COMP3702 Artificial Intelligence (Semester 2, 2022)

Assignment 2: HEXBOT MDP

Name: Peter Tran

Student ID: s4585782

Student email: peter.k.tran2@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)

|  |  |
| --- | --- |
| **Components** | **Definition** |
| State Space | The state space is the possible combinations of the robot positions, robot orientation, widget positions, and widget orientation. A state in the game is represented as the class ‘State’ and is an instance that holds the current goal state, environment information, target location, hazard locations, robot, and widgets. The whole state space is represented as ‘self.states’ in the code and is generated via a BFS traversal from all the possible states. |
| Action Space | The action space is the set of all possible actions the agent can perform. These are ‘FORWARD, ‘REVERSE’, ‘SPIN\_RIGHT’ and ‘SPIN\_LEFT’. In the code, this is represented as the constant ROBOT\_ACTIONS. |
| Transition Function | The transition function is the outcome of a state given some initial state s and an action. There are 6 possible results that can occur from a nominal action: ‘NORMAL’, ‘NORMAL, NORMAL’, ‘NORMAL, DRIFT\_CW’, ‘NORMAL, DRIFT\_CCW’, ‘NORMAL, NORMAL, DRIFT\_CW’, ‘NORMAL, NORMAL, DRIFT\_CCW’. Therefore, the transition function will convert an action and state to one of these possible results to happen. This depends on the probabilities given from the environment read from the testcases. In the code, this behaviour can be seen in the apply\_action\_noise(). The probabilities of these actions are seen in the stoch\_action() function. |
| Reward Function | Reward function takes in the state and actions and outputs a value calculated from the action cost plus the expected value of any collision penalties. In the code, this is done through the function apply\_dynamics as it returns the reward. |

1. The purpose of the discount factor in MDPs is to allow the agent to preference whether it would like the rewards sooner rather than later. So basically, it decides on how much weight to give to future rewards in the value function.

|  |  |
| --- | --- |
| **Dimensions** | **Definition** |
| Planning Horizon | Indefinite |
| Sensing Uncertainty | Fully observable |
| Effect Uncertainty | Perfect rationality |
| Computational Limits | Knowledge is given |
| Learning | Fully observable |

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

1. Value iteration was implemented with non-asynchronous updates, however with the use of in-place updates to update each state once per iteration.
2. Policy iteration was implemented using linear algebra with numpy library, instead of the naïve approach and the reward model is dependent on only the state and action.
3. I will pick testcase 1,2,4 to compare. Vi and Pi have been run by modifying the testcase files to run the correct algorithm. Testcases 2 and 3 are very similar so they were not tested. 3 trials of each test run were conducted to minimalize random errors and noise.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Testcases | 1 | | 2 | | 4 | |
|  | Time to converge | # of iterations | Time to converge | # of iterations | Time to converge | # of iterations |
| Value Iteration | 30.96s | 100 | 56.19s | 127 | 67.52s | 214 |
| Policy Iteration | 5.82s | 10 | 11.87s | 15 | 8.83s | 19 |
| Ratio (VI/PI) | 5.32 | 10 | 4.73 | 8.46 | 7.64 | 11.26 |

1. Differences between VI and PI

As can be seen, For PI, it requires significantly less number of iterations than VI to converge. On average, PI is converging 5.9 times faster than VI.

Additionally, as a result, the time to converge is also much lower. This is also helped by the fact that for PI, linear algebra can be used to greatly speed on the computational time. On average, the number of iterations was 9.9 larger for VI than it was for PI.

These values make sense as policy iteration is cheaper to compute and requires fewer iterations to converge than value iteration. This in turn, will make policy iteration faster than value iteration. Policy iteration updates the policy every iteration instead of value iteration which iterates over the value function instead.

In these testcase results, there are no differences that do not make sense.

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

1. If the hazard penalty is low, I predict for the robot to attempt to go through the riskier path as it will have less to lose and and is not expensive if it does happen to enter the hazard zone. This riskier path is shorter so even with the hazard penalty, it may still be overall, less expensive. This is because the reward function takes in the state and action and calculates using the penalty value is expected reward for moving to that state.

Vice versa, If the hazard penalty is increased and high, I predict that the opposite will occur, and the robot will choose to go down to the safer path. This is because if the hazard penalty is too high, then the risks outweigh the benefits and the robot may end up in a more lower reward state if it does collide with the hazard.

For the transition probabilities, if these were high for the chances of drift and double, then I predict the robot would also choose to go to toward the lower and safer route as the chances for unintentional movement is much greater, and consequently, will result in a lower reward value. However, if the chance for drift is lower, while the chances of double is high, the robot may decide to go the shorter route as I predict that doubles may actual help the robot reach the goal state faster.

1. For the current given testcase 6, the robot choses the riskier route. The results will be based on the benchmark which the just the default testcase 6.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Combination** | **Double** | **CW** | **CCW** | **Hazard** | **Results** |
| **Benchmark (given)** | 0.25,0.1,0,0 | 0.05,0.025,0,0 | 0.05,0.025,0,0 | 10 | Riskier path |
| **High Hazard (50)** | 0.25,0.1,0,0 | 0.05,0.025,0,0 | 0.05,0.025,0,0 | 50 | Riskier path |
| **Higher Hazard (100)** | 0,0,0,0 | 0,0,0,0 | 0,0,0,0 | 100 | Safer path |
| **Very High Hazard (1000)** | 0,0,0,0 | 0,0,0,0 | 0,0,0,0 | 1000 | Failed to reach goal |
| **Low Hazard** | 0.25,0.1,0,0 | 0.05,0.025,0,0 | 0.05,0.025,0,0 | 2 | Riskier path |
| **No hazard** | 0.25,0.1,0,0 | 0.05,0.025,0,0 | 0.05,0.025,0,0 | 0 | Riskier path |
| **Increase all transition probabilities** | 0.5,0.2,0,0 | 0.1,0.05,0,0 | 0.1,0.05,0,0 | 10 | Riskier path |
| **Higher Increase all transition probabilities** | 0.6,0.3,0,0 | 0.2,0.1,0,0 | 0.2,0.1,0,0 | 10 | Safer path |
| **Very High Increase in transition probabilities** | 0.8,0.8,0,0 | 0.6,0.6,0,0 | 0.6,0.6,0,0 | 10 | Failed to reach goal |
| **Decrease all transition probabilities** | 0.125,0.05,0,0 | 0.025,0.0125,0,0 | 0.025,0.0125,0,0 | 10 | Riskier path |
| **High chance of transition and low hazard** | 0.5,0.2,0,0 | 0.1,0.05,0,0 | 0.1,0.05,0,0 | 2 | Riskier path |
| **High chance of transition and high hazard** | 0.5,0.2,0,0 | 0.1,0.05,0,0 | 0.1,0.05,0,0 | 50 | Safer path |

Overall, the results of the combinations were expected, however I was expecting the robot to be less ‘tolerant’ as it required changing the values significantly for the robot to choose the safer path over the riskier path.

This can be seen as using a hazard penalty of 50 did not make the robot choose the safer option but changing to 100 did work. Unexpectedly, changing the hazard penalty to 1000, causes the robot to fail reaching the goal state.

Same can be said for the transition probabilities. Doubling the probabilities did not affect the robot’s path choice however significantly increasing the chances does cause the robot to take the safer path. Further setting the transition probabilities to very high numbers cause the robot to not converge (or at least maybe I did not run the test for long enough).