**Ex. 3 – Markov Decision Process**

**I think This version is the correct one, the differences are in part a, where value iteration have different results**

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

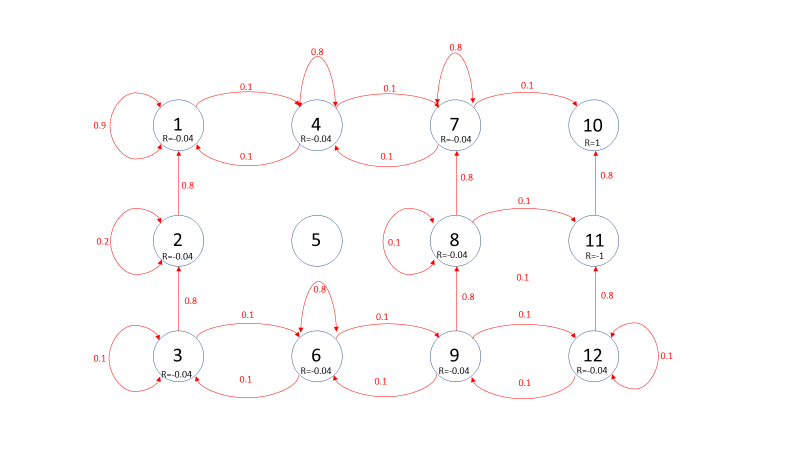
**Figure 2.1:** The grid world

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 labeled also ), 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=N:

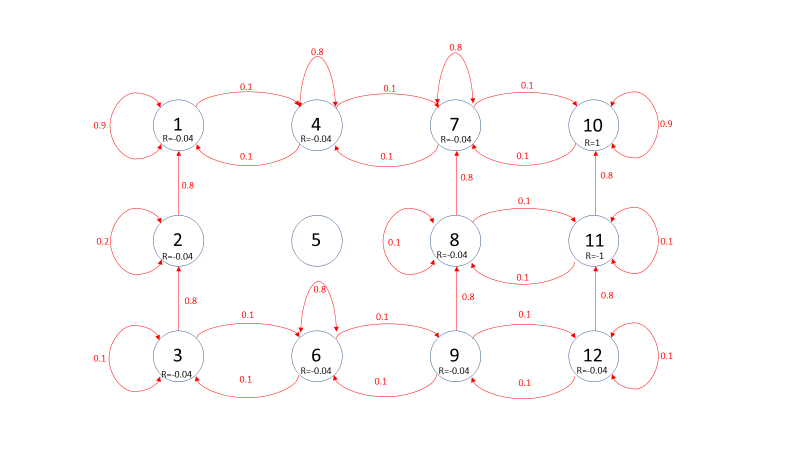
**Figure 2.2:** The transition model for an action a=North.

This model is manifested in the transition matrix :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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:

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 a 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.

* 1. **Value iterations**

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

Or vector wise as seen in the code:

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:

Or equivalently, with vectors:

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.:

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:

**First simulation , r = -0.04**



**Figure 2.3:** MDP simulation with . 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 , and see the agent is ‘far-sighted’:



**Figure 2.4:** MDP simulation with . 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 , r = -0.02**

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

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 .



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 .

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 .

* 1. **Policy iteration**

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

By expanding the brackets, we can express and as the expected values for the state with respect to a policy :

thus obtaining:

Or vector wise as seen in the code:

As i.e. the rewards are functions of the state only and are equal for all actions from a given state, we obtain:

Which is the reward as given in the exercise:

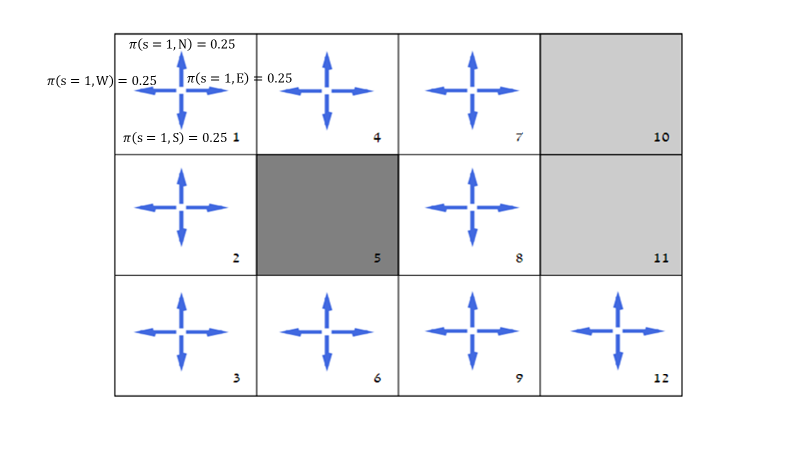
Obtaining or is a bit trickier, as we need to average the matrices according to the weighing (policy) .

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 matrices because each state has a different policy distribution. We therefor take a closer look at the expression:

which means we need to average the rows of the four matrices, separately. Each row represents a transition model from a state, with corresponding , so we average each matching row separately between all 4 matrices according to

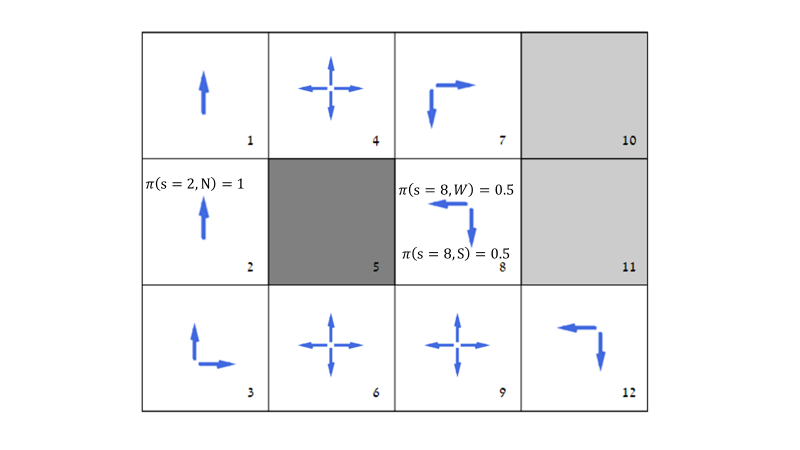
Eventually we obtain 12 rows, each for a different state, and combine them to a stochastic mean transition matrix .

After obtaining we ran the simulation with a random equally distributed policy for each state, as the initial policy:

**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.:

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:

and calculate the matching SVF once more:

And take

For each state.

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

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 , 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 .