COMP3702 Artificial Intelligence (Semester 2, 2022)
Assignment 2: HEXBOT MDP – **Report Template**

Name: Eskil Pedersen

Student ID: 47613722

Student email: e.pedersen@uqconnect.edu.au

Note: Please edit the name, student ID number and student email to reflect your identity and **do not modify the design or the layout in the assignment template**, including changing the paging.

Question 1 (Complete your full answer to Question 1 on the remainder of page 1)

a)

State space: Consists of all possible legal configurations of the robot positions and orientation as well as each widget's centre position and orientation. The state space is defined in line 27 in the code.

Action space: Consists of the four possible actions [forwards, backward, spin left, and spin right]. These can be found in constans.py, and the array ROBOT_ACTIONS

Transition function: Takes in an action, a state, and a next state, and gives the probability of ending up in the next state when starting from the state and performing the given action. T: $(S\ x\ A\ x\ S') -> P$, where S is the state, A is the given action, S' is the resulting state, and P is the probability of that happening. When calling self.stoch_action(a) I get different probabilities for the different combinations of actions that can occur when performing action a. I can then use the function apply_dynamics(state, action) to get the resulting next state when performing the original action a.

Reward function: Is a Function R: (s, a, s_next) -> r. Where r is the reward for being in a state s, performing an action a, and ending in the state s_next.

b)

The discount factor determines how the agent prefers rewards in the near future compared to rewards in the more distant future. The closer zero the discount factor is, the more the agent prefers rewards in the near future. On the other side, if the discount factor is closer to 1, the agent prefers rewards in the more distant future.

c)

Planning Horizon: **Indefinite**. The agent looks ahead a finite number of steps, to the goal state. However, this is not a predetermined number of steps, because the agent doesn't know how many actions are required to get to the goal state.

Sensing Uncertainty: **Fully observable**. The agent has access to the complete state of the environment at each point in time, and therefore it is fully observable.

Effect Uncertainty: **Stochastic**. The agent knows the probability distribution over the resulting states when performing an action from a given state.

Computational Limits: **Perfect rationally**. The agent finds the best thing to do at the current state, without considering its computational resources.

Learning: **Knowledge is given**. The designer of the agent gives the agent knowledge, and the agent doesn't learn from data or past experiences.

Question 2 (Complete your full answer to Question 2 on page 2)

a)

The value iteration is using synchronous updates where all states are updated in each iteration and therefore does not use asynchronous updates.

The policy evaluation in PI is using linear algebra to update the rewards for each state.

b)

Testcase	VI Time	VI Iterations	PI Time	PI Iterations
1	6.6 s	54	6.9 s	10
2	7.3 s	56	8.5 s	17
4	6.7 s	68	11.1 s	21

c)

I will first discuss the iterations of VI and PI. For all test cases, PI has significantly lower iterations compared to VI. It makes sense that PI has fewer iterations because the policies converge in fewer iterations than the values.

On the other hand, VI is faster than PI on all the test cases. This can be explained by a couple of reasons, but PI is considered a faster implementation. PI should be quicker than VI because:

- The policies converge in fewer iterations than the values.
- We are solving linear algebra equations to update the values, instead of having to compute the values one by one, until convergence is achieved. Which is assumed to be much faster.
- We are using a max operator in VI to find the best new action, and this is a slow operation.

Due to the reasons above, PI should be faster. However, the faster computational times from VI can be explained by one of or a combination of the reasons below:

- The initialization of PI is slower than the initialization of VI. There is nothing in the initialize of VI, all the required variables are declared in _init_. However, in pi_initialise the t-model and r-model are computed, which loops through all combinations of legal states and robot actions. From my test, it seems like pi_initialise takes around 3 seconds.
- The policy update in PI is as slow as a value iteration pass.
- The small number of iterations doesn't show the computational differences between PI and VI, at least not when the initialization of PI takes such a long time.

Question 3 (Complete your full answer to Question 3 on page 3)

a)

I think the optimal path will change to the less risky path when the hazard penalty increases, and I think it's more likely that the robot takes the riskier path when the hazard penalty decreases.

In policy evaluation, the values are updated by the probability of ending up in a state s' from a state s, with the movement given by pi(s), times the sum of the reward for getting to state s' and the discounted value already stores for s'. When increasing the hazard penalty, the computed value will be more negative, due to the reward for colliding with a hazard being much higher.

$$V^{\pi_i}(s) = \sum_{s' \in \mathcal{S}} P(s' \mid \pi_i(s), s) \left[R(s, \pi_i(s), s') + \gamma V^{\pi_i}(s') \right]$$

Therefore, it's less likely to choose an action that makes the robot go towards the hazards, and therefore will the robot choose to go the less risky path.

When the transition probabilities increase, I think the optimal path will be less risky. Since there is a higher probability of crashing with a hazard.

b)

Number	Hazard	Probabilities	Resulting path
1	10.0	Same as original values	Risky
2	100.0	Same as original values	Less risky
3	30.0	Same as original values	Risky
4	10.0	Double the original values	Risky
5	100.0	Double the original values	Less risky
6	30.0	Double the original values	Less risky
7	10.0	Half the original values	Risky
8	100.0	Half the original values	Less risky
9	30.0	Half the original values	Risky

The three first results were mostly what I expected. The limit for when the robot chooses the riskier path, is somewhere between 30 and 100, given the same transition probabilities.

For cases 4, 5, and 6 I would expect the robot to go the less risky path, but the robot still chose the riskier path when the hazard had a reward of 10. This can be explained by the transition probabilities only being doubled, while the increase of the hazard from 10 to 30 is a multiplication by 3.

For cases 7,8, and 9 would I expect the robot to go up, but when the hazard was set to 100, the robot still took the less risky path. This can probably be explained by the transition probabilities only being halved, but the hazard was multiplied by 10 from the original data.