COMP3702 Artificial Intelligence (Semester 2, 2022)
Assignment 2: HexBot MDP

Key information:

- **Due: 4pm, Friday 30 September**
- This assignment will assess your skills in developing algorithms for solving MDPs.
- Assignment 2 contributes 20% to your final grade.
- This assignment consists of two parts: (1) programming and (2) a report.
- This is an individual assignment.
- Both code and report are to be submitted via Gradescope (`https://www.gradescope.com/`). You can find instructions on how to register for the COMP3702 Gradescope site on Blackboard.
- Your program (Part 1, 60/100) will be graded using the Gradescope code autograder, using testcases similar to those in the support code provided at `https://gitlab.com/3702-2022/a2-support`.
- Your report (Part 2, 40/100) should fit the template provided, be in .pdf format and named according to the format `a2-COMP3702-[SID].pdf`, where SID is your student ID. Reports will be graded by the teaching team.

The HexBot Robot AI Environment

You have been tasked with developing a planning algorithm for automatically controlling HexBot, a multi-purpose robot which operates in a hexagonal environment, and has the capability to push, pull and rotate 'Widgets' in order to reposition them to target locations. To aid you in this task, we have provided support code for the HexBot robot environment which you will interface with to develop your solution. To optimally solve a level, your AI agent must efficiently find a sequence of actions so that every Target cell is occupied by part of a Widget, while incurring the minimum possible action cost.

For A2, the HexGrid environment has been extended to model non-deterministic outcomes of actions. Cost and action validity are now replaced by a reward function where action costs are represented by negative received rewards, with additional penalties (i.e. negative rewards) being incurred when a collision occurs (between the robot or a widget and an obstacle, or between widgets). Updates to the game environment are indicated in pink font.

Levels in HexBot are composed of a Hexagonal grid of cells, where each cell contains a character representing the cell type. An example game level is shown in Figure 1.

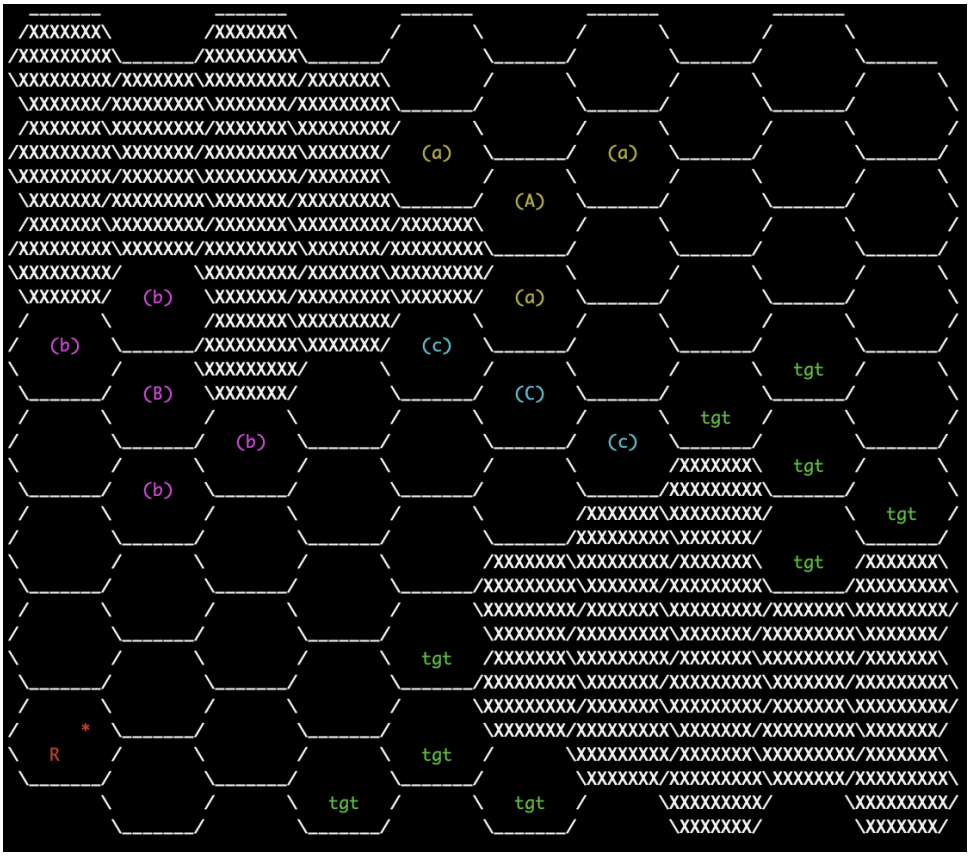

Figure 1: Example game level of HexBot

Environment representation

Hexagonal Grid

The environment is represented by a hexagonal grid. Each cell of the hex grid is indexed by (row, column) coordinates. The hex grid is indexed top to bottom, left to right. That is, the top left corner has coordinates $(0, 0)$ and the bottom right corner has coordinates $(n_{rows} - 1, n_{cols} - 1)$. Even numbered columns (starting from zero) are in the top half of the row, and odd numbered columns are in the bottom half of the row. An example is shown in Figure 2.

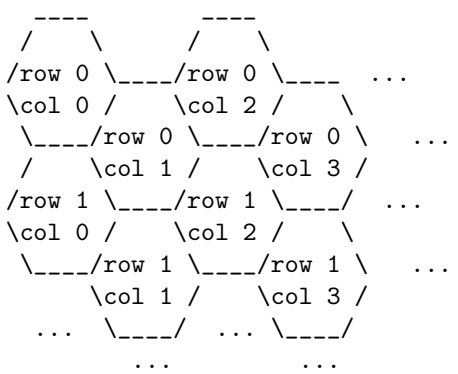

Figure 2: Example hexagonal grid showing the order that rows and columns are indexed

Two cells in the hex grid are considered adjacent if they share an edge. For each non-border cell, there are 6 adjacent cells.

Robot and its Actions

The HexBot robot occupies a single cell in the hex grid. In the visualisation, the robot is represented by the

cell marked with the character 'R'. The side of the cell marked with '*' represents the front of the robot. The state of the robot is defined by its (row, column) coordinates and its orientation (i.e. the direction its front side is pointing towards).

At each time step, the agent is prompted to select an action. The robot has 4 nominal actions:

- Forward → move to the adjacent cell in the direction of the front of the robot (keeping the same orientation)
- Reverse → move to the adjacent cell in the opposite direction to the front of the robot (keeping the same orientation)
- Spin Left → rotate left (relative to the robot's front, i.e. counterclockwise) by 60 degrees (staying in the same cell)
- Spin Right → rotate right (i.e. clockwise) by 60 degrees (staying in the same cell)

Each time the robot selects an action, there is a fixed probability (given as a parameter of each testcase) for the robot to 'drift' by 60 degrees in a clockwise or counterclockwise direction (separate probabilities for each drift direction) before the selected nominal action is performed. The probability of drift occurring depends on which nominal action is selected, with some actions more likely to result in drift. Drifting CW and CCW are mutually exclusive events. Drift occurring does not cause any additional cost/reward penalty to be incurred except in the case where the movement direction resulting from drift causes a collision to occur.

Additionally, there is a fixed probability (also given as a parameter of each testcase) for the robot to 'double move', i.e. perform the nominal selected action twice. The probability of a double move occurring depends on which action is selected. Double movement may occur simultaneously with drift (CW or CCW). Double movement does not cause additional cost/reward penalty to be incurred (i.e. the movement is 'two for the price of one') except where the double movement results in a collision occurring.

The robot is equipped with a gripper on its front side which allows it to manipulate Widgets. When the robot is positioned with its front side adjacent to a widget, performing the 'Forward' action will result in the Widget being pushed, while performing the 'Reverse' action will result in the Widget being pulled.

Action Costs
Each action has an associated cost, representing the amount of energy used by performing that action.

If the robot moves without pushing or pulling a widget, the cost of the action is given by a base action cost, `ACTION_BASE_COST[a]` where 'a' is the action that was performed.

If the robot pushes or pulls a widget, an additional cost of `ACTION_PUSH_COST[a]` is added on top, so the total cost is `ACTION_BASE_COST[a] + ACTION_PUSH_COST[a]`.

The costs are detailed in the `constants.py` file of the support code:

```
ACTION_BASE_COST = {FORWARD: 1.0, REVERSE: 1.0, SPIN_LEFT: 0.1, SPIN_RIGHT: 0.1}
ACTION_PUSH_COST = {FORWARD: 0.8, REVERSE: 0.5, SPIN_LEFT: 0.0, SPIN_RIGHT: 0.0}
```

Obstacles
Some cells in the hex grid are obstacles. In the visualisation, these cells are filled with the character 'X'. Any action which causes the robot or any part of a Widget to enter an obstacle cell results in collision, causing a penalty value (given as a parameter of each testcase) to be subtracted from the received reward. The outside boundary of the hex grid behaves in the same way as an obstacle.

Additionally, the environment now contains an additional obstacle type, called 'hazards'. Hazards behave in the same way as obstacles, but when collision occurs, a different (larger) penalty is subtracted from the reward. As a result, avoiding collisions with hazards has greater importance than avoiding collisions with obstacles. Hazards are represented by '!!!' in the visualisation.

Widgets
Widgets are objects which occupy multiple cells of the hexagonal grid, and can be rotated and translated by

the HexBot robot. The state of each widget is defined by its centre position (row, column) coordinates and its orientation. Widgets have rotational symmetries - orientations which are rotationally symmetric are considered to be the same.

In the visualisation, each Widget in the environment is assigned a unique letter 'a', 'b', 'c', etc. Cells which are occupied by a widget are marked with the letter assigned to that widget (surrounded by round brackets). The centre position of the widget is marked by the uppercase version of the letter, while all other cells occupied by the widget are marked with the lowercase.

Three widget types are possible, called Widget3, Widget4 and Widget5, where the trailing number denotes the number of cells occupied by the widget. The shapes of these three Widget types and each of their possible orientations are shown in Figures 3 to 5 below.

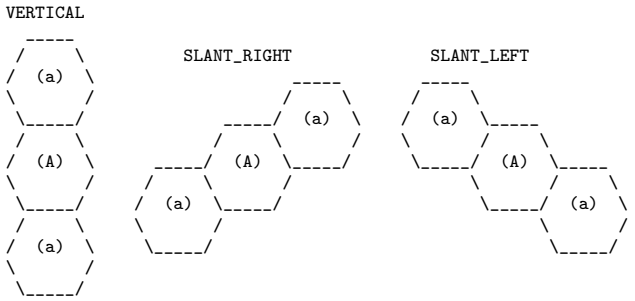

Figure 3: Widget3

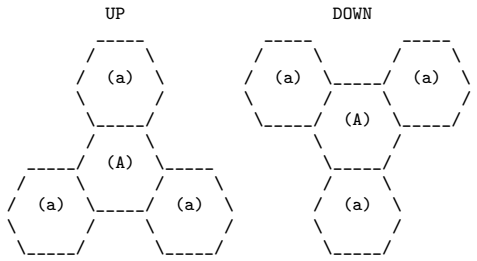

Figure 4: Widget4

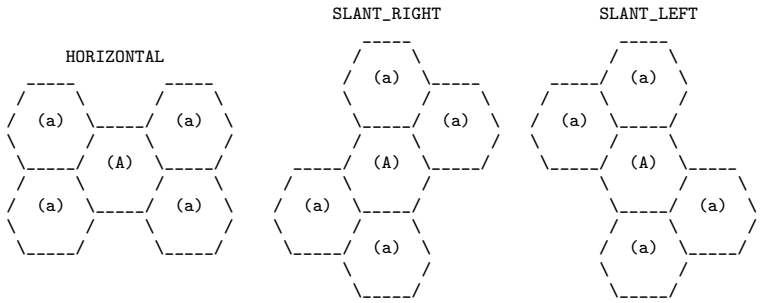

Figure 5: Widget5

Two types of widget movement are possible - translation (change in centre position) and rotation (change in orientation).

Translation occurs when the robot is positioned with its front side adjacent to one of the widgets cells such that the robot's orientation is in line with the widget's centre position. Translation results in the centre position of the widget moving in the same direction as the robot. The orientation of the widget does not change when translation occurs. Translation can occur when either 'Forward' or 'Reverse' actions are performed. For an action which results in translation to be valid, the new position of all cells of the moved widget must not intersect with the environment boundary, obstacles, the cells of any other widgets or the robot's new position.

Rotation occurs when the robot's new position intersects one of the cells of a widget but the robot's orientation does not point towards the centre of that widget. Rotation results in the widget spinning around its centre point, causing the widget to change orientation. The position of the centre point does not change when rotation occurs. Rotation can only occur for the 'Forward' action - performing 'Reverse' in a situation where 'Forward' would result in a widget rotation is considered invalid.

The following diagrams show which moves result in translation or rotation for each widget type, with the arrows indicating directions from which the robot can push or pull a widget in order to cause a translation or rotation of the widget. Pushing in a direction which is not marked with an arrow is considered invalid.

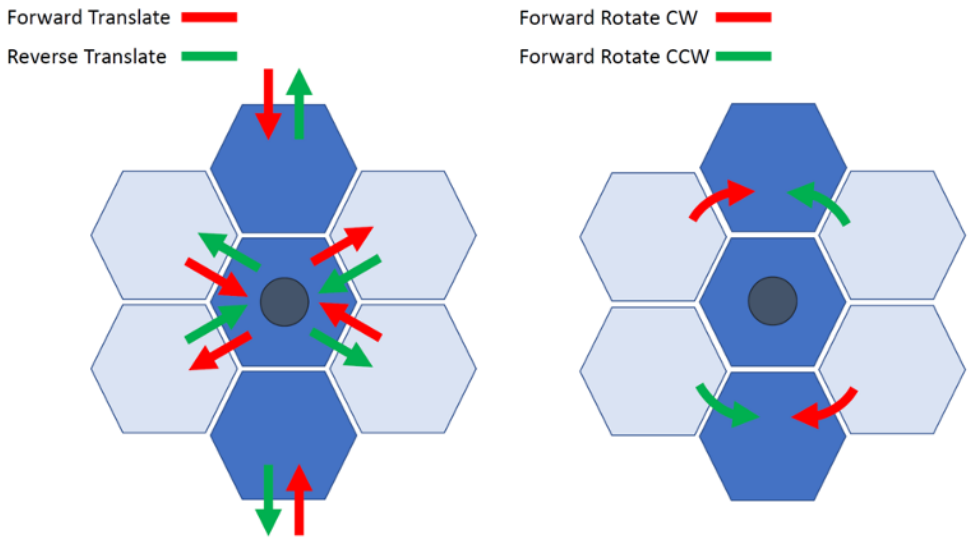

Figure 6: Widget3 translations and rotations

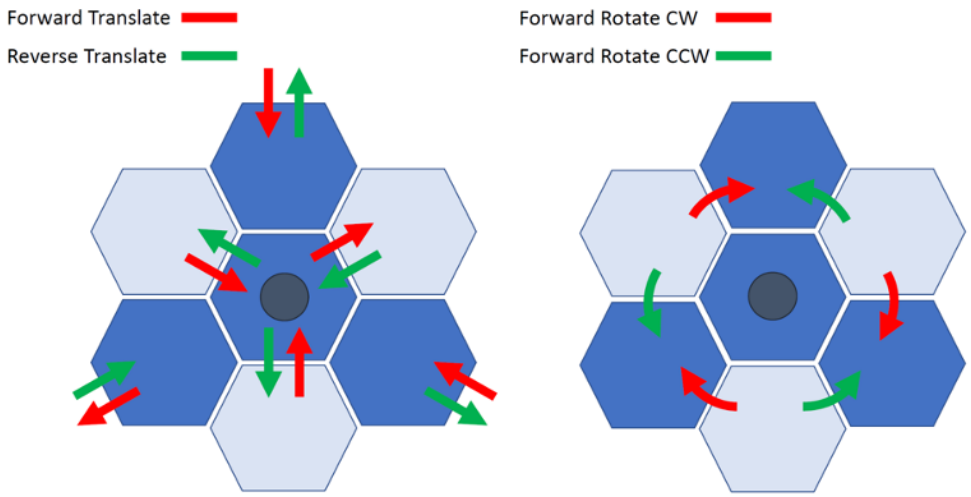

Figure 7: Widget4 translations and rotations

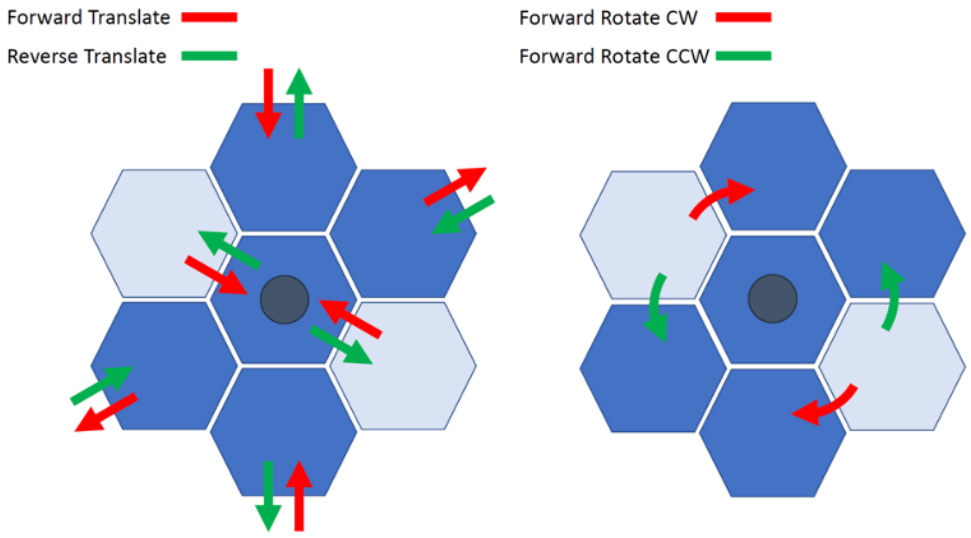

Figure 8: Widget5 translations and rotations

Targets
The hex grid contains a number of 'target' cells. In the visualisation, these cells are marked with 'tgt'. For a HexBot environment to be considered solved, each target cell must be occupied by part of a Widget. The number of targets in an environment is always less than or equal to the total number of cells occupied by all Widgets.

Interactive mode

A good way to gain an understanding of the game is to play it. You can play the game to get a feel for how it works by launching an interactive game session from the terminal with the following command:

```
$ python play.py <input_file>.txt
```

where `<input_file>.txt` is a valid testcase file (from the support code, with path relative to the current directory), e.g. `testcases/ex1.txt`.

Depending on your python installation, you should run the code using `python`, `python3` or `py`.

In interactive mode, type the symbol for your chosen action and press enter to perform the action: press 'W' to move the robot forward, 'S' to move the robot in reverse, 'A' to turn the robot left (counterclockwise) and 'D' to turn the robot right (clockwise). Use '[' to exit the simulation, and ']' to reset the environment to the initial configuration.

HexBot as an MDP

In this assignment, you will write the components of a program to play HexBot, with the objective of finding a high-quality solution to the problem using various sequential decision-making algorithms based on the Markov decision process (MDP) framework. This assignment will test your skills in defining a MDP for a practical problem and developing effective exact algorithms for MDPs.

What is provided to you

We will provide supporting code in Python, in the form of:

1. A class representing HexBot game map and a number of helper functions

2. A parser method to take an input file (testcase) and convert it into a HexBot map
3. A tester
4. Testcases to test and evaluate your solution
5. A script to allow you to play HexBot interactively
6. A solution file template

The support code can be found at: `https://gitlab.com/3702-2022/a2-support`. Autograding of code will be done through Gradescope, so that you can test your submission and continue to improve it based on this feedback — you are strongly encouraged to make use of this feedback.

Your assignment task

Your task is to develop algorithms for computing policies, and to write a report on your algorithms' performance. You will be graded on both your submitted **program (Part 1, 60%)** and the **report (Part 2, 40%)**. These percentages will be scaled to the 20% course weighting for this assessment item.

The provided support code formulates HexBot as an MDP, and your task is to submit code implementing the following MDP algorithms:

1. Value Iteration (VI)
2. Policy Iteration (PI)

Individual testcases specify which strategy (value iteration/policy iteration) will be applied, but you may modify the strategy specified in your local copy of the test cases for the purpose of comparing the performance of your two algorithms. Note that any local changes you make to the test cases will not modify the test cases on Gradescope (against which your programming component will be graded). The difficulty of higher level testcases will be designed to require a more advanced solution (e.g. linear algebra policy iteration).

Once you have implemented and tested the algorithms above, you are to complete the questions listed in the section "Part 2 - The Report" and submit the report to Gradescope.

More detail of what is required for the programming and report parts are given below.

Part 1 — The programming task

Your program will be graded using the Gradescope autograder, using testcases similar to those in the support code provided at `https://gitlab.com/3702-2022/a2-support`.

Interaction with the testcases and autograder

We now provide you with some details explaining how your code will interact with the testcases and the autograder (with special thanks to Nick Collins for his efforts making this work seamlessly).

Implement your solution using the supplied `solution.py` Template file. You are required to fill in the following method stubs:

- `__init__(game_env)`
- `vi_initialise()`
- `vi_is_converged()`
- `vi_iteration()`

- `vi_plan_offline()`
- `vi_get_state_value()`
- `vi_select_action()`
- `pi_initialise()`
- `pi_is_converged()`
- `pi_iteration()`
- `pi_plan_offline()`
- `pi_select_action()`

You can add additional helper methods and classes (either in `solution.py` or in files you create) if you wish. To ensure your code is handled correctly by the autograder, you should avoid using any try-except blocks in your implementation of the above methods (as this can interfere with our time-out handling). Refer to the documentation in `solution.py` for more details.

Grading rubric for the programming component (total marks: 60/100)

For marking, we will use 6 test cases to evaluate your solution. Each test case uses the algorithm specified as the solver type. Each test case is scored out of 10.0 marks, in the following four categories:

- Agent successfully reaches the goal
- Total reward
- Time elapsed
- Iterations performed

The 10 marks for each test case are evenly divided amongst the four categories (i.e. 2.5 marks are allocated for each category in each test case).

- Each test case has targets for total reward, time elapsed, and iterations performed.
- Maximum score is achieved when your program matches or beats the target in each category
- Partial marks are available for up to 1.3 x total reward, 2 x time elapsed and 1.3 x number of iterations
- Total mark for the test case is a weighted sum of the scores for each category
- Total code mark is the sum of the marks for each test case

Part 2 — The report

The report tests your understanding of MDP algorithms and the methods you have used in your code, and contributes 40/100 of your assignment mark.

Question 1. MDP problem definition (15 marks)

a) Define the State space, Action space, Transition function, and Reward function components of the HexBox MDP as well as where these are represented in your code. (10 marks)

b) Describe the purpose of a discount factor in MDPs. (2.5 marks)

c) State what the following dimensions of complexity are of your agent in the HexBot MDP. (See `https://artint.info/html/ArtInt_12.html` for definitions) (2.5 marks)

- Planning Horizon
- Sensing Uncertainty
- Effect Uncertainty
- Computational Limits
- Learning

Question 2. Comparison of algorithms and optimisations (15 marks)

This question requires a comparison of your implementation of value iteration (VI) and policy iteration (PI). If you did not implement PI, you may receive partial marks for this question by providing insightful relevant comments on your implementation of VI. For example, if you tried standard VI and asynchronous VI, you may compare these two approaches for partial marks.

a) Describe your implementations of value iteration and policy iteration in one sentence each. Include details such as whether you used asynchronous updates, and how you handled policy evaluation in PI. (2 marks)

b) Pick three representative testcases and compare the performance of Value Iteration (VI) and Policy Iteration (PI) according to the following measures. Please include the numerical values of your experiments in your comparisons.

   - Time to converge to the solution. (3 marks)
   - Number of iterations to converge to the solution. (3 marks)

   In order to do this, you'll need to modify the `# solver type` in your local copy of the test cases (the text files in the `testcases` directory).

c) Comment on the difference between the numbers you found for VI and PI. List any reasons why the differences either make sense, or do not make sense. (7 marks)

Question 3. "Risky Business": Optimal policy variation based on probabilities & costs (10 marks)
One consideration in the solution of a Markov Decision Process (i.e. the optimal policy) is the trade off between a risky higher reward vs a lower risk lower reward, which depends on the probabilities of non-deterministic dynamics of the environment and the rewards associated with certain states and actions.

Consider testcase `ex6.txt`, which includes a risky (but lower cost) path through the top half of the grid, and a less risky (but higher cost) path through the bottom half of the grid. Explore how the policy of the agent changes with `hazard penalty` and `transition probabilities`.

If you did not implement PI, you may change the solver type to VI in order to answer this question.

a) How do you expect the optimal path to change as the hazard penalty and transition probabilities change? Use facts about the algorithms to justify why you expect these changes to have such effects.(5 marks)

b) Picking three values for hazard penalty, and three sets of values for the transition probabilities, explore how the optimal policy changes over the 9 combinations of these factors. Do the experimental results align with what you thought should happen? If not, why? (5 marks)

Academic Misconduct

The University defines Academic Misconduct as involving "a range of unethical behaviours that are designed to give a student an unfair and unearned advantage over their peers." UQ takes Academic Misconduct very seriously and any suspected cases will be investigated through the University's standard policy (`https://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct`). If you are found guilty, you may be expelled from the University with no award.

It is the responsibility of the student to ensure that you understand what constitutes Academic Misconduct and to ensure that you do not break the rules. If you are unclear about what is required, please ask.

It is also the responsibility of the student to take reasonable precautions to guard against unauthorised access by others to his/her work, however stored in whatever format, both before and after assessment.

In the coding part of COMP3702 assignments, you are allowed to draw on publicly-accessible resources and provided tutorial solutions, but you must make reference or attribution to its source, by doing the following:

- All blocks of code that you take from public sources must be referenced in adjacent comments in your code.
- Please also include a list of references indicating code you have drawn on in your `solution.py` docstring.

However, you must not show your code to, or share your code with, any other student under any circumstances. You must not post your code to public discussion forums (including Ed Discussion) or save your code in publicly accessible repositories (check your security settings). You must not look at or copy code from any other student.

All submitted files (code and report) will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected. The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you collude to develop your code or answer your report questions, you will be caught.

For more information, please consult the following University web pages:

- Information regarding Academic Integrity and Misconduct:
  - https://my.uq.edu.au/information-and-services/manage-my-program/student-integrity-and-conduct/academic-integrity-and-student-conduct
  - http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct
- Information on Student Services:
  - https://www.uq.edu.au/student-services/

Late submission

Students should not leave assignment preparation until the last minute and must plan their workloads to meet advertised or notified deadlines. It is your responsibility to manage your time effectively.

Late Penalties: Where an assessment item is submitted after the original deadline without an approved extension, a late penalty will apply, as detailed in the COMP3702 Electronic Course Profile (ECP). The late penalty shall be

- 10% of the maximum possible mark for the assessment item will be deducted per calendar day (or part thereof) late, up to a maximum of seven (7) days. After seven days, no marks will be awarded for the item. A day is considered to be a 24 hour block from the assessment item due time. Negative marks will not be awarded.

In the event of exceptional circumstances, you may submit a request for an extension. You can find guidelines on acceptable reasons for an extension here https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/applying-extension All requests for extension must be submitted on the UQ Application for Extension of Progressive Assessment form at least 48 hours prior to the submission deadline.