Ex. 3 – Markov Decision Process

Ori Aharon (ID: 200274504)

Nadav Lehrer (ID: 302170378)

1.1 **Markov decision processes** (**MDPs**) is an extension of Markov Chain (MC) and the difference between MC to MDP is the addition of actions (allowing choice) and rewards (giving motivation). MDP is useful for studying a wide range of optimization problems and provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. For example, we can use MDP in order to find the optimal action for Chess game or Damka.

1.2 **The State-Value Function** V(s) is the expected return an agent is to receive from being in state s behaving under a policy π(a|s). More specifically, the state-value is an expectation over the action-values under a policy: $V(s) = \sum_a \pi(a|s)Q(s,a)$.

1.3 **The Action-Aalue Function** Q(s,a) represents the expected return (cumulative discounted reward) an agent is to receive when taking action "a" in state "s", and behaving according to a policy π(a|s) afterwards (which is the probability of taking an action in a given state).

1.4 **The Policy** is the series of actions that the decision maker will use. The decision maker chooses the policy on purpose to maximize the expected total rewards.

1.5 **Dynamic programming** is a method refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. (for instance, solving problem from the end to the start).

1.6 **Value iteration** is a method of computing an optimal MDP policy and its value. Value iteration starts at the "end" and then works backward, refining an estimate of either Q* or V*. There is really no end, so it uses an arbitrary end point. Let Vk be the value function assuming there are k stages to go, and let Qk be the Q-function assuming there are k stages to go. These can be defined recursively. Value iteration starts with an arbitrary function V0 and uses the following equations to get the functions for k+1 stages to go from the functions for k stages to go:

$$Q_{k+1}(s,a) = \Sigma_{s'} P(s'|s,a)\,(R(s,a,s')+ \gamma V_k(s')) \quad \text{for } k \geq 0$$
$$V_k(s) = \max_a Q_k(s,a) \quad \text{for } k>0.$$

It can either save the V[S] array or the Q[S,A] array. Saving the V array results in less storage, but it is more difficult to determine an optimal action, and one more iteration is needed to determine which action results in the greatest value.

1.7 **The policy iteration** algorithm manipulates the policy directly, rather than finding it indirectly via the optimal value function. The value function of a policy is just the expected infinite discounted reward that will be gained, at each state, by executing that policy. It can be determined by solving a set of linear equations. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can, we change the policy to take the new action whenever it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible, then the policy is guaranteed to be optimal.

1.8 **Reinforcement Learning (RL)** is simply copied from nature. For instance, a child learns walking by trying. At first not much successful but after some time when it has learned, how to balance, how to use legs then unsuccessful attempts are reduced and walking becomes better and better. This process can be described as a search of choosing the right action in an appropriate situation, simply trial-and-error-search. If an action was successfully matched to a situation, it may be promising but to reach the goal this process must be kept on, i.e. the first step of a child may cause a „WOW", a reward.

In a Reinforcement Learning framework, the environment consists of a set of possible states S. Over the time the agent gets into certain states $s_t \in S$ per time step t. The agent as a learner and decision maker selects an action at from the set of possible actions $A(s_t)$ and executes it. Then the environment reacts to the action of the agent, i.e. the agent gets a reward r_{t+1} in the next time step also known as the reinforcement value. Rewards are numerical values. A_t the same time the agent passes to a new state s_{t+1} of the environment. In this scenario the agent's policy π_t is the process of mapping state representations to probabilities of selecting each possible action. The process of updating a policy to maximize the expected overall reinforcement is the general characteristic of a Reinforcement Learning Problem. $\pi(s,a)$ means the probability that a_t = a if s_t = s. As the probabilities are updated in each step to approximate an optimal value, the policy adapts as well.

The purpose of Reinforcement Learning (RL) is to solve a Markov Decision Process (MDP) when you don't know the MDP, in other words you don't know the states you can visit and you don't know the transition function from each state. So RL is a technique to learn an MDP and solve it for the optimal policy at the same time.

2.

a. In this section we were asked to solve a navigation problem given an agent who navigates in a grid world as such:

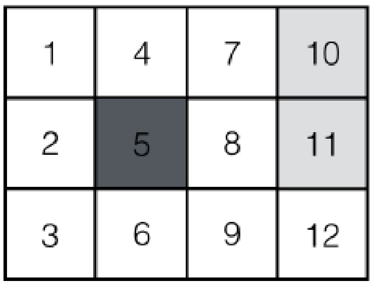

Figure 2.1: The grid world

In this world, each cell represents a state in which the agent can reside in, secluding state 5 which represents an obstacle.

To model this problem, as an MDP and eventually provide an optimal deterministic policy, we need to supply a reward model and a transition model.

Transition model

The model provided in the assignment, states that an action taken by the agent ($a \in A = \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ labeled also $A = \{N = 1, E = 2, S = 3, W = 4\}$), is performed with probability 0.8 and the probability that the action is affected by the environment resulting in a motion perpendicular to the intended action, is 0.2. Given the fact the modeled world is reflective (a fair model), and that state 5 is an obstacle, and states 11 and 10 are terminal states (or absorbing states) , we can now build the transition matrices.

Let us show as an example the model of the action a=Noth:

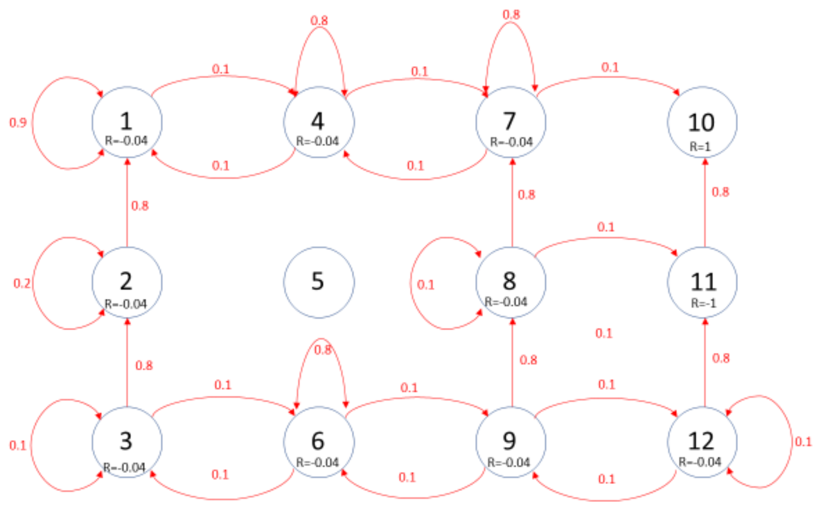

Figure 2.2: The transition model for an action a=North.

This model is manifested in the transition matrix $P_{ss'}^{a=N}$:

From state ↓	To state →	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
1		0.9	0	0	0.1	0	0	0	0	0	0	0	0
2		0.8	0.2	0	0	0	0	0	0	0	0	0	0
3		0	0.8	0.1	0	0	0.1	0	0	0	0	0	0
4		0.1	0	0	0.8	0	0	0.1	0	0	0	0	0
5		0	0	0	0	0	0	0	0	0	0	0	0
6		0	0	0.1	0	0	0.8	0	0	0.1	0	0	0
7		0	0	0	0.1	0	0	0.8	0	0	0.1	0	0
8		0	0	0	0	0	0	0.8	0.1	0	0	0.1	0
9		0	0	0	0	0	0.1	0	0.8	0	0	0	0.1
10		0	0	0	0	0	0	0	0	0	0	0	0
11		0	0	0	0	0	0	0	0	0	0	0	0
12		0	0	0	0	0	0	0	0	0.1	0	0.8	0.1

Reward function

As mentioned, to solve the problem with an MDP process, we need to provide a reward function. As seen in the exercise, the reward is strictly based on the state, and has no dependency on the action as in the general case. So, the modelling is made according to the data provided:

$$r(s) = \begin{cases} 1 & s = 10 \\ -1 & s = 11 \\ -0.04 & else \end{cases}$$

This reward model guaranties a trajectory that will eventually end at state 10 (as it is a terminal absorbing state) and will probably avoid state 11 (as it carries minimum reward). Moreover, assigning a negative reward to the other states, is a common way to assure that the trajectory will be the shortest, but as we will see assigning different values to the reward will result in a policy that is "willing" to take either more or less risk.

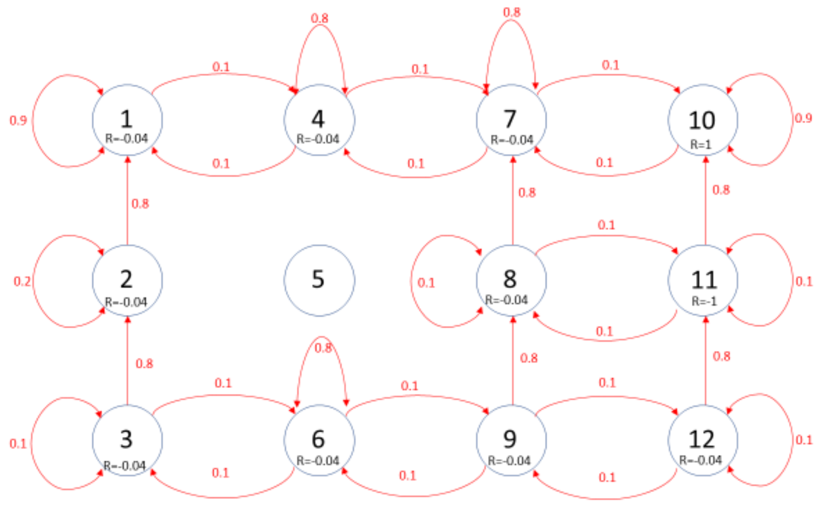

Figure 2.2: The transition model for an action a=North, with state rewards as given.

b. Value iterations

In this section we preformed value iterations, by utilizing the bellman optimality equation, iteratively:

$$v_{k+1}(s) = \max_{a\in\mathcal{A}} q_k^a(s) = \max_{a\in\mathcal{A}}\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

Or vector wise as seen in the code:

$$\boldsymbol{v}_{k+1} = \max_{a\in\mathcal{A}} \boldsymbol{q}_k^a = \max_{a\in\mathcal{A}}(\boldsymbol{\mathcal{R}}^a + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{a}} \boldsymbol{v}_{\boldsymbol{k}} \;)$$

In this method, we use the fact that the optimal State Value Function (**SVF**) in a given state, is guaranteed by a deterministic policy that achieves the maximum Action Value Function (**AVF**). The fact that the "max" operator is non-linear, forces us to solve this recursive relation, iteratively.

The way we utilized this relation to solve this MDP, is by computing 4 different values of AVF for each state, each one corresponding to an action:

$$\boldsymbol{q}_k^N(s),\ \boldsymbol{q}_k^E(s),\ \boldsymbol{q}_k^S(s),\ \boldsymbol{q}_k^W(s)$$

Or equivalently, with vectors:

$$\boldsymbol{\mathcal{R}}^N + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{N}} \boldsymbol{v}_{\boldsymbol{k}};\quad \boldsymbol{\mathcal{R}}^E + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{E}} \boldsymbol{v}_{\boldsymbol{k}};\quad \boldsymbol{\mathcal{R}}^S + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{S}} \boldsymbol{v}_{\boldsymbol{k}};\quad \boldsymbol{\mathcal{R}}^W + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{W}} \boldsymbol{v}_{\boldsymbol{k}}$$

Than after, by taking the maximum among these AVF of each state, we can get the maximum SVF (optimal SVF), and corresponding action, for each state, i.e.:

$$v_{K+1}^*(s) \;= \max_{a\in\{N,E,S,W\}} (q_{k+1}^N(s),\ q_{k+1}^E(s),\ q_{k+1}^S(s),\ q_{k+1}^W(s))$$

These values of SVF are calculated, until following iterative terms converge within a given tolerance. Than after, we can extract the deterministic policy as such:

$$\pi(s) = a(s) = \underset{a\in\{N,E,S,W\}}{\operatorname{argmax}} \left(q_{k+1}^{N}(s),\ q_{k+1}^{E}(s),\ q_{k+1}^{S}(s),\ q_{k+1}^{W}(s)\right)$$

First simulation $\gamma = 0.9$, r = -0.04

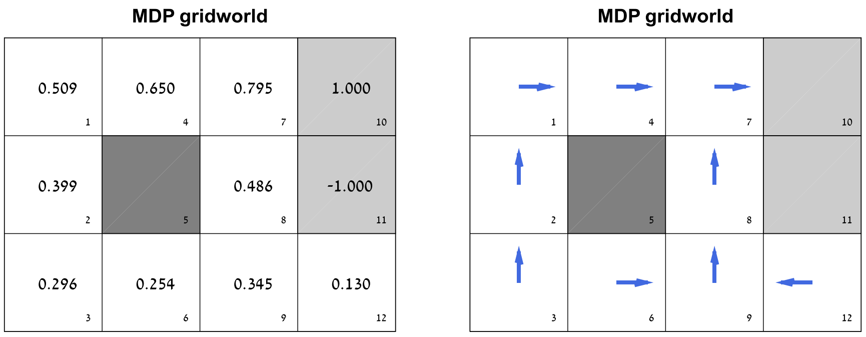

Figure 2.3: MDP simulation with $\gamma = 0.9, r = -0.04$. On the left are the maximal values obtained via Bellman optimality, iterative solution. On the right the corresponding action that gives that value for each state.

We can therefore see, that given a discount factor of 0.9, is translated as attributing the agent with more 'myopic' or 'short-sighted' characteristics, and results in the policy being less patient and more prone to collect immediate reward. Let us view the same MDP with discount factor of $\gamma = 1$, and see the agent is 'far-sighted':

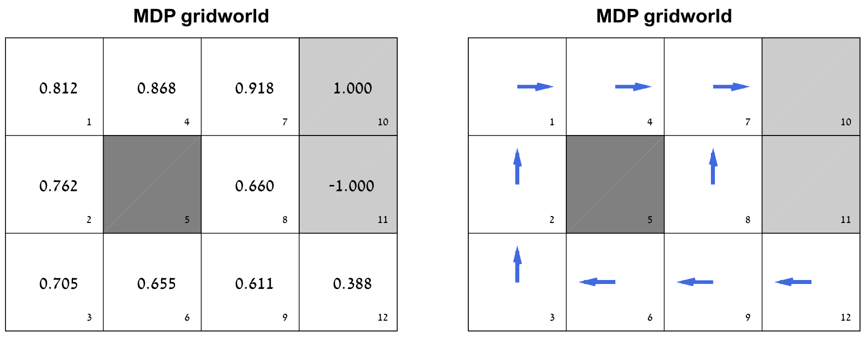

Figure 2.4: MDP simulation with $\gamma = 1, r = -0.04$. On the left are the maximal values obtained via Bellman optimality, iterative solution. On the right the corresponding action that gives that value for each state

So, we see that now the agent is more "patient" as it knows, that taking the longer route, will probably result in a larger accumulated value, as the probability of falling into state 11 with -1 penalty is higher when collecting immediate reward at state 8 for example.

Second simulation $\gamma = 1$, r = -0.02

this time, we ran the MDP with a reward of:

$$r(s) = \begin{cases} 1 & s = 10 \\ -1 & s = 11 \\ -0.02 & else \end{cases}$$

And a discount factor of 1, making the agent more "far-sighted" and less prone to take risks such as actions that might throw the agent to an unwanted state of minimum accumulated reward, or equivalently maximum penalty followed by the end of the process (state 11).

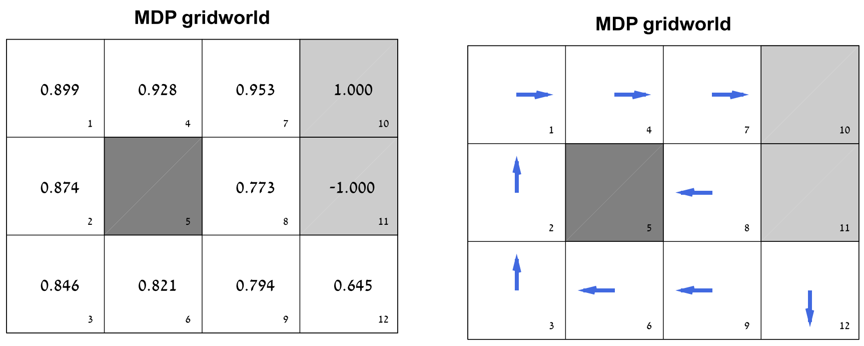

Figure 2.5: MDP simulation with $\gamma = 1, r = -0.02$.

As we can see in state 12, by setting the immediate state reward/penalty to closer to zero i.e. r=-0.02 instead of r=-0.04, and the discount factor to 1, the preferred action from this state is the only action that will have no chance of landing the agent in state 11 that carries a penalty of -1 and the end of the process, thus resulting in a South action that will either land the agent in the same 12'th state (with probability 0.9 of bouncing from the southern and eastern wall) or will result in a movement towards state 9. The same result and justification will apply to state 8.

Setting the state reward to r=-0.04, will therefor result in an agent willing to take more risk of landing in the pit of state 11 as it will want to avoid accumulating a penalty of -0.04 once it remains in the same state:

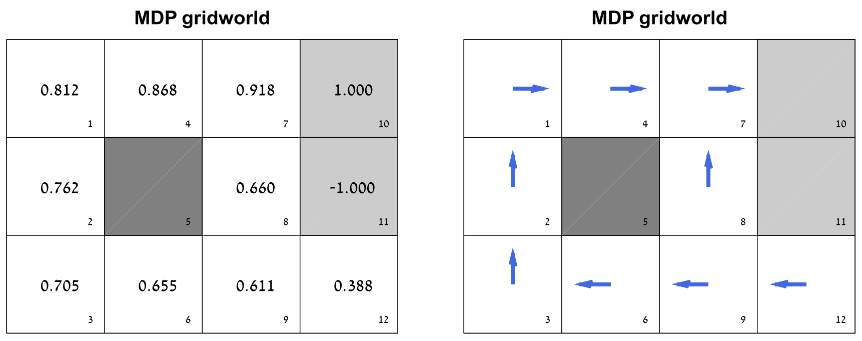

Figure 2.6: MDP simulation with $\gamma = 1, r = -0.04$.

So, we can conclude from this section that setting a lower state reward of r=-0.04, results in an agent that has less incentive to take an action that will probably land him in the same state again.

Moreover, setting the discount factor to 1 will result in a policy that is willing to take the risk, although it can carry a large penalty, in return of a probable immediate reward:

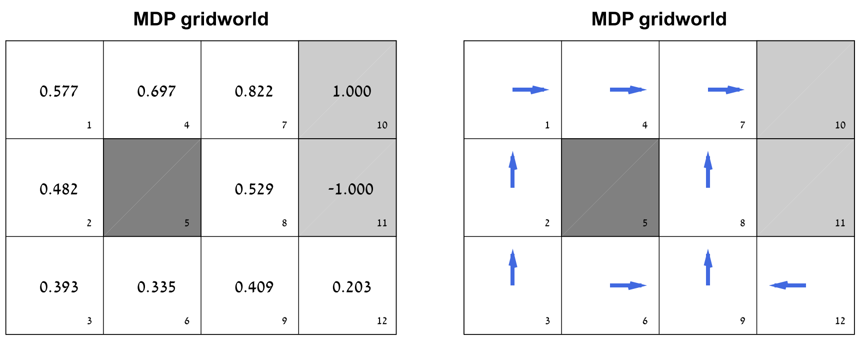

Figure 2.7: MDP simulation with $\gamma = 0.9, r = -0.02$.

c. Policy iteration

In this section we preformed the value iterations, by utilizing the bellman expectation equation, iteratively:

$$v^{\pi}{}_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k^{\pi}(s') \right)$$

By expanding the brackets, we can express $\mathcal{R}_s^a$ and $\mathcal{P}_{ss'}^a$ as the expected values for the state with respect to a policy $\pi(a|s)$:

$$\mathcal{R}_s^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a$$

$$\mathcal{P}_{ss'}^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a$$

thus obtaining:

$$v_{k+1}^{\pi}(s) = \mathcal{R}_s^{\pi} + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\pi} v_k^{\pi}(s')$$

Or vector wise as seen in the code:

$$\boldsymbol{v}_{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma \boldsymbol{\mathcal{P}}^{\pi} \boldsymbol{v_k}$$

As $\mathcal{R}_s^a = \mathcal{R}_s \neq f(a)$ i.e. the rewards are functions of the state only and are equal for all actions from a given state, we obtain:

$$\mathcal{R}_s^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}_s = \mathcal{R}_s \sum_{a \in \mathcal{A}} \pi(a|s) = \mathcal{R}_s$$

Which is the reward as given in the exercise:

$$r(s) = \begin{cases} 1 & s = 10 \\ -1 & s = 11 \\ -0.04 & else \end{cases}$$

Obtaining $\boldsymbol{\mathcal{P}}^{\pi}$ or $\mathcal{P}_{ss'}^{\pi}$ is a bit trickier, as we need to average the matrices according to the weighing (policy) $\pi(a|s)$.

For example, in some states the policy can be given as a deterministic one e.g. "take action 'North' with probability 1", or otherwise a uniformly distributed random policy that states "take all four actions N, E, S or W, with equal probability", So we can't simply average our four $\boldsymbol{\mathcal{P}}^{a}$ matrices because each state has a different policy distribution $\pi(a|s)$. We therefor take a closer look at the expression:

$$\mathcal{P}_{ss'}^{\pi} = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{P}_{ss'}^{a}$$

which means we need to average the rows of the four matrices, separately. Each row represents a transition model from a state, with corresponding $\pi(\cdot|s)$, so we average each matching row separately between all 4 matrices according to $\pi(a|s)$. Eventually we obtain 12 rows, each for a different state, and combine them to a stochastic mean transition matrix $\boldsymbol{\mathcal{P}}^{\pi}$.

After obtaining $\boldsymbol{\mathcal{P}}^{\pi}$ we ran the simulation with a random equally distributed policy for each state, as the initial policy:

$$\pi(a|s) = 0.25 \quad \forall a \in \mathcal{A} \ \ \forall s \in \mathcal{S}$$

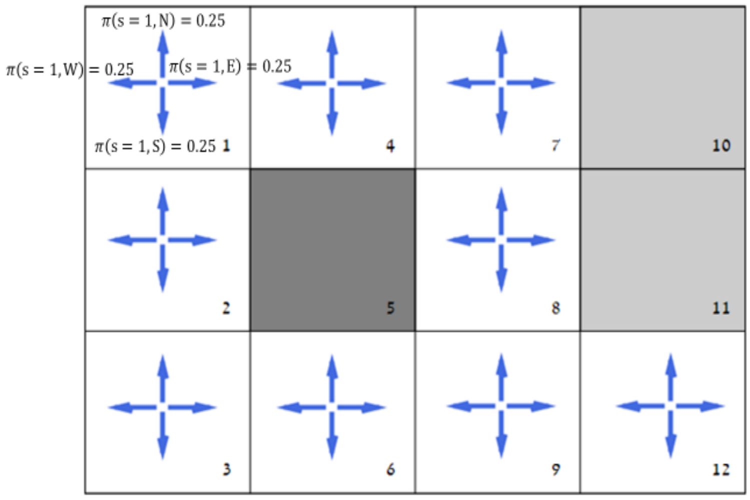

Figure 2.8: Graphical depiction of an equally distributed random policy at each state. i.e. the probability of every action at every state equals 0.25.

Then after, we obtained the state value function (**SVF)** of each state, given a policy, and re-defind the policy to the greedy one, i.e.:

$$\pi_{gredy}(a|s)$$

By giving probability to the actions which take the agent to the most rewarding adjacent cell, which can result in a policy as such:

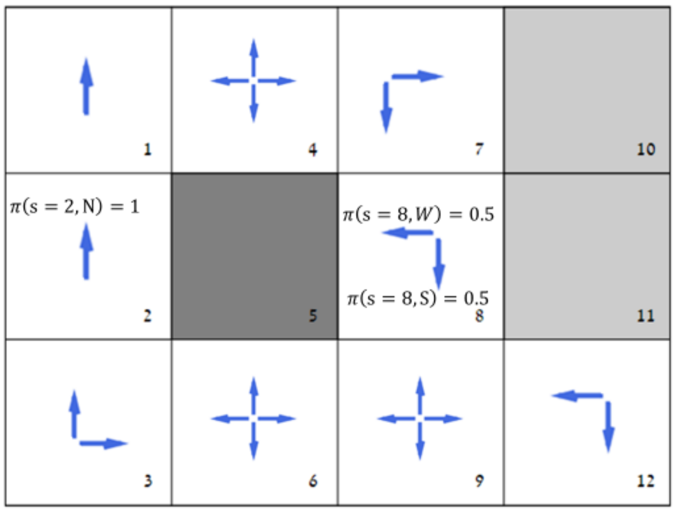

Figure 2.8: Graphical depiction of a random policy at some states, and deterministic one at others.

After reassigning policies, we calculate the mean transition matrix again by:

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a\in\mathcal{A}} \pi(a|s)\mathcal{P}^{a}_{ss'}$$

and calculate the matching SVF once more:

$$\boldsymbol{v}_{k+1} = \boldsymbol{\mathcal{R}}^{\pi} + \gamma\boldsymbol{\mathcal{P}}^{\pi}\boldsymbol{v}_{\boldsymbol{k}}$$

And take

$$\pi_{gredy}(a|s)$$

For each state.

We perform this scheme iteratively until convergence of two following SVF vectors, of

$$\pi^{k+1}_{gredy}(a|s) \;\; and \;\; \pi^{k}_{gredy}(a|s)$$

In the end we may still not get a deterministic policy, as the optimal (greedy) policy is not necessarily unique, so after convergence to the optimal policy we can simply choose a deterministic one from the optimal. In our case, the result of the optimal policy was already a deterministic one.

simulation $\gamma = 0.9$, r = -0.04

The results of this simulation were discussed in the previous section; however, it should be noted that the same result was yielded using the policy iteration method and the value iterations as we can see, Supporting the theory:

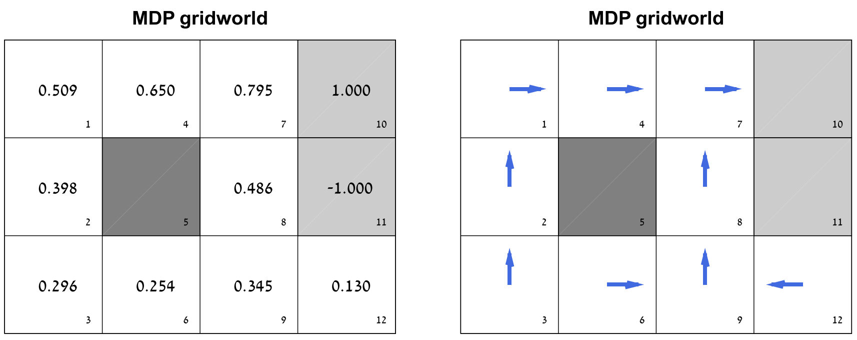

Figure 2.9: MDP simulation with $\gamma = 0.9, r = -0.04$.