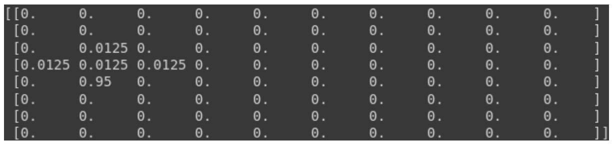
CS470 : Introduction to Artificial Intelligence

Gabriel SCHWAB

May 2, 2023

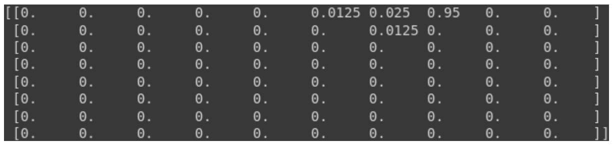
1 Markov Decision Process

Transition probabilities to all the next states given :



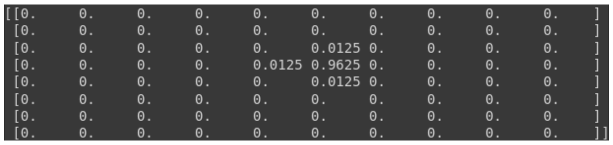
```
[[0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.0125 0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.0125 0.0125 0.0125 0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.95   0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]]
```

Figure 1: Transition probabilities with s_t = [3,1] and a_t = "DOWN"



```
[[0.     0.     0.     0.     0.     0.0125 0.025  0.95   0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0125 0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]]
```

Figure 2: Transition probabilities with s_t = [0,6] and a_t = "RIGHT"



```
[[0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.0125 0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.0125 0.9625 0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.0125 0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]]
```

Figure 3: Transition probabilities with s_t = [3,5] and a_t = "RIGHT"

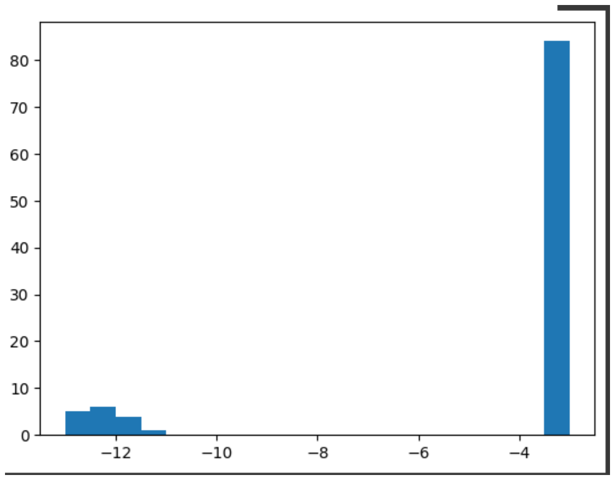


Figure 4: Histogram of return using dummy Policy for 100 episodes

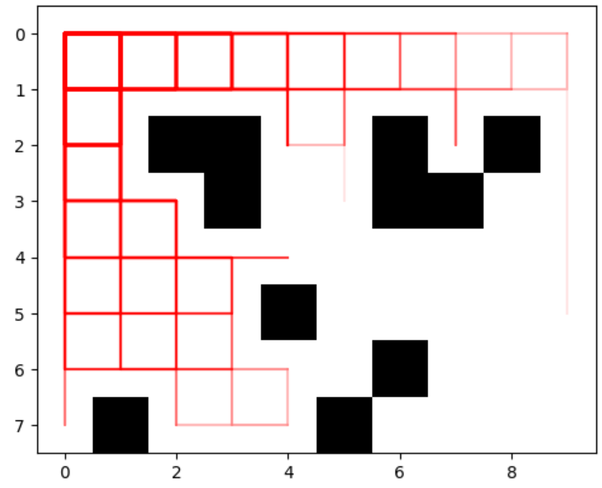


Figure 5: Distribution of trajectories produced by a dummy policy for 300 episodes

```
def transition_model(self, state, action):
    ####################################################################
    #####################   PLACE YOUR CODE HERE   ####################
    ###                                                              ###
    if state == self.terminal_state or state in self.traps:
      probs[state] = 1
      return probs

    current_pos = self.serial_to_twod(state)

    action_probs = np.full(5, self.epsilon/4)
    action_probs[action] = 1 - self.epsilon

    for new_action, prob in enumerate(action_probs):
      next_pos = current_pos + action_pos_dict[new_action]

      if next_pos[0] < 0 or next_pos[1] < 0 or \
            next_pos[0] >= self.grid_map_shape[0] or next_pos[1] >= self.grid_map_shape[1]:
        next_state = state
        probs[next_state] += prob
      elif self.twod_to_serial(next_pos) in self.obstacles:
          next_state = state
          probs[next_state] += prob
      else:
        next_state = self.twod_to_serial(next_pos)
        probs[next_state] += prob
    ###                                                                ###
    ######################################################################
    ######################################################################
```

```
def compute_reward(self, state, action, next_state):
    ######################################################################
    #####################   PLACE YOUR CODE HERE   #####################
    ###                                                                ###
    if next_state == self.terminal_state:
      reward += 5
    elif next_state in self.traps:
      reward -= 10

    reward -= 0.1
    ###                                                                ###
    ######################################################################
    ######################################################################
```

```
def is_done(self, state, action, next_state):
    ######################################################################
    #####################   PLACE YOUR CODE HERE   #####################
    ###                                                                ###
    if next_state in self.traps or next_state == self.terminal_state:
      done = True
    else:
      done = False
    ###                                                                ###
    ######################################################################
    ######################################################################
```

```
def step(self, action):
    ######################################################################
    #####################   PLACE YOUR CODE HERE   #####################
    ###                                                                ###
    next_state = np.random.choice(np.arange(self.observation_space.n), p=probs)
    p = probs[next_state]
    reward = self.compute_reward(self.observation, action, next_state)
    done = self.is_done(self.observation, action, next_state)
    self.observation = next_state
    ###                                                                ###
    ######################################################################
    ######################################################################
```

2 Dynamic Programming

2.1 Value Iteration

The first value of the 8 states of the gridworld environment are : 5.45728406, 6.23678565, 7.13122724, 8.13483629, 9.26075762, 10.52456869, 11.87047723, 10.4561357.

This is the best action at each state based on the state-action values :

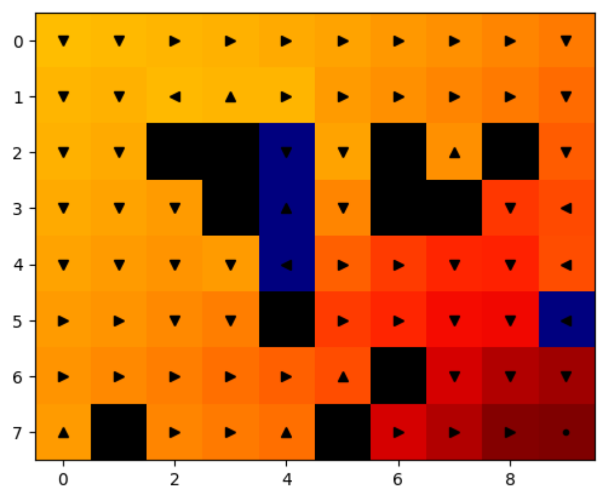


Figure 6: Policy of the gridworld environnement after value iteration

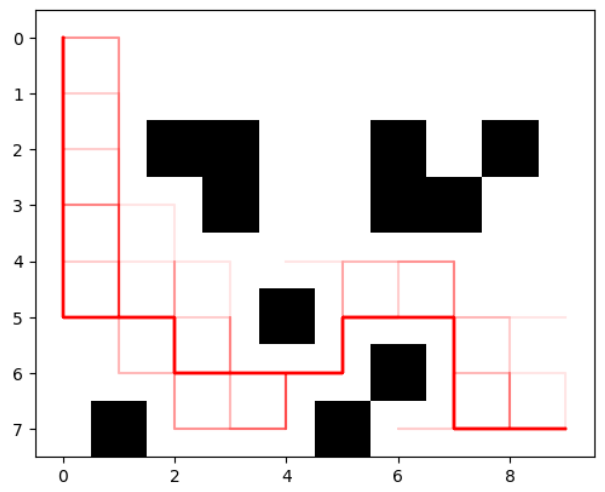


Figure 7: Distribution of trajectories produced by the trained policy for 100 episodes

```
def value_iteration(self):
            ##############################################################
            #####################   PLACE YOUR CODE HERE   ################
            ###                                                        ###

            for action in range(self.env.action_space.n):
              transitions = self.env.transition_model(state, action)
              for next_state, prob in enumerate(transitions):
                reward = self.env.compute_reward(state, action, next_state)
                Q[action] += prob * (reward + self.discount_factor * self.V[next_state])

            max_error = max(max_error, abs(np.max(Q) - self.V[state]))
            self.V[state] = np.max(Q)
            ###                                                        ###
            ##############################################################
            ##############################################################
```

```
def get_action(self, state):
    ##################################################################
```

```python
    ####################   PLACE YOUR CODE HERE   ###################
    ###                                                          ###

    for action in range(self.env.action_space.n):
      transitions = self.env.transition_model(state, action)
      for next_state, prob in enumerate(transitions):
        reward = self.env.compute_reward(state, action, next_state)
        Q[action]+=prob * (reward + self.discount_factor * self.V[next_state])

    max_actions = np.where(Q == np.max(Q))[0]
    action = np.random.choice(max_actions)
    ###                                                          ###
    ################################################################
    ################################################################
```

2.2 Comparison under different transition models

Increasing the value of ϵ increases the probability of taking a random action. This favors exploration over exploitation. We notice that the expected return does not stabilize during the iterations because the chosen actions are too random. Whereas with $\epsilon = 0.1$, we learn the optimal policy more quickly and thus we obtain almost every time the end reward.

Concerning the map that describes the best policy, we notice that for the one with $\epsilon = 0.1$, the state-value are higher than for $\epsilon = 0.4$. The value iteration algorithm has therefore converged faster with $\epsilon = 0.1$.

Finally, we notice that as ϵ increases, the distribution of trajectories is more dispersed.

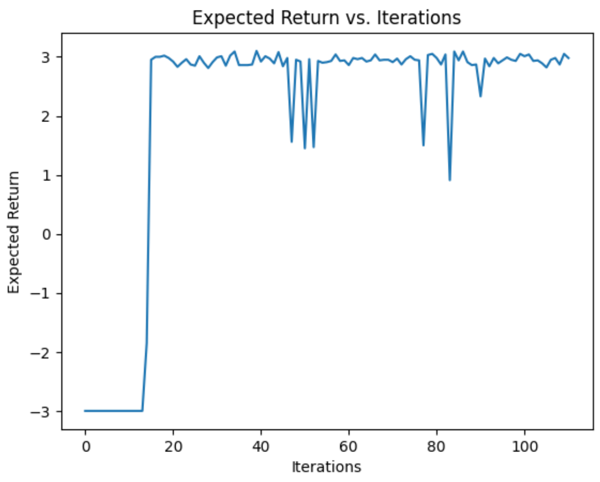


Figure 8: Expected return for $\epsilon = 0.1$

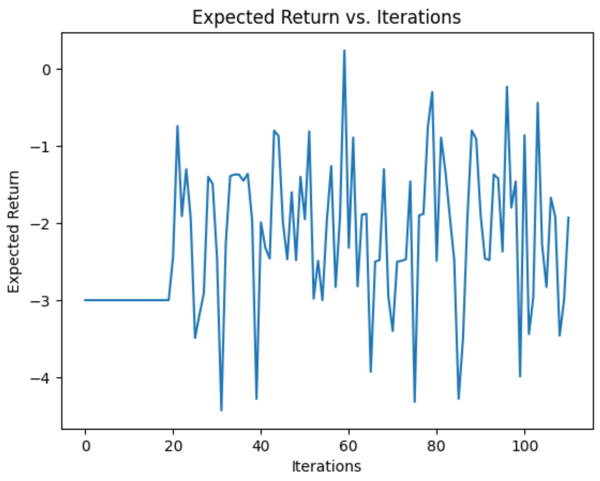


Figure 9: Expected return for $\epsilon = 0.4$

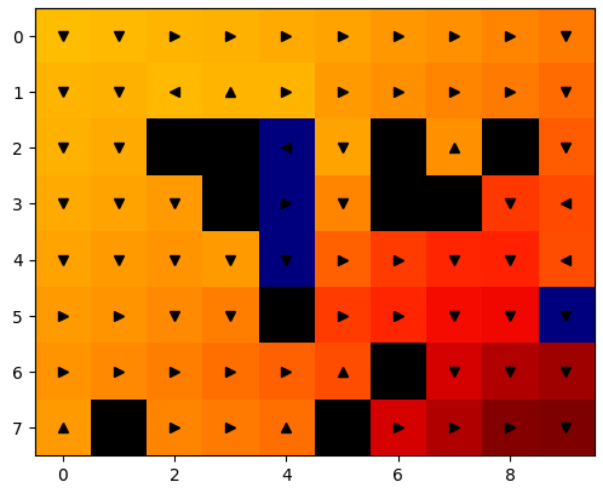


Figure 10: Best action at each state for $\epsilon = 0.1$

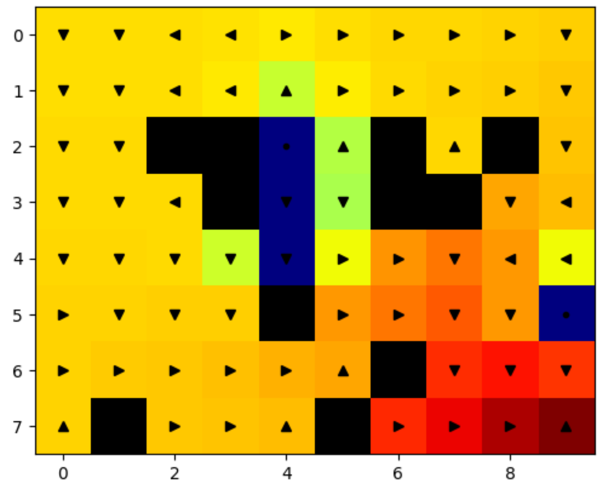


Figure 11: Best action at each state for $\epsilon = 0.4$

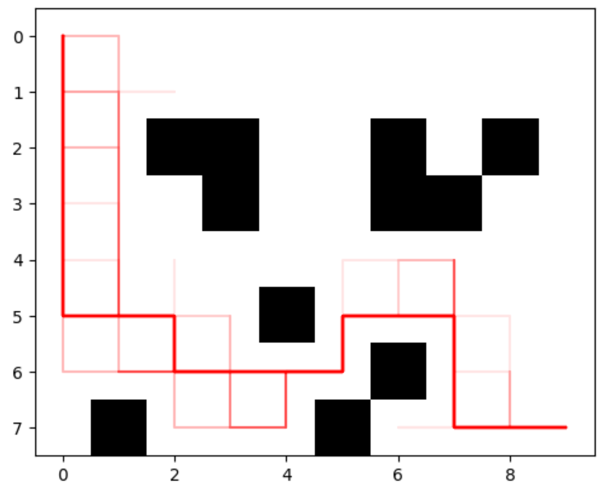


Figure 12: Distribution of trajectories for $\epsilon = 0.1$

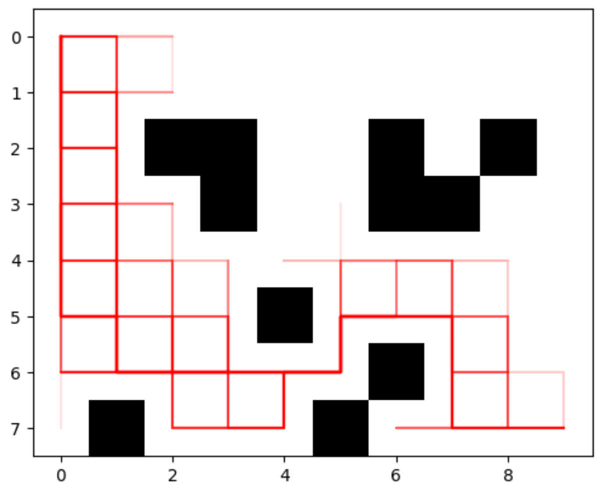


Figure 13: Distribution of trajectories for $\epsilon = 0.4$