COMP3702/COMP7702 Artificial Intelligence (Semester 2, 2020)
Assignment 1: Search in LaserTank

Key information:

- **Due: 5pm, Friday 4 September**
- This assignment will assess your skills in developing discrete search techniques for challenging problems.
- Assignment 1 contributes 10% 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/COMP7702 Gradescope site on Blackboard.
- Your program (Part 1) will be graded using the Gradescope code autograder, using the testcases in the support code provided at `https://gitlab.com/3702-2020/assignment-1-support-code`.
- Your report (Part 2) should fit the template provided, be in .pdf format and named according to the format `a1-[courseCode]-[SID].pdf`. Reports will be graded by the teaching team.

The LaserTank AI Environment

LaserTank is an open source computer puzzle game requiring logical thinking to solve a variety of levels, originally developed by Jim Kindley in 1995. In LaserTank, a player controls a tank vehicle which can move forwards, turn clockwise and counterclockwise, and shoot a laser which can be used to interact with special map elements. The game takes place in a 2D map divided into grid cells. The player must navigate from a **starting position** to the **flag** in as few moves as possible while avoiding "game over" conditions. The game is over if the player moves into a dangerous cell (i.e. water cells or any cell in the line of fire area of an anti-tank).

The LaserTank codebase comes as part of your assignment support code; while a high-level description is provided in a separate LaserTank AI Environment document, which you can find on Blackboard.

LaserTank as a search problem

In this assignment, you will write the components of a program to play LaserTank, 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 only, in the form of:

1. A class representing LaserTank game map and a number of helper functions
2. A parser method to take an input file (testcase) and convert it into a LaserTank map
3. A state visualiser
4. A tester
5. Testcases to test and evaluate your solution
6. A solver file template

The support code can be found at: `https://gitlab.com/3702-2020/assignment-1-support-code`. 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 outputs a path (series of actions) for the agent (i.e. the Laser Tank), 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 10% course weighting for this assessment item.

To turn LaserTank 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 (utility function); **we assume that each step has a uniform cost of 1.**

Note that a goal-state test function is provided in the support code. Once you have defined the components above, you are to submit code implementing one of two discrete search algorithms:

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

Your submitted code should run your A* search implementation if you have it. If you haven't been able to implement A* search, your code can run UCS instead. 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. Under the grading rubric discussed below, the testcases used to assess the programming component will give a higher mark for A* search, and you will not be able to answer some of the report questions without considering A* or implementing it. These elements of the rubric give you an incentive to implement A* search over the simpler UCS algorithm. *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 COMP3702 testcases in the support code provided at `https://gitlab.com/3702-2020/assignment-1-support-code`.

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). Your solution code only needs to interact with the autograder via the *output_file* it generates. This is handled as follows:

- The file `solver.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.
- Your program will: (i) take a `testcase` filename and an `output_filename` as arguments, (ii) find a solution to the testcase, and (iii) write the solution to an output file with the given `output_filename`.
- Your code should generate a solution in the form of a comma-separated list of `actions`, taken from the set of *move symbols* defined in the supplied `laser_tank.py` file, which are:
  - `MOVE_FORWARD = 'f'`
  - `TURN_LEFT = 'l'`
  - `TURN_RIGHT = 'r'`
  - `SHOOT_LASER = 's'`

- The `main()` method stub in `solver.py` makes it clear how to interact with the environment: (i) The `LaserTankMap.process_input_file(filename)` function handles reading the input file, (ii) your code is called to solve the problem, with your solver's actions written to the `actions` variable, then (iii) `write_output_file(filename, actions)` function handles writing to the output file in the correct format, which is passed to the autograder.
- The script `tester.py` can be used to test individual testcases.
- The *autograder* (hidden to students) handles running your python program with all of the testcases. It will run the tester python program on your output file 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 path lengths and test time limits. Looking at the testcases might also help you develop heuristics using your human intelligence and intuition.
- To ensure your submission is graded correctly, do not rename any of the provided files or alter the methods `LaserTankMap.process_input_file()` or `solver.write_output_file()`.

More detailed information on the LaserTank implementation is provided in the Assignment 1 Support Code *README.md*, while a high-level description is provided in the LaserTank AI Environment description document.

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

For marking, we will use 8 different testcases to evaluate your solution. There will be 3 easy, 3 medium and 2 difficult test cases, and marks will be allocated according to the following rules:

- *Solving* a testcase means finding *an optimal path* within the given time limit (time limits are given in each test case file).
- *Approximately solving* a testcase means finding *a sub-optimal solution path*, which is longer than an optimal one, within the given time limit.
- If your code computes one (approximate or optimal) solution to one test case, COMP3702 students receive 25 marks, and COMP7702 students receive 20 marks.
- Above this, each subsequent testcase that your code solves receives another 5 marks per testcase, up to a maximum of 60 marks.
- Approximate solutions are penalised in proportion to how far they are from an optimal solution's length. Each subsequent testcase that your code approximately solves receives 5 marks minus a penalty of 0.5 marks for each step over the optimal solution length, to a minimum of 0 marks for approximate solutions that are 10 steps or more longer than the optimal path.
- Part marks are given for programming attempts that fail to (approximately or optimally) solve one test case, as indicated in the tables below.

The details of separate grading rubrics for COMP3702 and COMP7702 are given in the tables below, where your marks are given by the highest performance threshold your submission passes.

Note on Gradescope's autograder: Due to limitations with the Gradescope autograder, your marks for Part 1 displayed within Gradescope will reflect only the marks you have earned for solving or approximately solving each test case. The additional 25 marks for COMP3702 students, or 20 marks for COMP7702 students, for approximately solving one or more testcases will be added manually in Blackboard, as will part marks for submission that fail to meet this threshold.

COMP3702	
Marks	*Performance threshold*
<10	The program does not run (marks at manual grader's discretion).
10	The program runs but fails to approximately solve any testcases within $2\times$ the given time limit.
20	The program approximately solves one of the testcases within $2\times$ the time limit.
25	The program approximately solves more than one of the testcases within $2\times$ the time limit.
	For the remainder, the higher mark in the left column are for optimal solutions. Approximate solutions incur a penalty of 0.5 marks for each step over the optimal solution length, with 0 marks for ≥10 steps longer
(25-) 30	The program (approximately) solves one testcase.
(30-) 35	The program (approximately) solves 2 testcases.
(35-) 40	The program (approximately) solves 3 testcases.
(40-) 45	The program (approximately) solves 4 testcases.
(45-) 50	The program (approximately) solves 5 testcases.
(50-) 55	The program (approximately) solves 6 testcases.
(55-) 60	The program (approximately) solves at least 7 of the 8 testcases.

COMP7702	
Marks	*Performance threshold*
<10	The program does not run (marks at manual grader's discretion).
10	The program runs but fails to approximately solve any testcases within $2\times$ the given time limit.
15	The program approximately solves one of the testcases within $2\times$ the time limit.
20	The program approximately solves more than one of the testcases within $2\times$ the time limit.
	For the remainder, the higher mark in the left column are for optimal solutions. Approximate solutions incur a penalty of 0.5 marks for each step over the optimal solution length, with 0 marks for ≥10 steps longer
(20-) 25	The program (approximately) solves one testcase.
(25-) 30	The program (approximately) solves 2 testcase.
(30-) 35	The program (approximately) solves 3 testcases.
(35-) 40	The program (approximately) solves 4 testcases.
(40-) 45	The program (approximately) solves 5 testcases.
(45-) 50	The program (approximately) solves 6 testcases.
(50-) 55	The program (approximately) solves 7 testcases.
(55-) 60	The program (approximately) solves all 8 testcases.

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-[courseCode]-[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 dimensions of complexity in LaserTank, and explain your selection.

Question 2. **(5 marks)**
Describe the components of your agent design for LaserTank.

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

a) The number of nodes generated

b) The number of nodes on the fringe 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.

Question 4. **(15 marks)**

A challenging aspect of designing a LaserTank agent are the "special" map elements, such as the ice, bricks, mirrors and teleporter tiles. Design and describe functions, i.e. heuristics, that account for some or all of these special tiles in the search task. Your documentation should provide a thorough explanation of the rationale for using your chosen heuristics (maximum of 5 marks per heuristic).