Assignment 2

Ujjwal Sharma
23M0837

Question 1

Part 1 - Howards Policy Iteration

The algorithm is implemented in the `howardsPolicyIteration.py` file. Value evaluation is performed using the value iteration algorithm, which is defined in the `valueIterationSolver` function. The Q-Value calculation is handled by the `getQValue` function

Algorithm

```
1. $policy \leftarrow randomly\ initialized,\ value \leftarrow valueIterationSolver(policy),$
$QValues \leftarrow getQValue(states)$
2. $While\ policy\ is\ not\ stable$
3. $Get\ the\ current\ value\ for\ the\ policy\ and\ identify\ the\ improvable\ states\ for\ every\ state$
4. $If\ any\ state\ has\ an\ improvable\ state\ the\ update\ the\ policy\ by\ selecting\ the\ action\ which\ has\ the\ maximum\ Q - Value\ (tie\ breaks\ at\ random)$
5. $Check\ if\ policy\ is\ stable$
```

Implementation in `howardsPolicyIteration`

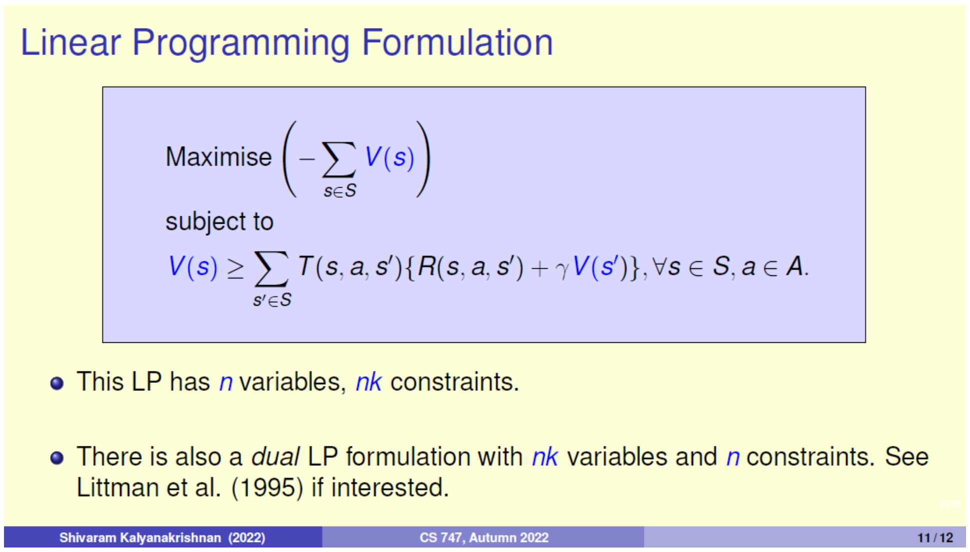
Part 2 - Linear Programming

The algorithm is implemented in the `linearProgrammingSolver.py` file. The logic for handling linear programming (LP) constraints is encapsulated in the `linearProgrammingSolver` function. This function, in turn, calls `getPolicyFromValueStar` to derive the optimal policy based on the optimal state values.

Optimizing LP Code.

The algorithm takes a sparse transition matrix and converts it into a dense representation by retaining only the states that are reachable through a given action. This optimization significantly improves the performance of the code.

Algorithm



Linear Programming Formulation

Maximise $\left(-\sum_{s \in S} V(s)\right)$

subject to

$$V(s) \geq \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V(s')\}, \forall s \in S, a \in A.$$

- This LP has n variables, nk constraints.
- There is also a *dual* LP formulation with nk variables and n constraints. See Littman et al. (1995) if interested.

Shivaram Kalyanakrishnan (2022) CS 747, Autumn 2022 11 / 12

Question 2

Encoder

The encoder takes in the grid world and encodes it into a set of valid states. By using these valid states and the actions permissible on that valid state, the encoder produces the transitions to other valid states.

Valid States Construction

For each cell in the grid world that is not a wall, we create eight possible valid states: four with the key and four without the key. The exception is the door cell, which only has four valid states, all of which are with the key.

Each valid state is represented as a tuple `(row, column, direction, key)`, where `key` is a binary value [0,1], indicating whether the key is present, and `direction` is one of the four possible directions: ["^", ">", "v", "<"].

The `generateStatesFromGridWorld` function is responsible for implementing the process of generating these valid states as described above.

Transitions

The goal is to reach the goal state after collecting the key in the fewest steps possible. To achieve this, three types of transitions are defined:

1. **Goal State Transition**: If the next state is a goal state, a large reward of 10,000 is assigned. The transition, along with the reward and probability, is then printed.

2. **Key State Transition**: If the next state contains a key, a smaller reward of 100 is given. This encourages the agent to prioritize reaching the key state. The transition is printed with the corresponding reward and probability.

3. **Other States Transition**: For all other states, a penalty of -1 is assigned (referred to as the "alive penalty"). This discourages the agent from revisiting the same states. The transition is printed with the corresponding reward and probability.

Important Corner Case

When the current state is the key state, the agent must collect the key. At this point, the agent is restricted to only move to valid states where the key is present.

The transitions are defined in the `generateTransitions` function in the `encoder.py` file.

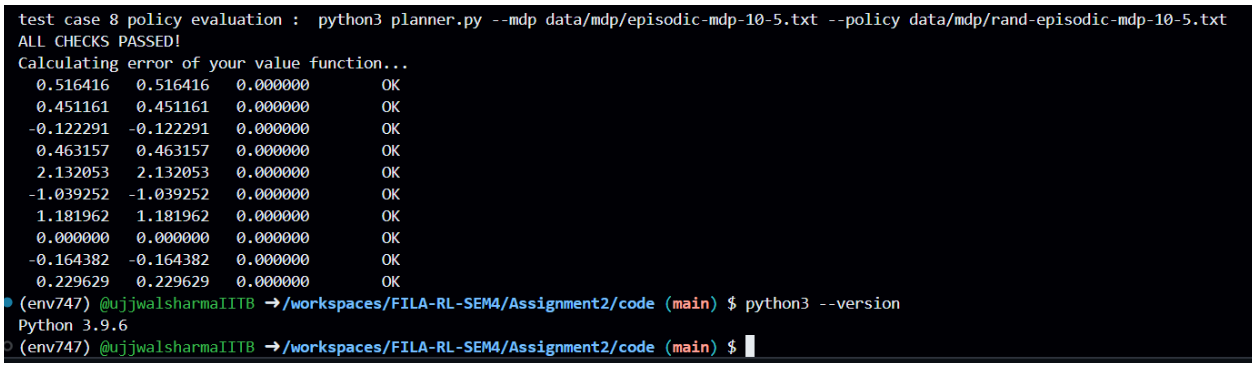
Decoder

The decoder is invoked after the encoder encodes the state and the planner generates the optimal value-action pair. The objective is to print the optimal action for a given snapshot of the grid world. Using the state encoder from `encoder.py`, I encode the grid world and identify the current state, which is represented as a tuple (row, column, direction, key). Then, based on the value-action pair generated by the planner, I print the optimal action for that state. The function `getThePathForThePolicyAndValue` in `decoder.py` implements this process.

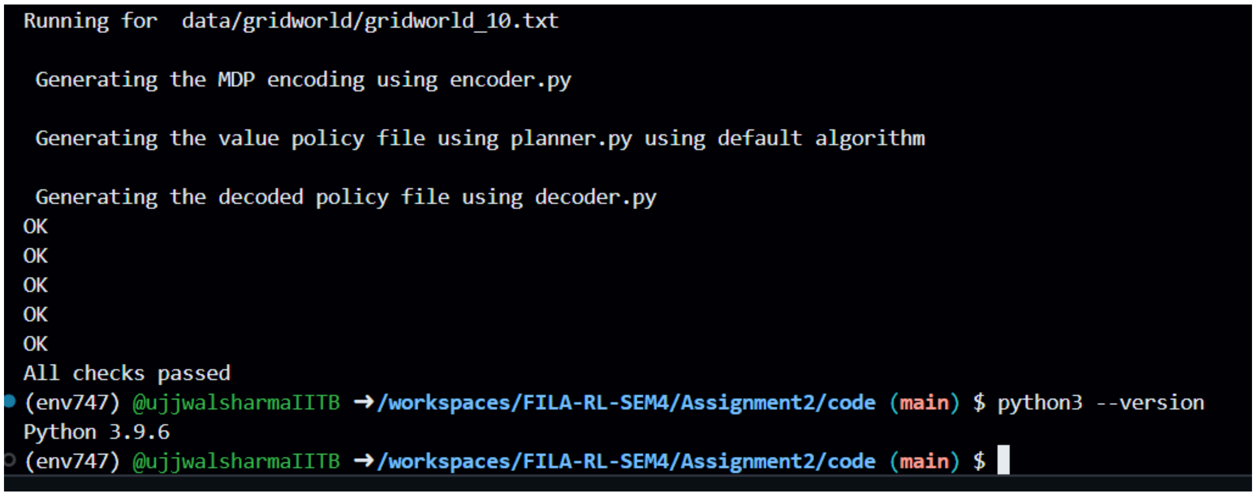
Appendix

All my test cases are passing and I have tested my code in both Windows and Linux

Linux



```
test case 8 policy evaluation :  python3 planner.py --mdp data/mdp/episodic-mdp-10-5.txt --policy data/mdp/rand-episodic-mdp-10-5.txt
ALL CHECKS PASSED!
Calculating error of your value function...
  0.516416   0.516416   0.000000        OK
  0.451161   0.451161   0.000000        OK
 -0.122291  -0.122291   0.000000        OK
  0.463157   0.463157   0.000000        OK
  2.132053   2.132053   0.000000        OK
 -1.039252  -1.039252   0.000000        OK
  1.181962   1.181962   0.000000        OK
  0.000000   0.000000   0.000000        OK
 -0.164382  -0.164382   0.000000        OK
  0.229629   0.229629   0.000000        OK
(env747) @ujjwalsharmaIITB ➜/workspaces/FILA-RL-SEM4/Assignment2/code (main) $ python3 --version
Python 3.9.6
(env747) @ujjwalsharmaIITB ➜/workspaces/FILA-RL-SEM4/Assignment2/code (main) $
```



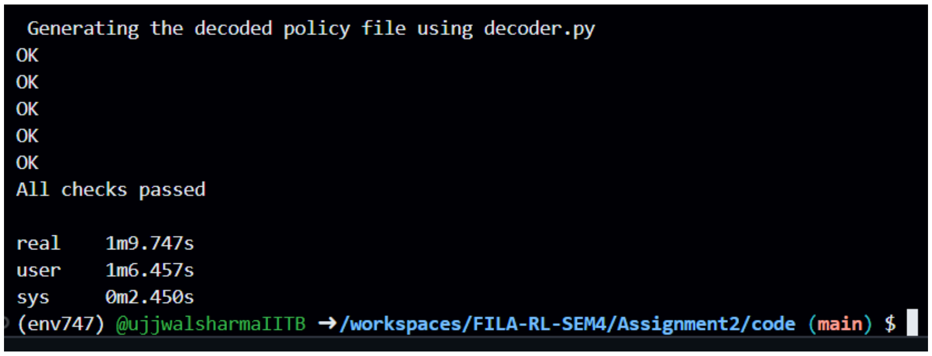
```
Running for  data/gridworld/gridworld_10.txt

 Generating the MDP encoding using encoder.py

 Generating the value policy file using planner.py using default algorithm

 Generating the decoded policy file using decoder.py
OK
OK
OK
OK
OK
All checks passed
(env747) @ujjwalsharmaIITB ➜/workspaces/FILA-RL-SEM4/Assignment2/code (main) $ python3 --version
Python 3.9.6
(env747) @ujjwalsharmaIITB ➜/workspaces/FILA-RL-SEM4/Assignment2/code (main) $
```

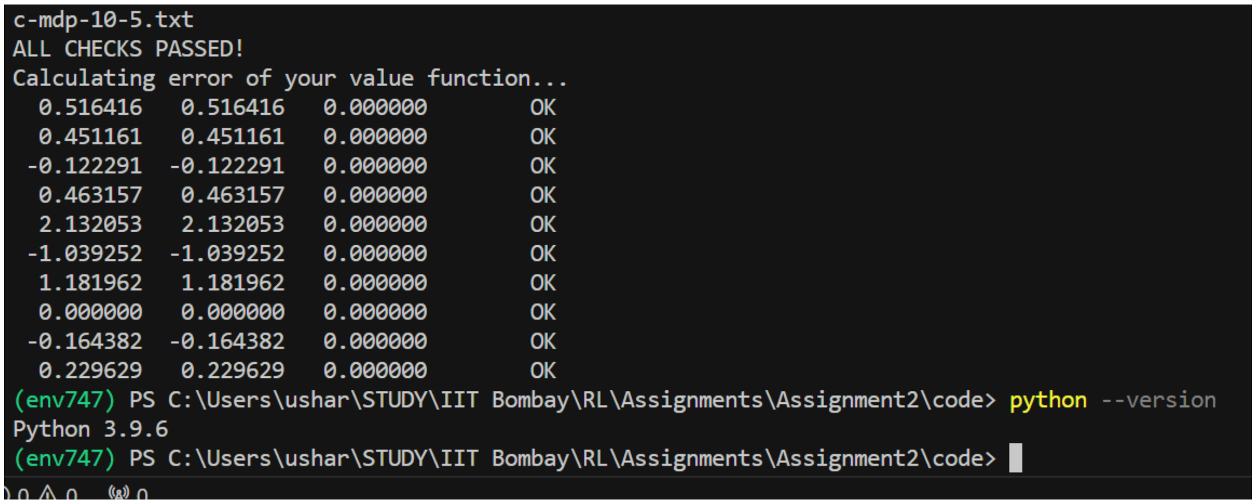
All tasks combined in Task2 completed in almost a minute



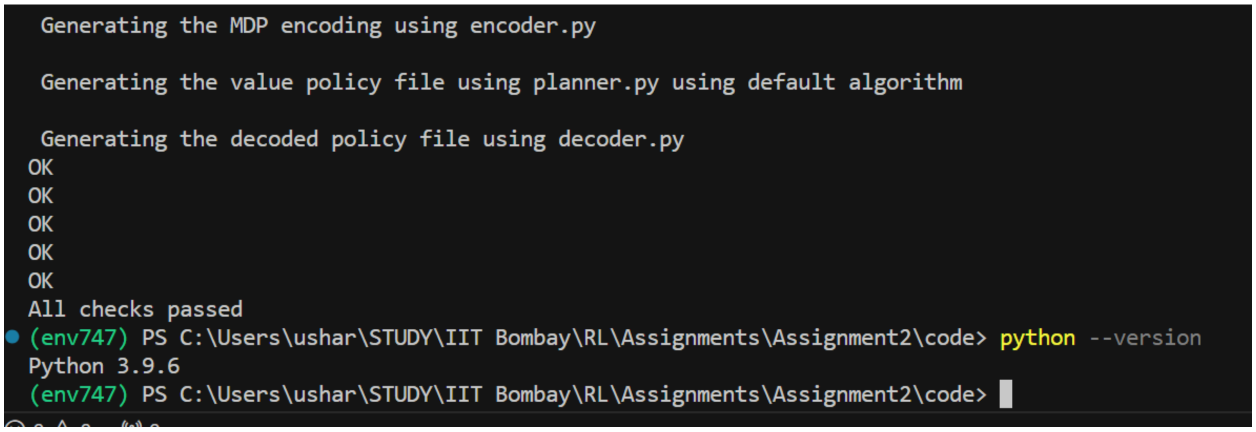
```
 Generating the decoded policy file using decoder.py
OK
OK
OK
OK
OK
All checks passed

real    1m9.747s
user    1m6.457s
sys     0m2.450s
(env747) @ujjwalsharmaIITB ➜/workspaces/FILA-RL-SEM4/Assignment2/code (main) $
```

Windows



```
c-mdp-10-5.txt
ALL CHECKS PASSED!
Calculating error of your value function...
  0.516416   0.516416   0.000000        OK
  0.451161   0.451161   0.000000        OK
 -0.122291  -0.122291   0.000000        OK
  0.463157   0.463157   0.000000        OK
  2.132053   2.132053   0.000000        OK
 -1.039252  -1.039252   0.000000        OK
  1.181962   1.181962   0.000000        OK
  0.000000   0.000000   0.000000        OK
 -0.164382  -0.164382   0.000000        OK
  0.229629   0.229629   0.000000        OK
(env747) PS C:\Users\ushar\STUDY\IIT Bombay\RL\Assignments\Assignment2\code> python --version
Python 3.9.6
(env747) PS C:\Users\ushar\STUDY\IIT Bombay\RL\Assignments\Assignment2\code>
```



```
   Generating the MDP encoding using encoder.py

   Generating the value policy file using planner.py using default algorithm

   Generating the decoded policy file using decoder.py
 OK
 OK
 OK
 OK
 OK
 All checks passed
 (env747) PS C:\Users\ushar\STUDY\IIT Bombay\RL\Assignments\Assignment2\code> python --version
 Python 3.9.6
 (env747) PS C:\Users\ushar\STUDY\IIT Bombay\RL\Assignments\Assignment2\code>
```

All the code files can be accessed using github
Link: https://github.com/ujjwalsharmaIITB/FILA-RL-SEM4/tree/main/Assignment2/code