Homework 3

Part 1

1. **MDP:** Markov Decision Processes are a tool for modeling sequential decision-making problems where a decision maker interacts with the environment in a sequential fashion. It's goal is to select actions that maximize expected future reward. A Markov Decision Process is a tuple $< S, A, P, R, \gamma >$. Where S is finite set of state, A is finite set of actions, P is state transition matrix: $p(s^{'}|a, s) = p(S_{t+1} = s^{'}|s_t = s, A_t = a)) = P^{a}_{ss'}$, R is the reward function: $r(s, a) = E[R_{t+1} = r|S_t = s, A_t = a] = R^{a}_{s}$ and $\gamma \in [0, 1]$ is the discount factor. As it was defined, MDP can only solve problems where all the states and actions are given (and has a finite number), or in other words, the model of the environment is known and fully observant. The following MDP example is taken from medium.com

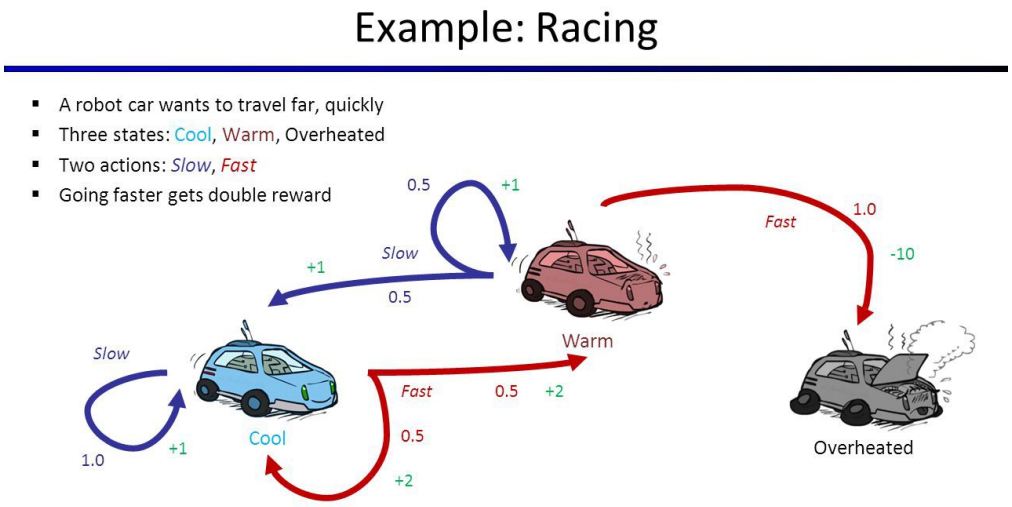

Figure 1: MDP example

2. **State-value function:** The state-value function $v_\pi(s)$ for policy π is the expected return (total discounted future reward) starting from state s, and then following policy π. The mathematical definition will be:

$$v_\pi(s) =: E_\pi[G_t|S_t = s] = E[G_t|S_t = s, \pi] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s, \pi] \quad (1)$$

Where $E_\pi[\cdot]$ is expected value of a random variable given that the agent follows policy π. The state-value function describes how good it is to be in a particular state s.

3. **Action-value function:** The action-value function $q_\pi(s, a)$ for policy π is the expected return (total discounted future reward) starting from state s, taking an (arbitrary) action a and then following policy π. The mathematical definition of $q_\pi(s, a)$:

$$\begin{aligned} v_\pi(s) =: E_\pi[G_t|S_t = s, A_t = a] = E[G_t|S_t = s, A_t = a, \pi] = \\ = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s, A_t = a, \pi] \end{aligned} \quad (2)$$

The action-value function describes how good it is to take a particular action a (not necessarily from policy π) from a given state s and then following policy.

4. **Policy:** An optimal policy is a mapping from states to actions that maximizes expected total future rewards. A policy defines the agent's behavior. There are two types of policies, deterministic policy - $a_t = \pi(s_t)$ and stochastic policy - $\pi(a|s) = p(A_t = a|S_t = s, \pi)$

5. **Dynamic programming:** dynamic programming is a technique that help in solving a problem where a naive solution will take a lot more to solve. For this work, dynamic programming is used for planning in an MDP. Planning can be divided into two tasks: prediction (evaluate a given policy) and control (find the best policy).

6. **Value iteration:** First, solve Bellman optimally equation to find the optimal value-function $v_*(s)$:

$$v_*(s) = \underset{a \in A}{max}[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')], \forall s \in S \tag{3}$$

   Second, Derive optimal policy π^* from the optimal value-function $v_*(s)$:

$$\pi^*(s) = \underset{a \in A}{argmax}[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')] \tag{4}$$

   This policy is the greedy policy, denoted $\pi_{\text{greedy}}(v)$, because it greedily selects the best action using the value function v. In short: value iteration - $v_*(s) \Rightarrow \pi^*(s)$.

7. **Policy iteration:** The policy iteration algorithm is composed out of a policy evaluation and policy improvement step. First, policy evaluation: for a given policy π the value function v_π is estimated. Second, policy improvement: from the value function v_π an improved policy is derived following a greedy strategy. Third, This cycle is repeated until convergence to the optimal policy π^* and optimal value function v^* is obtained.

8. **Reinforcement Learning (RL):** A tool to build agents that learn behaviors in a dynamic, uncertain and often unstructured world. As opposed to agents that execute manually pre-programmed behavior in a static, deterministic and highly structured world. Behavior is a sequence of actions with a particular goal. RL is learning by trial-and-error, in which an agent learns how to act from success and failure, reward and punishment, pleasure and pain. Almost all RL problems can be formulated in the theoretical framework of a MDP.

   RL is used to solve problems where the model is not known or the model is too complex or the model is too big. There are some differences to MDP. First, in RL the only inputs are experiences s_t, a_t, r_t for $t = 0, 1, ...$ Second, in RL no model of the environment is a-priory available, i.e., no model of the state transitions and rewards is given. Third, In some RL algorithms the model of the environment is learned from experiences. Fourth, In RL a reward is sampled without knowing the underlying model of the reward function. Fifth, In RL a state transitions is experienced without knowing the state transition probabilities.

   The following RL example is taken from towardsdatascience.com

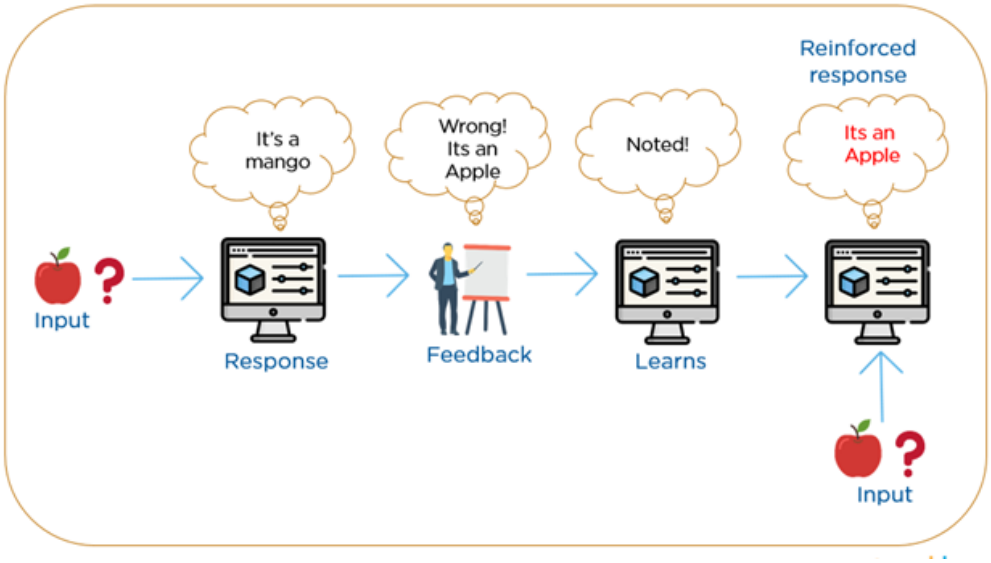

Figure 2: RL example

Homework 3

Part 2

1. The first part of the question is answered in the python code. for the second part:

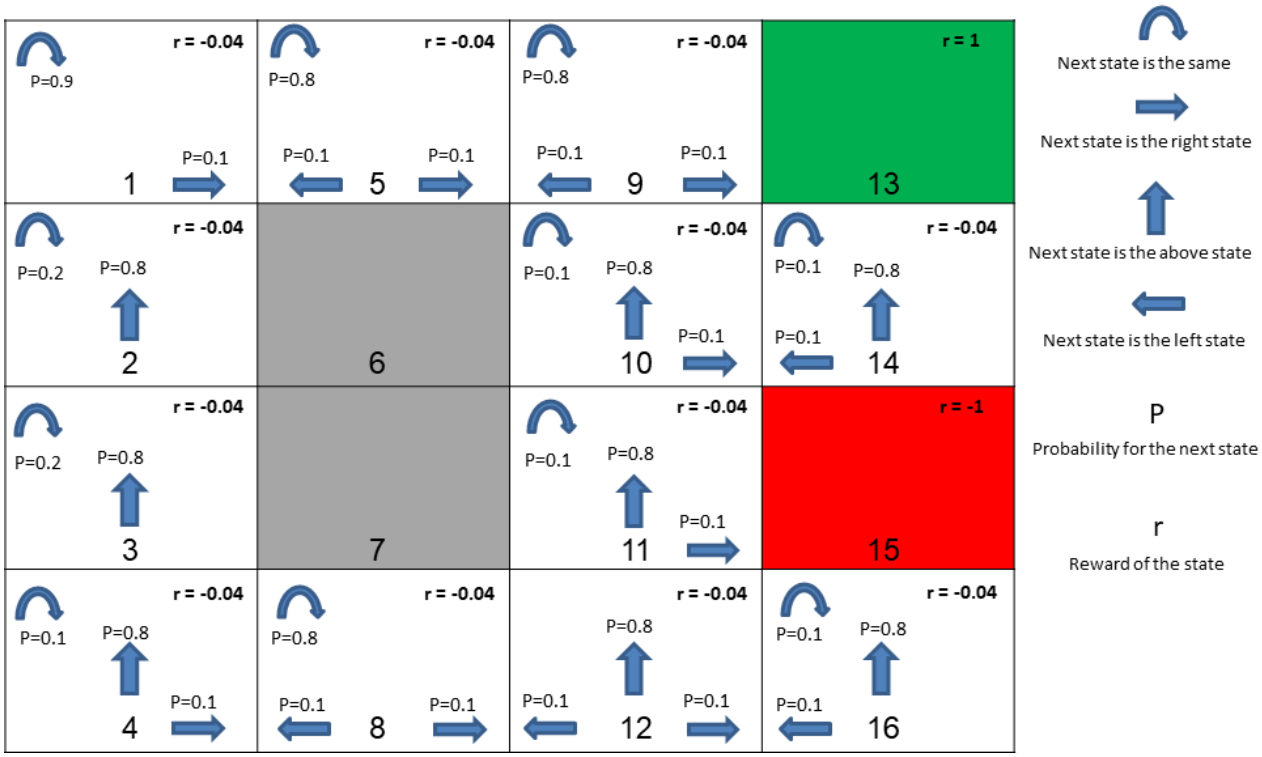

Figure 3: graphical model of the MDP for $a = N$.

2. For $\gamma = 1$ and $r = -0.04$, a value iteration has been executed. The result of the values:

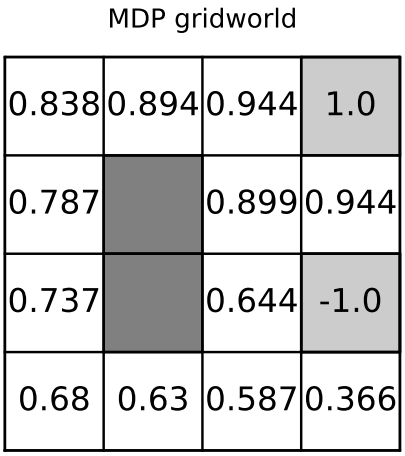

Figure 4: Optimal values question b.

The result of the policy:

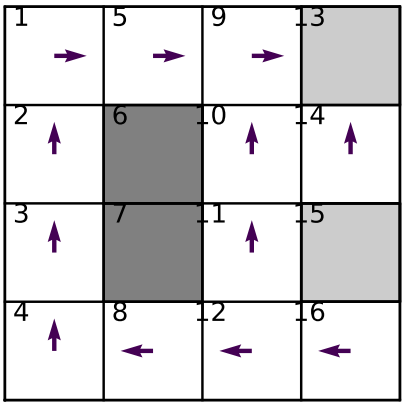

Figure 5: Optimal policy question b.

3. For $\gamma = 0.9$ and $r = -0.04$, a value iteration has been executed. The result of the values:

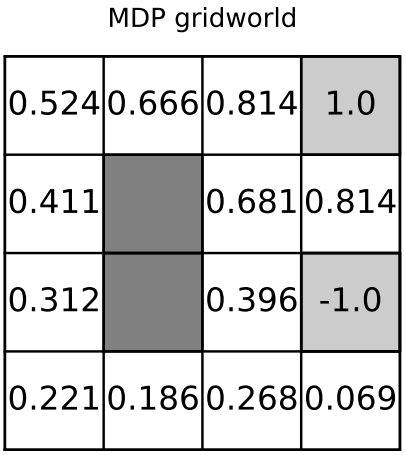

Figure 6: Optimal values question c.

The result of the policy:

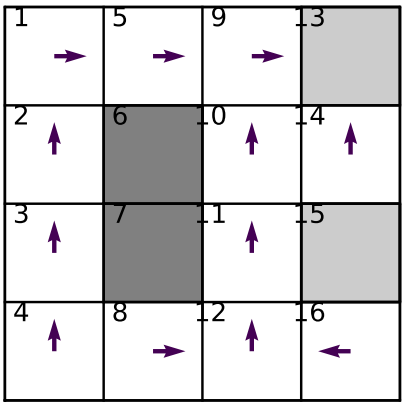

Figure 7: Optimal policy question c.

4. For $\gamma = 1$ and $r = -0.02$, a value iteration has been executed. The result of the values:

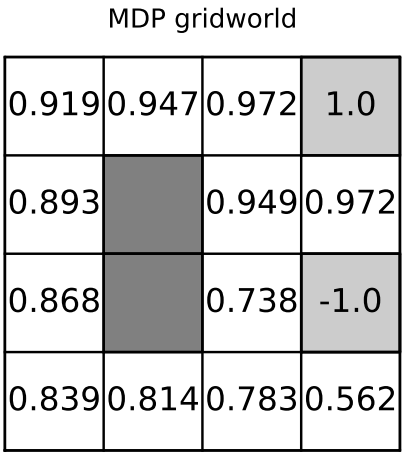

Figure 8: Optimal values question d.

The result of the policy:

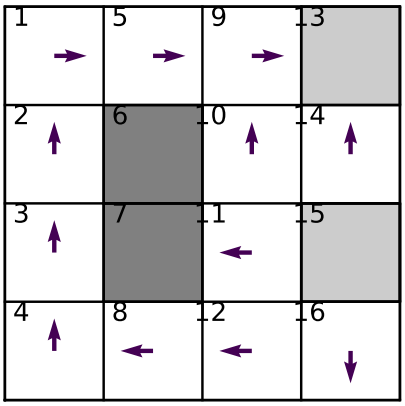

Figure 9: Optimal policy question d.

5. For $\gamma = 0.9$ and $r = -0.04$, a policy iteration has been executed. The initialized policy is changing so I will show only the first and the last optimal values and policy. The rest will be in the zip file I will attach with this work. The first results of the values:

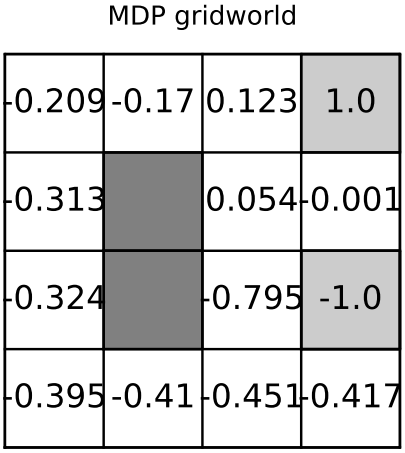

Figure 10: First values question e.

The first result of the policy:

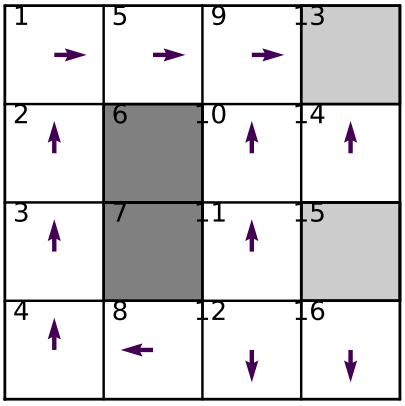

Figure 11: First policy question e.

The last results of the values:

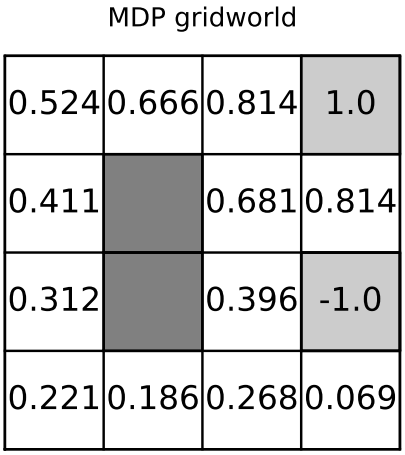

Figure 12: Last optimal values question e.

The last result of the policy:

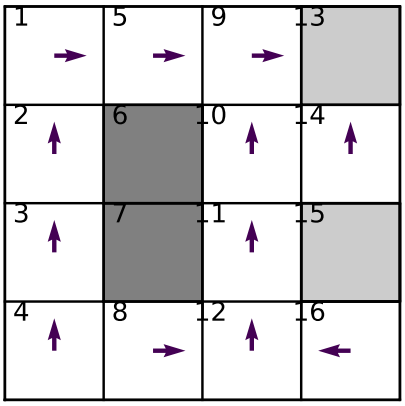

Figure 13: Last optimal policy question e.

We can see that we received the same policy as in question d. That mean that when $\gamma < 1$ the policy is more greedy in terms of the shortest path. While with $\gamma = 1$ the policy is greedy in terms of the safest path.