COMP3702 Artificial Intelligence (Semester 2, 2022)
Assignment 3: HexBot Reinforcement Learning

Key information:

- **Due: 4pm, Thursday 27 October**
- This assignment will assess your skills in developing algorithms for solving Reinforcement Learning Problems.
- Assignment 3 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/`).
- 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/a3-support`.
- Your report (Part 2, 40/100) should fit the template provided, be in .pdf format and named according to the format `a3-COMP3702-[SID].pdf`, where SID is your student ID. Reports will be graded by the teaching team.

The HexBot AI Environment

You have been tasked with developing a reinforcement learning algorithm for automatically controlling HexBot, a multi-purpose robot which operates in a hexagonal environment. For this version of the task, the HexBot must navigate to one of the designated targets while incurring the minimum penalty score possible. Widgets do not need to be considered. To aid you in this task, we have provided a simulator and visualisation for the HexBot robot environment which you will interface with to develop your solution.

For A3, the HexGrid environment has non-deterministic action outcomes with unknown probabilities which are randomised for each testcase. The cost of each available action and the penalties for collision with obstacles and hazards are also unknown and randomised for each testcase. **Additions and modifications to this document from A2 are shown in magenta text. However, aspects of the A2 environment description have also been removed.** It is recommended you read this entire section, rather than skimming over the changes in pink.

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 1.

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

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).

The robot has 4 available nominal actions:

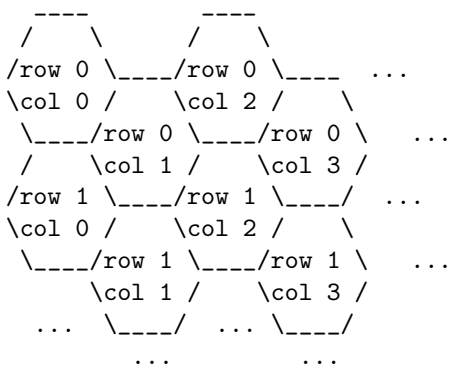

Figure 1: Example hexagonal grid showing the order that rows and columns are indexed

- 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 (set randomly based on the seed 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.

Additionally, there is a fixed probability (also set randomly based on the seed 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).

The reward received after each action is the minimum/most negative out of the rewards received for the nominal action and any additional (drift/double move) actions.

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 the agent to receive a negative obstacle collision penalty as reward. This reward replaces the movement cost which the agent would have otherwise incurred. 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 received as the reward. As a result, avoiding collisions with hazards has greater importance than avoiding collisions with obstacles. Hazards are represented by '!!!' in the visualisation.

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, one of the target cells must be occupied by the HexBot. Environments may contain multiple targets.

HexBot as a Reinforcement Learning problem

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 reinforcement learning algorithms. This assignment will

test your skills in defining reinforcement learning algorithms for a practical problem and understanding of key algorithm features and parameters.

What is provided to you

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

1. A class representing a 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/a3-support`. See the `README.md` for more details. 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 two reinforcement learning algorithms for computing paths (series of actions) for the agent, 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 provides a generative HexBot environment, and your task is to submit code implementing both of the following Reinforcement Learning algorithms:

1. Q-learning
2. SARSA

Individual testcases specify which strategy (Q-learning/SARSA) will be applied. Note that there are 3 hex grids, to which each algorithm is applied in a separate test (making 6 tests total over 2 algorithms for 3 grids).

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/a3-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, yet again!).

First, note that the Assignment 3 version of the class `Environment` (in `environment.py`) differs from previous assignments in that **the transition and reward functions are now randomised and unknown**

to the agent. The action outcome probabilities (double move, drift cw, and drift ccw) and costs/penalties (action costs, collision penalties, and hazard penalties) are randomised within some fixed range based on the seed of the filename, and are all stored in private variables. Your agent does not know these values, and therefore must interact with the environment to determine the optimal policy.

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

- `__init__()`
- `q_learn_train()`
- `q_learn_select_action(state)`
- `sarsa_train()`
- `sarsa_select_action(state)`

You may add to the `__init__` method if required, and 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). **The autograder will not allow you to upload your own copy of** `environment.py`.

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 training reward
- Total evaluation reward
- Time elapsed

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 training reward, total evaluation reward, and time elapsed.
- Maximum score is achieved when your program matches or beats the target in each category
- Partial marks are available for total training reward, total evaluation reward, and time elapsed.
- 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 Reinforcement Learning and the methods you have used in your code, and contributes 40/100 of your assignment mark.

Question 1. Q-learning is closely related to the Value Iteration algorithm for Markov decision processes.

a) (5 marks) Describe two key similarities between Q-learning and Value Iteration.

b) (5 marks) Give one key difference between Q-learning and Value Iteration.

For Questions 2, 3 and 4, consider test case `ex3.txt` for Q-learning and the equivalent SARSA test case, `ex4.txt`.

Question 2.

a) (5 marks) Explain the difference between off-policy and on-policy reinforcement learning algorithms. Explain where this difference is represented in the Q-learning and SARSA algorithms.

b) (5 marks) How does the difference between off-policy and on-policy algorithms affect the way in which Q-learning and SARSA solve test cases `ex3.txt` and `ex4.txt`? If you were unable to solve these test cases, it is sufficient to answer with reference to what you think would happen, based on the set up described in the test case files.

For questions 3 and 4, you are asked to plot the solution quality at each episode, as given by the 50-step moving average reward received by your learning agent. At time step t, the 50-step moving average reward is the average reward earned by your learning agent in the episodes $[t - 50, t]$, including episode restarts. If the Q-values imply a poor quality policy, this value will be low. If the Q-values correspond to a high-value policy, the 50-step moving average reward will be higher. We are using a moving average here because the reward is received only occasionally and there are sources of randomness in the transitions and the exploration strategy.

Question 3.

a) (5 marks) Plot (all on a single plot) the quality of the policy learned by Q-learning in testcase `ex3.txt` against episode number for three different fixed values of the `learning_rate` (which is called α in the lecture notes and in many texts and online tutorials), as given by the 50-step moving average reward (i.e. for this question, do not adjust α over time, rather keep it the same value throughout the learning process). Your plot should display the solution quality up to an episode count where the performance stabilises. This may take a significant number of episodes (e.g. >50,000) depending on the learning rates used. Note that the policy quality may still be noisy, but the algorithm's performance will stop increasing and its average quality will level out. Your plot should include axis labels and a legend.

b) (5 marks) Discuss the effect of varying the `learning_rate`. You should make reference to the Q-learning algorithm to support your discussion. If you were able to generate a plot in Q3a, you may also make reference to this in your discussion.

Question 4.

a) (5 marks) Plot (on a single plot) the quality of the learned policy against episode number under Q-learning and SARSA in test cases `ex3.txt` and `ex4.txt` respectively, as given by the 50-step moving average reward. Your plot should display the solution quality up to an episode count where the performance of both algorithms stabilise. Your plot should include axis labels and a legend.

b) (5 marks) With reference to your plot, compare the learning trajectory of the two algorithms, and their final solution quality. Discuss the way the solution quality of Q-learning and SARSA change as they learn to solve the testcase, both as they learn and once they have stabilised.

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.

It may take the autograder up to an hour to grade your submission. It is your responsibility to ensure you are uploading your code early enough and often enough that you are able to resolve any issues that may be revealed by the autograder *before the deadline*. Submitting non-functional code just before the deadline, and not allowing enough time to update your code in response to autograder feedback is not considered a valid reason to submit late without penalty.

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.