COMP3702 Artificial Intelligence (Semester 2, 2022)
Assignment 1: Search in HexBot

Key information:

- **Due: 4pm, Thursday 25 August 2022**
- This assignment assesses your skills in developing discrete search techniques for challenging problems.
- Assignment 1 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 a link to the COMP3702 Gradescope site on Blackboard.
- Your code (Part 1) will be graded using the Gradescope code autograder, using the testcases in the support code provided at `https://gitlab.com/3702-2022/a1-support`.
- Your report (Part 2) should fit the template provided, be in .pdf format and named according to the format `a1-COMP3702-[SID].pdf`. Reports will be graded by the teaching team.

The HexBot Robot AI Environment

You have been tasked with developing a search 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.

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.

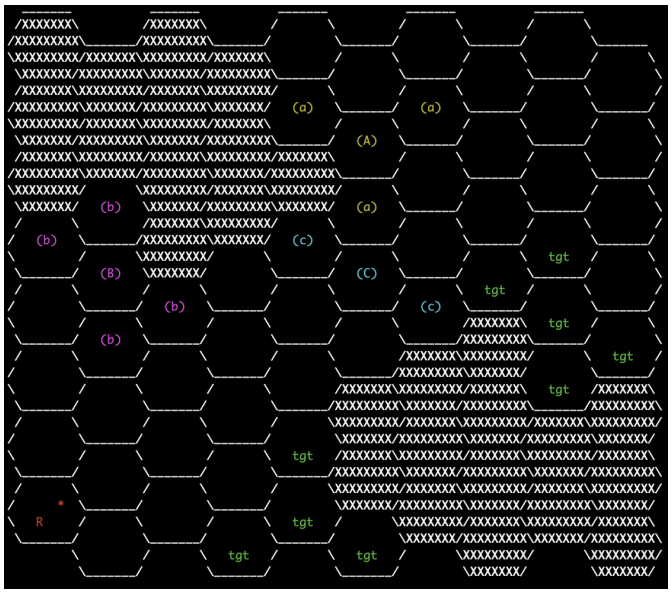

Figure 1: Example game level of HexBot

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.

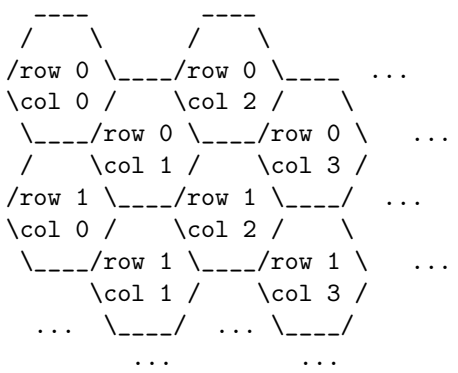

Figure 2: Example hexagonal grid showing the order that rows and columns are indexed

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 available actions:

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

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 is invalid (i.e. results in collision). The outside boundary of the hex grid behaves in the same way as an obstacle.

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.

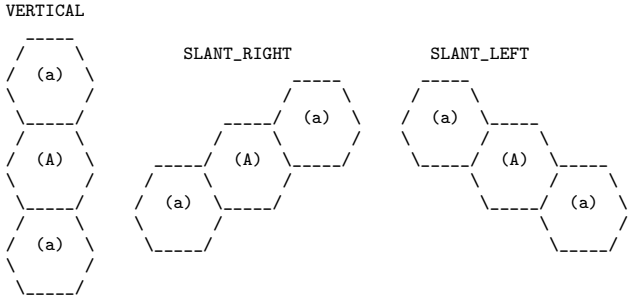

Figure 3: Widget3

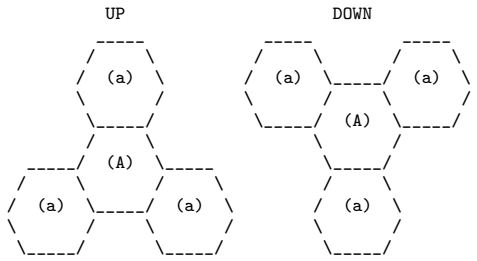

Figure 4: Widget4

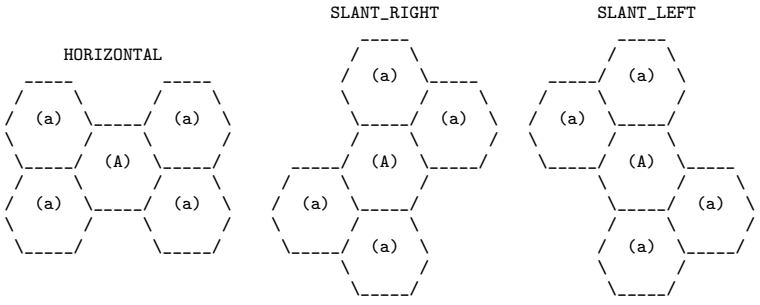

Figure 5: Widget5

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 current position is adjacent to the centre of the widget but the robot's orientation does not point towards the centre of the 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.

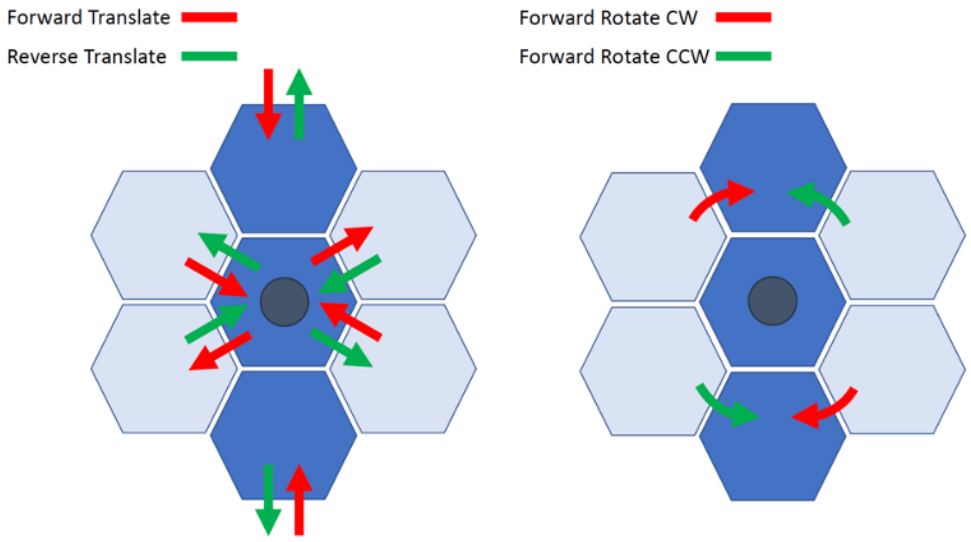

Figure 6: Widget3 translations and rotations

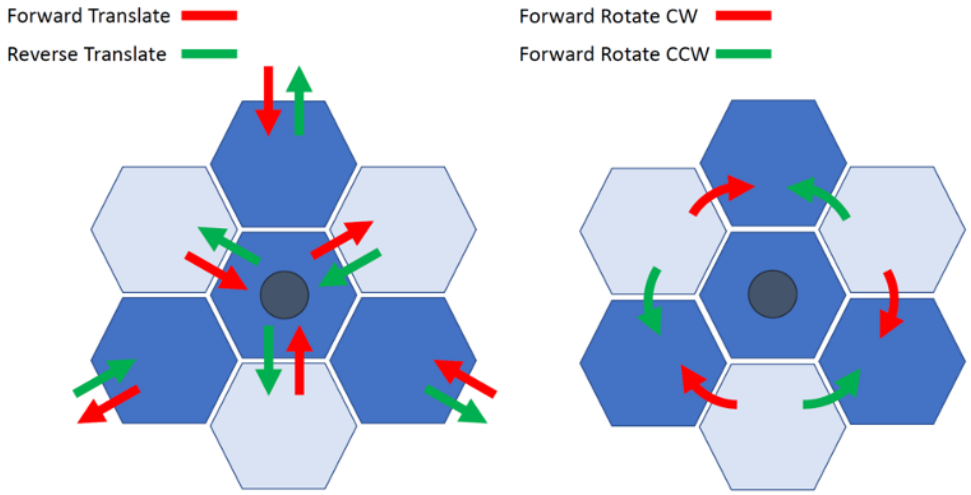

Figure 7: Widget4 translations and rotations

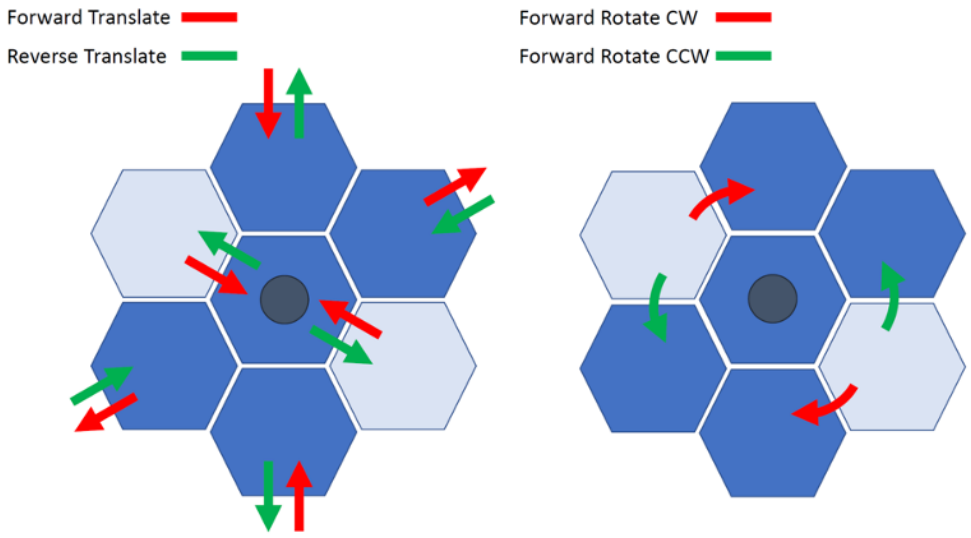

Figure 8: Widget5 translations and rotations

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 a search 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 search algorithms. This assignment will test your skills in defining a search space for a practical problem and developing good heuristics to make your program more efficient.

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 state visualiser
4. A tester
5. Testcases to test and evaluate your solution
6. A solution file template

The support code can be found at: `https://gitlab.com/3702-2022/a1-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 a program that determines a path for the agent (i.e. the HexBot) to solve game levels, and to provide a written report explaining your design decisions and analysing 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.

To turn HexBot into a search problem, you have will have to first define the following agent design components:

- A problem state representation (state space),
- A successor function that indicates which states can be reached from a given state (action space and transition function), and
- A cost function

Note that a goal-state test function is provided in the support code (in the `is_solved()` method of `environment.py`). Once you have defined the components above, you are to develop and submit code implementing two discrete search algorithms:

1. Uniform-Cost Search, and
2. A* Search

Your submitted code should run A* search or UCS search based on the '`search_type`' argument of the tester. Both UCS and A* will be run separately by the autograder. Finally, after you have implemented and tested the algorithms above, you are to complete the questions listed in the section "Part 2 - The Report" and submit them as a written report.

More detail of what is required for the programming and report parts are given below. *Hint: Start by implementing a working version of UCS, and then build your A* search algorithm out of UCS using your own heuristics.*

Part 1 — The programming task

Your program will be graded using the Gradescope autograder, using the testcases in the support code provided at `https://gitlab.com/3702-2022/a1-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). Your solution code interacts with the autograder via the path it returns from the Solver class of `solution.py` via the two methods, `solver.solve_ucs()` and `solver.solve_a_star()`.

This is handled as follows:

- The file `solution.py`, supplied in the support code, is a template for you to write your solution. All of the code you write can go inside this file, or if you create your own additional python files they must be invoked from this file.
- The script `tester.py` can be used to test your code on testcases.
  After you have implemented UCS (uniform cost search) and/or A* search in `solution.py` you can test them by going to your command prompt, navigating to your folder and running `tester.py`:

  Usage:

  ```
  $ python tester.py [search_type] [testcases] [-v (optional)]
  ```

  - search_type = 'ucs', 'a_star' or 'both'
  - testcases = a comma separated list of numbers of which test cases to run (can be 1 up to 5) (e.g. '1,3,4')
  - if -v is specified, the solver's trajectory will be visualised

  For example, to test UCS after you have written the code for it, you can type the following in the command prompt:

  ```
  $ python tester.py ucs 1 -v
  ```

- The *autograder* (hidden to students) handles running your python program with all of the testcases. It will run the tester python program on your solution code and assign a mark for each testcase based on the return code of tester.
- You can inspect the testcases in the support code, which each include information on their optimal solution cost (target cost), target times and target number of nodes expanded (for UCS and A*). Looking at the testcases might also help you develop heuristics using your human intelligence and intuition.
- To ensure your submission is graded correctly, write your solution code in `solution.py`, and do not rename any of the provided files or alter the methods in `environment.py` or `state.py`.

More detailed information on the HexBot implementation is provided in the Assignment 1 Support Code *README.md* and code comments.

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

For marking, we will use 5 test cases each run twice (for UCS and A*) to evaluate your solution.

Each test case is scored out of 6.0 marks, divided evenly into 1.5 for each category.

- Each test case is scored in four categories:
  - Completion (1 $\times$ completion weight if path ends at the goal, 0 otherwise)

  - Path Cost
  - Time Elapsed
  - #Nodes Expanded
- Each test case has a single target for Path Cost (applied to both UCS and A*)
- Each test case has separate targets for Time Elapsed and #Nodes expanded for UCS and A* (where the targets for A* are harder to achieve)
- Maximum score is achieved when your program matches or beats the target in each category
- Partial marks are available for up to 2x cost, 2x time elapsed and 2x number of nodes expanded
- 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 the methods you have used in your code, and contributes 40/100 of your assignment mark. **Please make use of the report templates provided on Blackboard**, because Gradescope makes use of a predefined assignment template. Submit your report via Gradescope, in .pdf format (i.e. use "save as pdf" or "print-to-file" functionality), and named according to the format `a1-COMP3702-[SID].pdf`. Reports will be graded by the teaching team.

Your report task is to answer the questions below:

Question 1. **(5 marks)**
State the ten dimensions of complexity of HexBot, and explain your selection.

Refer to the P&M textbook `https://artint.info/html/ArtInt_12.html` for a description of each dimension (modularity, representation, planning horizon, sensing uncertainty, effect uncertainty, preference, number of agents, learning, computational limits, and interaction).

Question 2. **(5 marks)**
Describe the components of your agent design for HexBot. Specifically, the Action Space, State Space, Transition Function and Utility Function.

Question 3. **(15 marks)**
Compare the performance of Uniform Cost Search and A* search in terms of the following statistics:

a) The number of nodes generated
b) The number of nodes on the frontier container when the search terminates
c) The number of nodes on the explored list (if there is one) when the search terminates
d) The run time of the algorithm (e.g. in units such as mins:secs). Note that you can report run-times from your own machine, not the Gradescope servers.
e) Discuss and interpret these results. If you are unable to implement A* search, please report and discuss the statistics above for UCS only, and what you would expect to change for A*.

Question 4. **(15 marks)**
Some challenging aspects of designing a HexBot agent are the hexagonal grid, the asymmetric cost of actions (pushing is more expensive than pulling a widget), rotation of widgets to avoid obstacles, choosing the order in which to manoeuvre each widget, and determining which target squares to cover.

Describe heuristics (or components of a combined heuristic function) that you have developed in the HexBot search task that account for these aspects or any other challenging aspects you have identified of the problem. Your documentation should provide a thorough explanation of the rationale for using your chosen heuristics considering factors such as admissibility and computational complexity (maximum of 5 marks per heuristic).

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, 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.