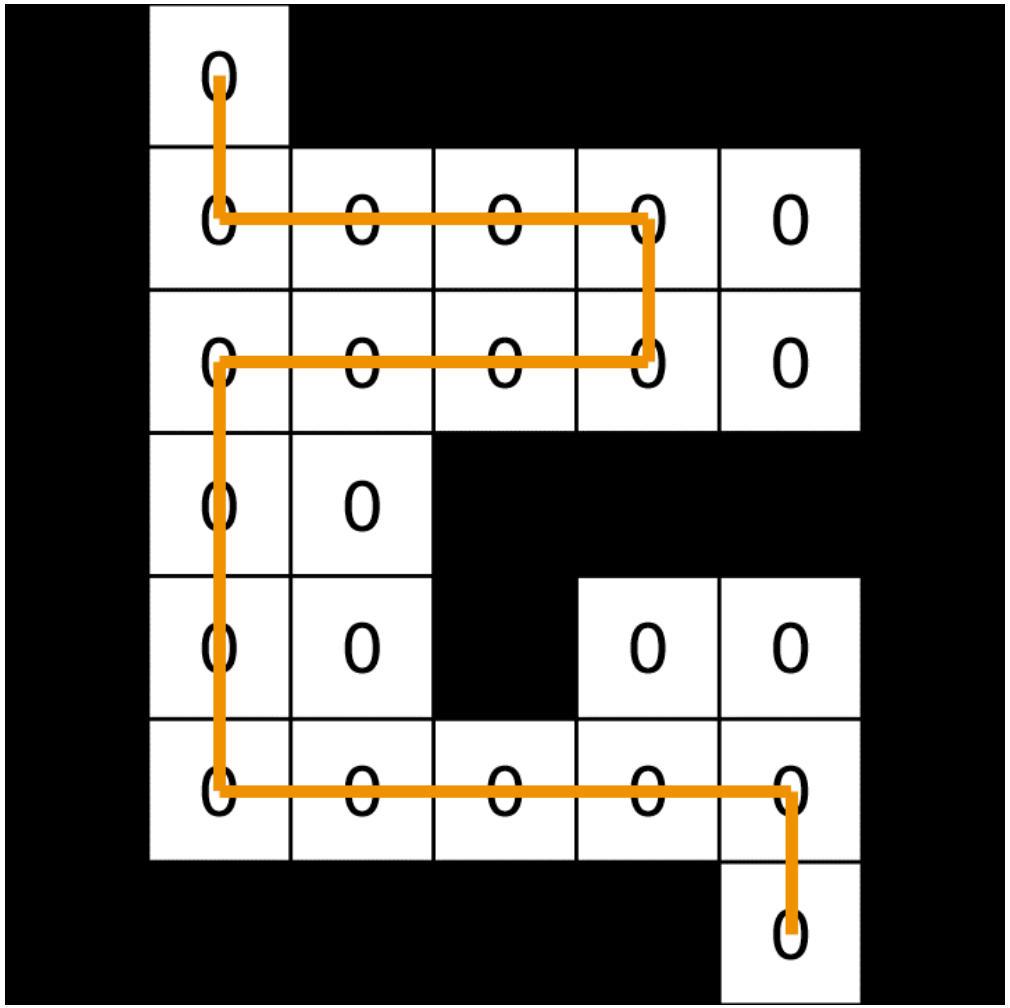
Reward System - *Agent*

- First game is random



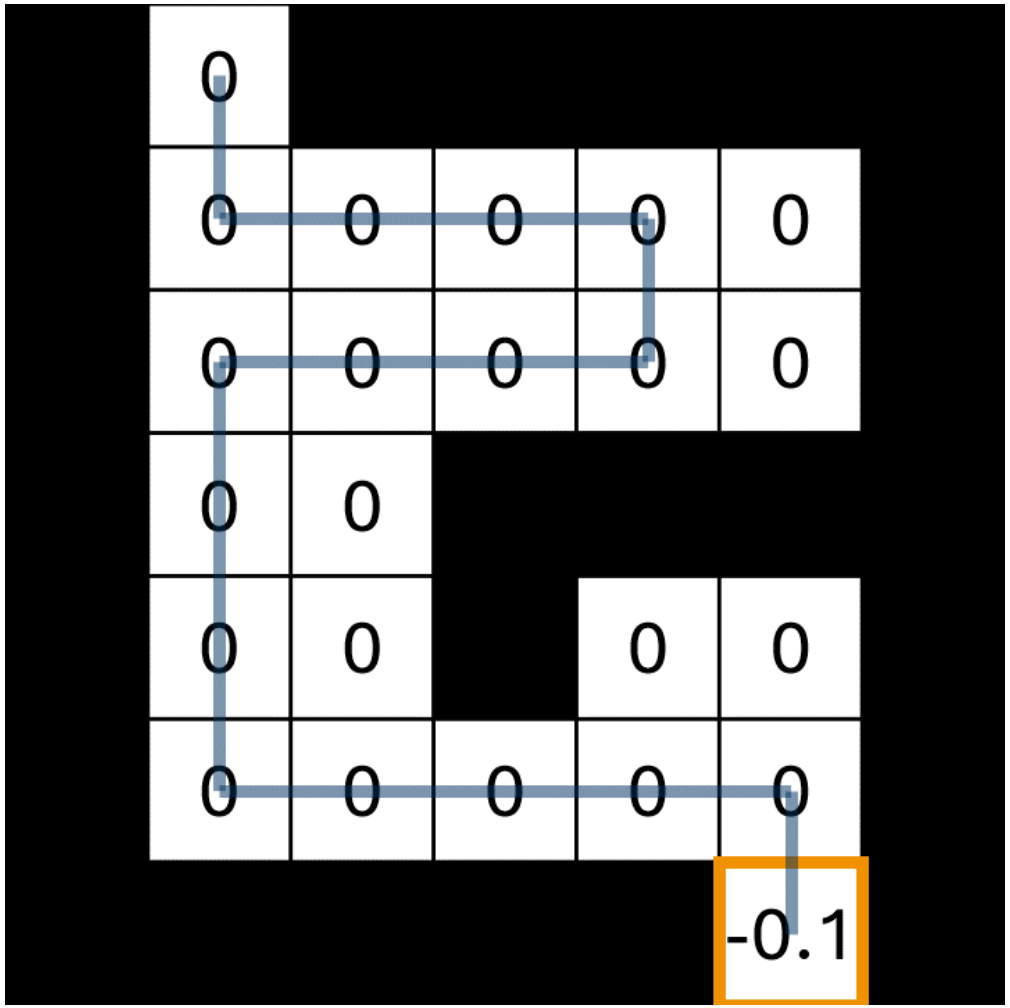
Reward System - *Agent - learning*

$$G_{\text{path}} = 0$$



Reward System - *Agent - learning*

- Loop path **backwards**



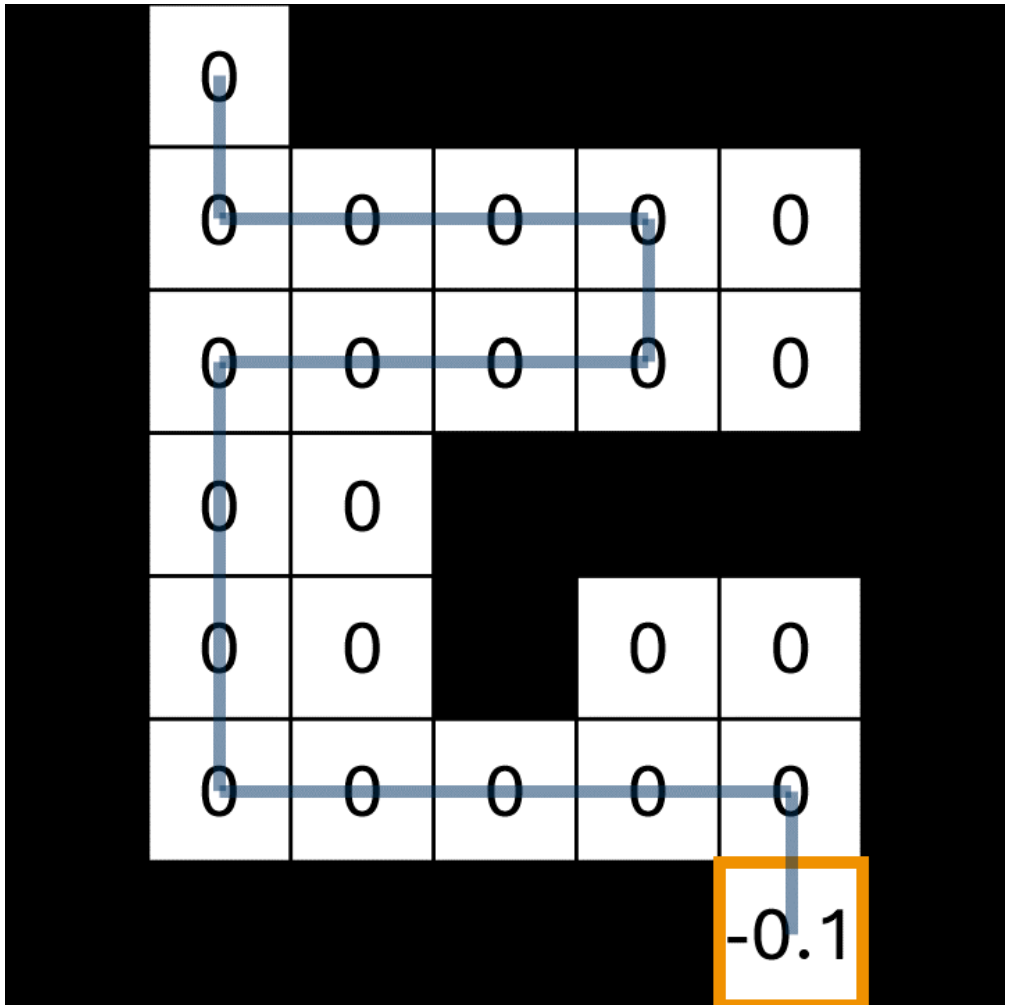
Reward System - *Agent - learning*

- Loop path **backwards**
- Compute cumulative reward along path

$$reward_{\text{pos}} = -1$$

$$G_{\text{path}} = G_{\text{path}} + reward_{\text{pos}}$$

$$G_{\text{path}} = -1$$



Reward System - *Agent - learning*

- Loop path **backwards**
- Compute cumulative reward along path

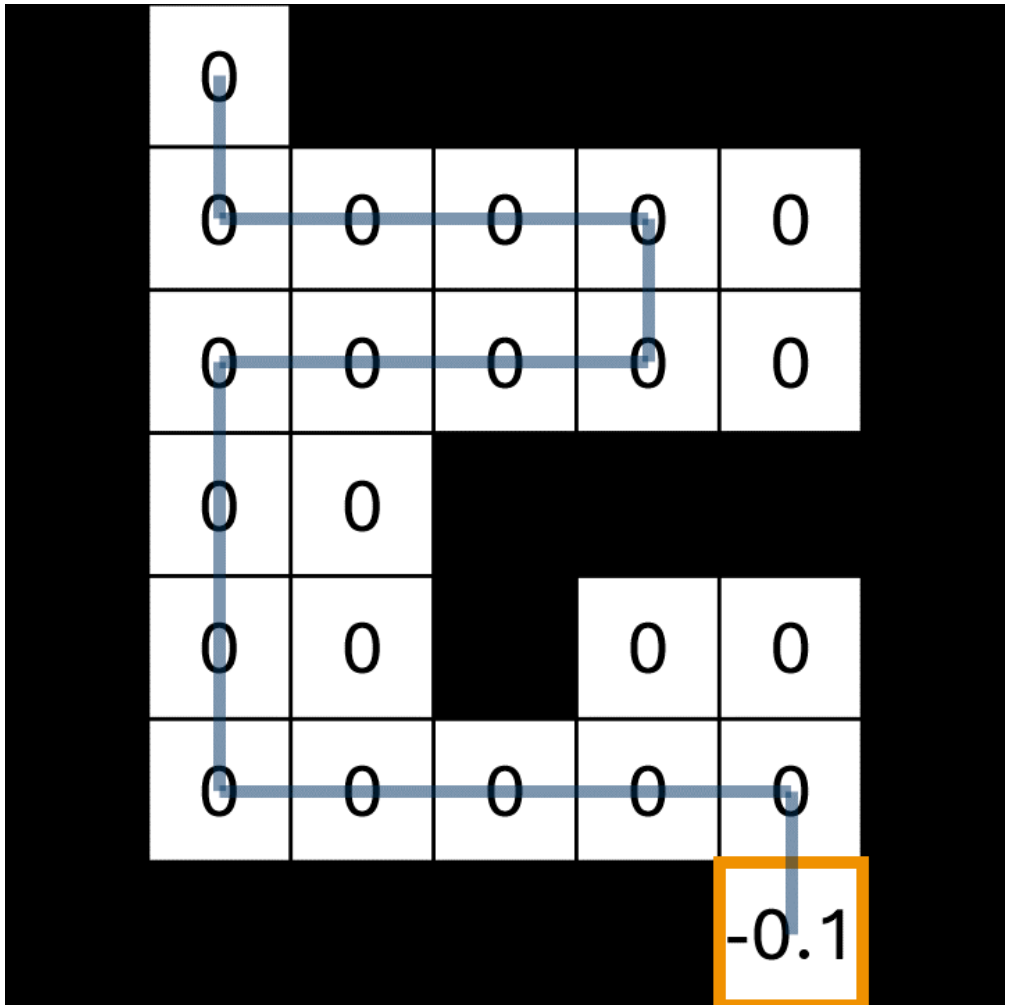
$$reward_{\text{pos}} = -1$$

$$G_{\text{path}} = G_{\text{path}} + reward_{\text{pos}}$$

$$G_{\text{path}} = -1$$

- Update reward matrix entry (eg. $\alpha = 0.1$)

$$G_{\text{pos}} = G_{\text{pos}} + \alpha(G_{\text{path}} - G_{\text{pos}}) = -0.1$$



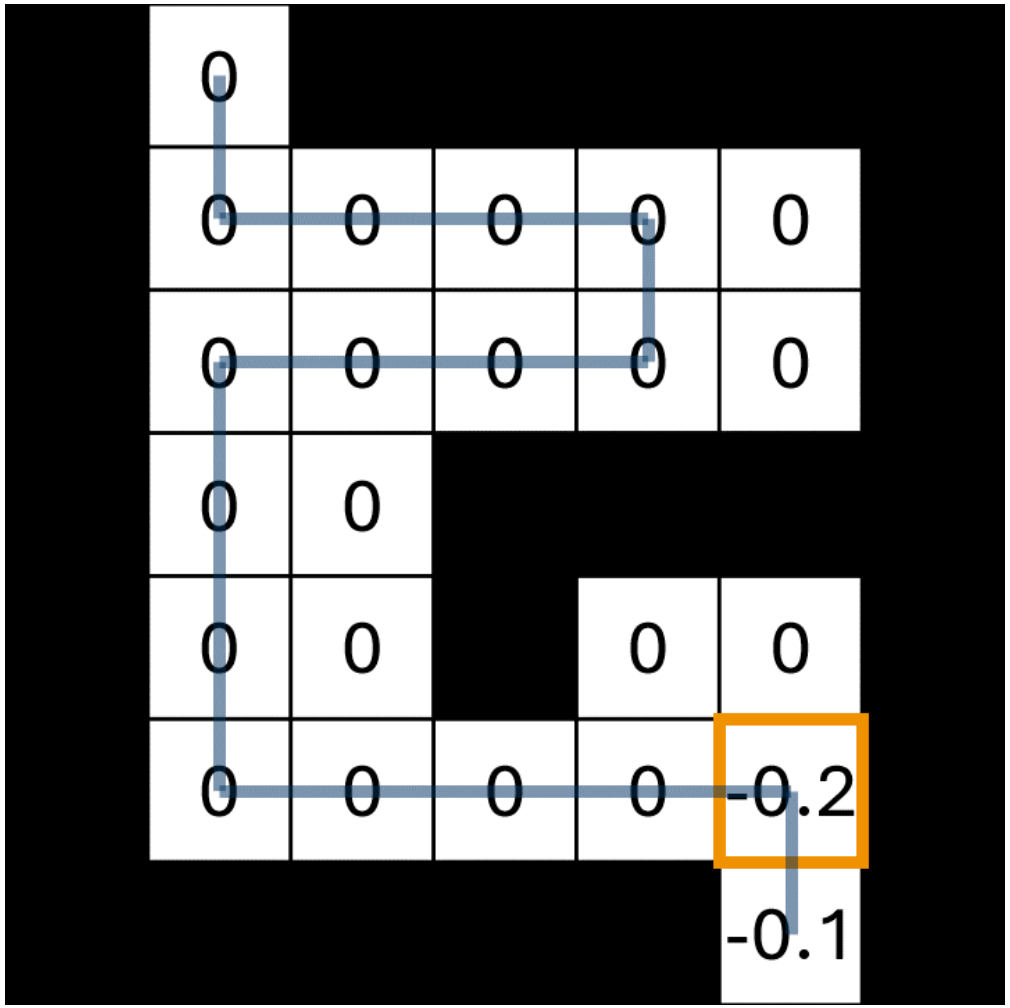
Reward System - *Agent - learning*

$$reward_{\text{pos}} = -1$$

$$G_{\text{path}} = G_{\text{path}} + reward_{\text{pos}}$$

$$G_{\text{path}} = -2$$

$$G_{\text{pos}} = G_{\text{pos}} + \alpha(G_{\text{path}} - G_{\text{pos}}) = -0.2$$



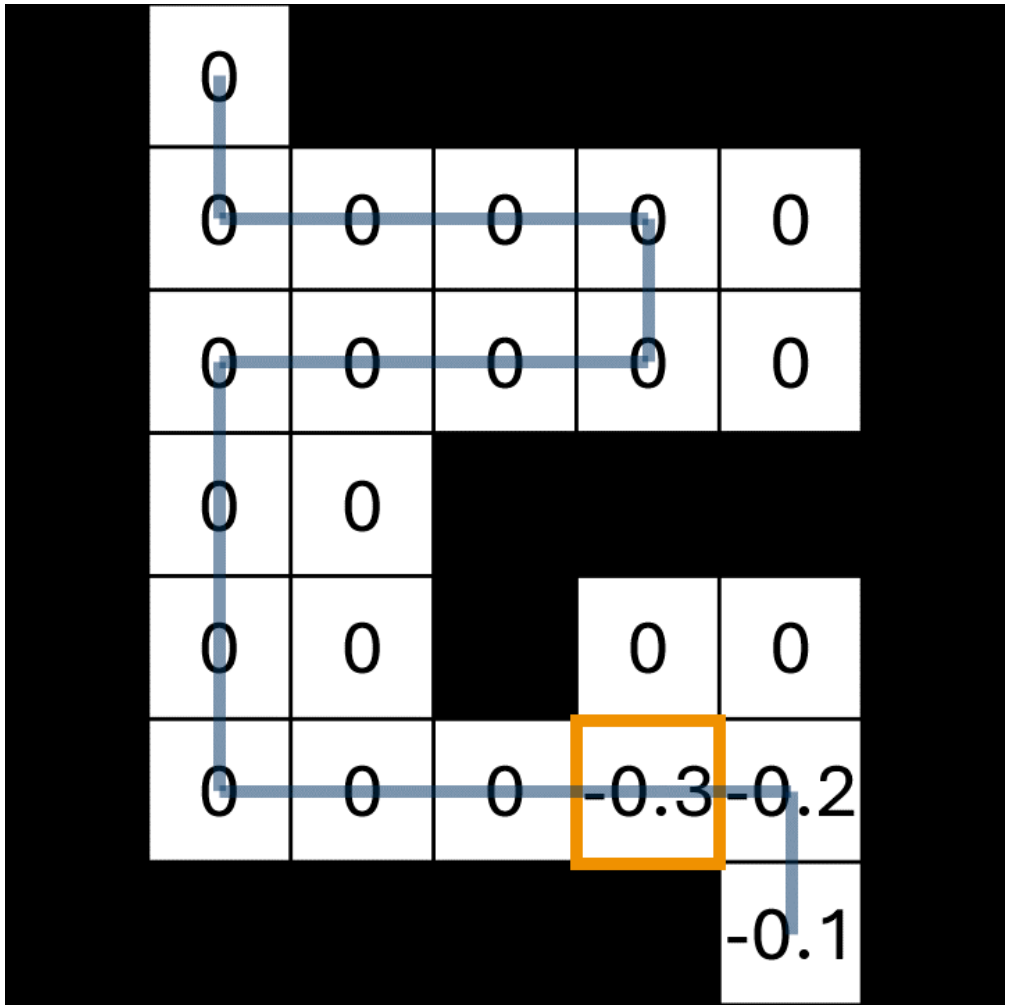
Reward System - *Agent - learning*

$$reward_{\text{pos}} = -1$$

$$G_{\text{path}} = G_{\text{path}} + reward_{\text{pos}}$$

$$G_{\text{path}} = -3$$

$$G_{\text{pos}} = G_{\text{pos}} + \alpha(G_{\text{path}} - G_{\text{pos}}) = -0.3$$



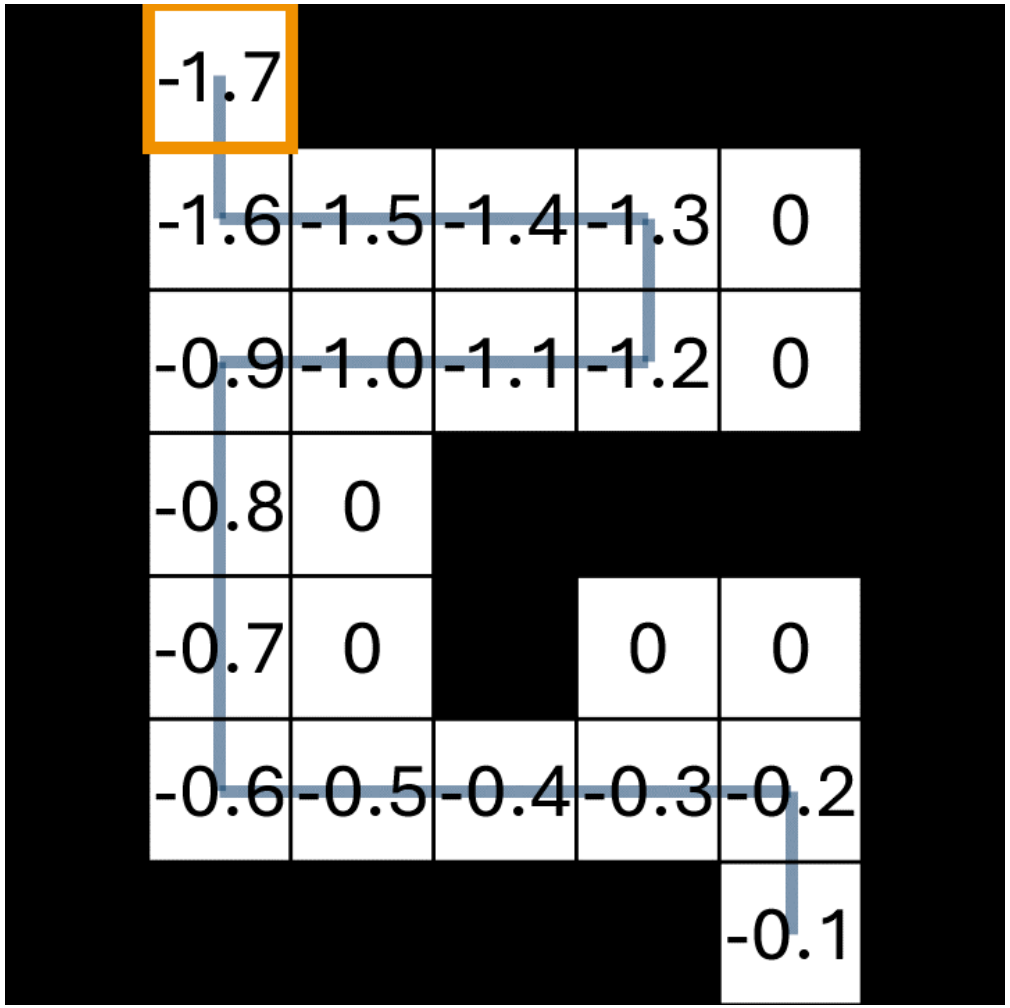
Reward System - *Agent - learning*

$$reward_{\text{pos}} = -1$$

$$G_{\text{path}} = G_{\text{path}} + reward_{\text{pos}}$$

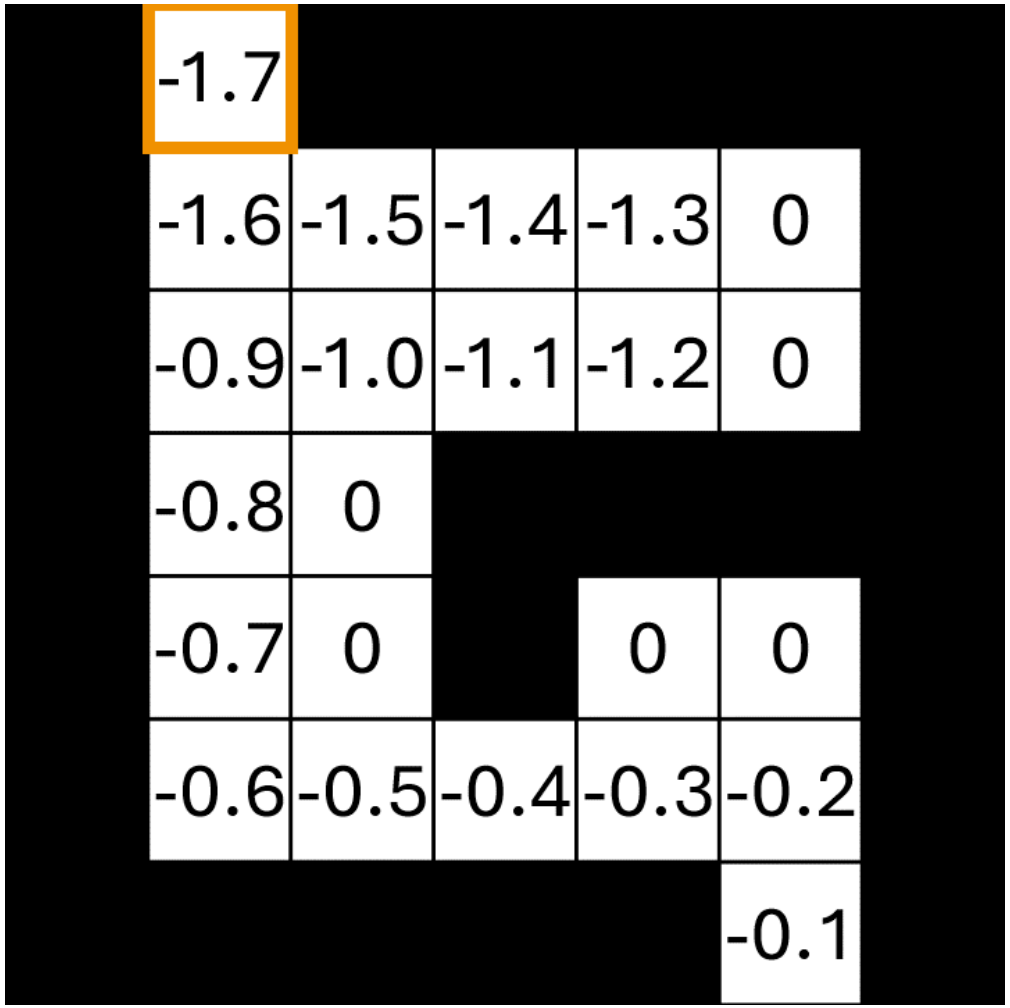
$$G_{\text{path}} = -17$$

$$G_{\text{pos}} = G_{\text{pos}} + \alpha(G_{\text{path}} - G_{\text{pos}}) = -1.7$$

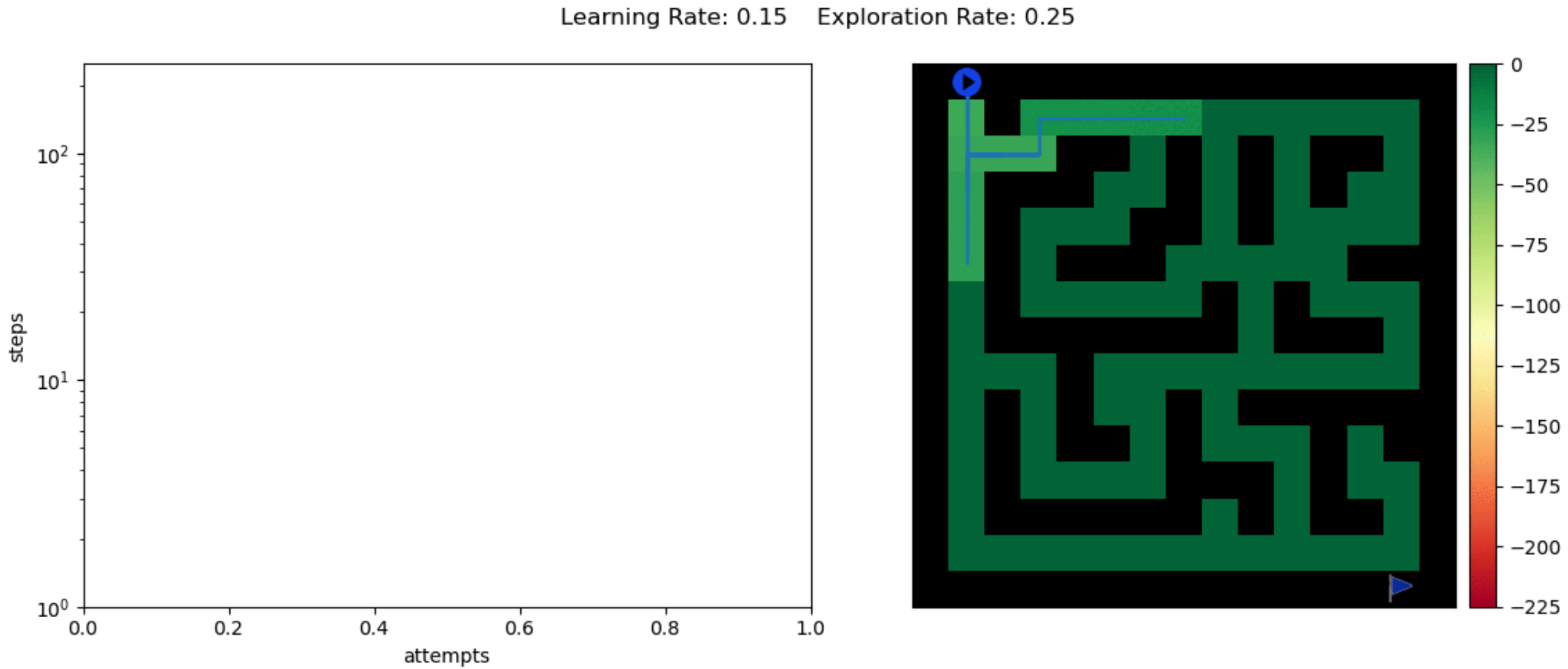


Reward System - *Agent - choosing*

- Consider second run
- Agent has learned (some) reward matrix
- Choose action towards maximum reward
- Exploration rate:
  - Choose random action by chance

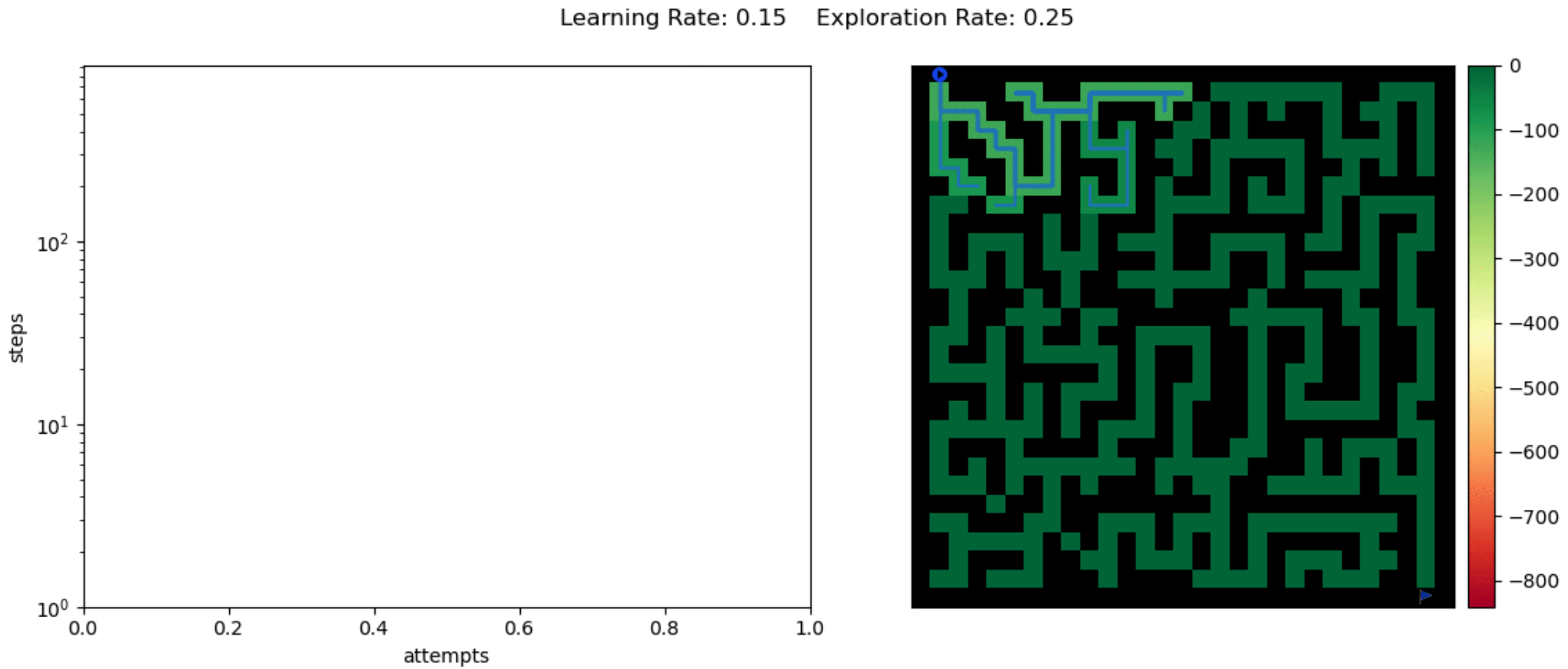


Small Maze

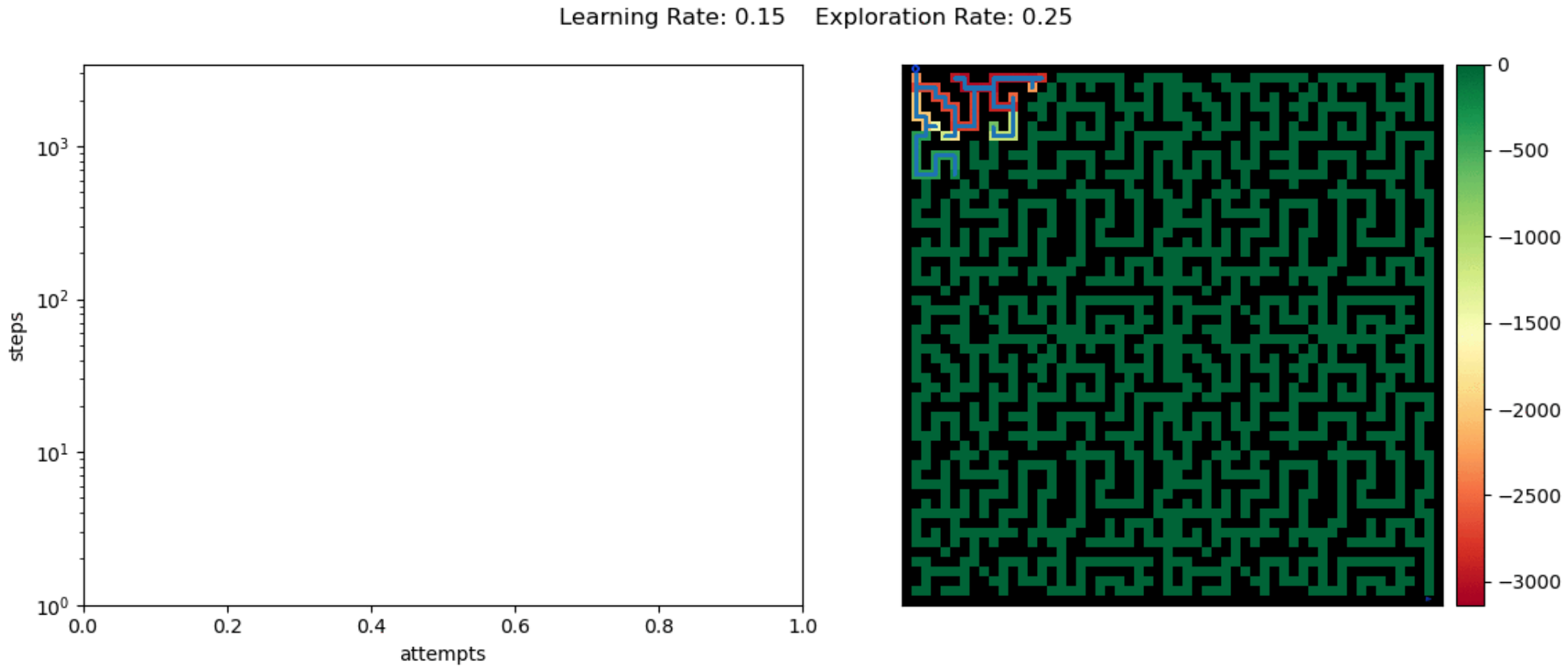


- Remove Backtracking

Bigger!


Learning Rate: 0.15 Exploration Rate: 0.25
steps
10^2
10^1
10^0
0.0
0.2
0.4
0.6
0.8
1.0
attempts
0
−100
−200
−300
−400
−500
−600
−700
−800

And BIGGER!



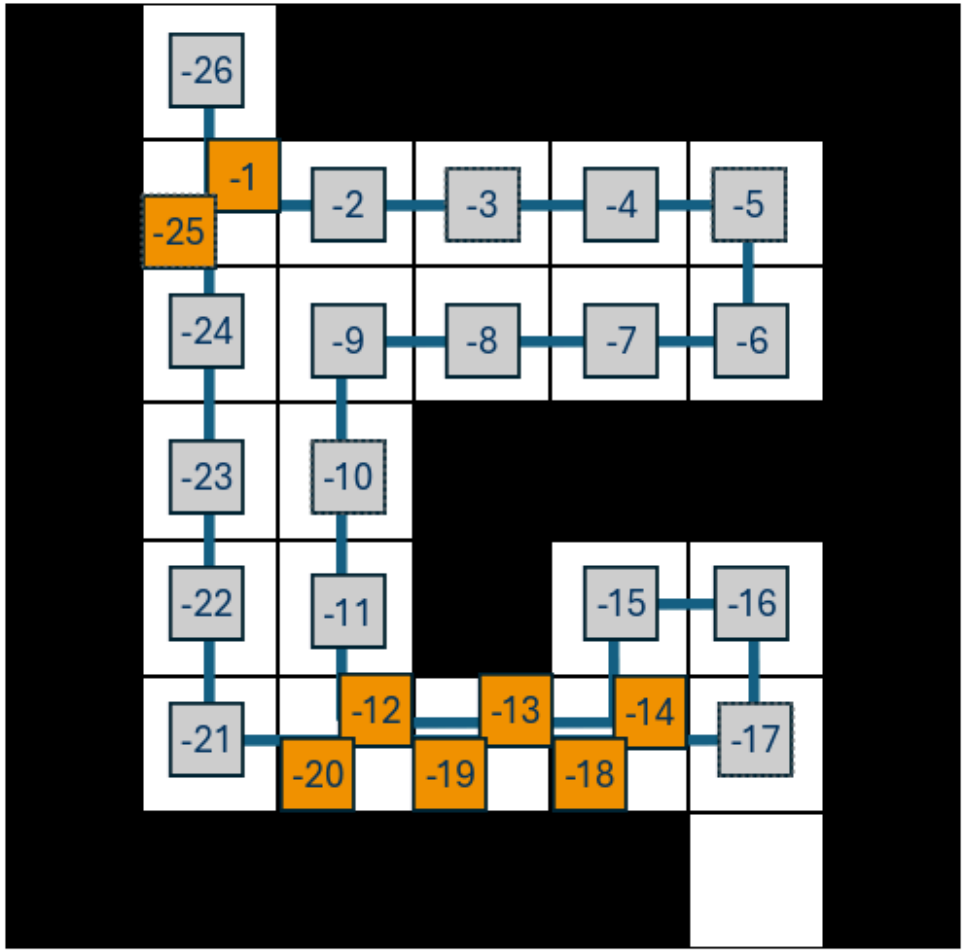
- Why arent dead ends deep red?

Cumulative Reward

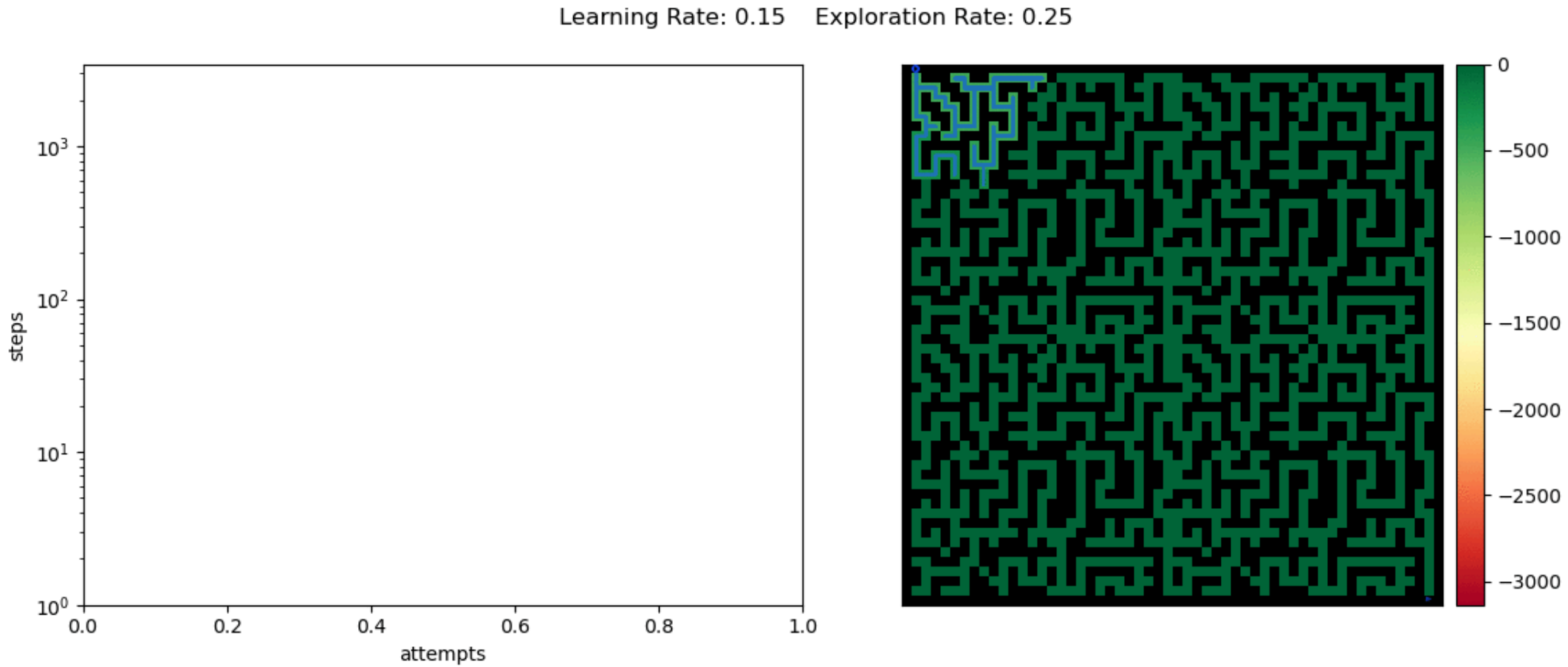
$$G_{\text{pos}} = G_{\text{nos}} + \alpha(G_{\text{path}} - G_{\text{pos}})$$

$$\Rightarrow$$

$$G_{\text{pos}} = G_{\text{pos}} + \alpha(\min\{G_{\text{path}}\} - G_{\text{pos}})$$



Hurray!


Learning Rate: 0.15 Exploration Rate: 0.25
steps
attempts
10^3
10^2
10^1
10^0
0.0
0.2
0.4
0.6
0.8
1.0
0
−500
−1000
−1500
−2000
−2500
−3000

Thanks for your attention

Appendix

Removing Backtracking

- If action returns to previous position
  - Remove action from `action_list` and choose again (except for dead end)

```
def chose_action(action_list):
  ...
  # attempt to make him not reverse moves immediatly
  # If action returns to previous position
  # - remove action from action list if possible (ie. not only option)
  # - choose again
  if tuple(action_list[index]()) == self._pos_history[-2]:
    if len(action_list) > 1:
      action_list.pop(index)
      return self.choose_action(action_list)
```

- May not backtrack directly
- Can still run in loops