COMP3702/COMP7702 Artificial Intelligence (Semester 2, 2020)
Assignment 1: Search in LASERTANK – **Report Template**

Name: Bosheng Zhang

Student ID: 45004830

Student email: bosheng.zhang@uqconnect.edu.au

Note: Please edit the name, student ID number and student email to reflect your identity and **do not modify the design or the layout in the assignment template**, including changing the paging.

Question 1 (Complete your full answer to Question 1 on the remainder page 1)

State the **dimensions of complexity** in LaserTank and explain your selection.

Dimension	Values
Modularity:	**Flat**: The game only has one level of abstraction, there is no organizational structure or interacting modules.
Planning horizon:	**Indefinite stages** planners are player who can perform limited but not predetermined steps in advance. For example, an agent who must reach a certain location may not know the steps to get there in advance.
Representation:	**States**: Each of the different ways the grid could be, would affect what the agent should do next.
Computational limits:	**Perfect rationality,** where an agent reasons about the best action without taking into account its limited computational resources.
Learning:	**Knowledge is given:** The knowledge to decide what to do is provided in the code.
Sensing uncertainty:	**Fully observable** is when the agent knows the state of the world from the observations (the agent knows each symbol meaning).
Effect uncertainty:	**Deterministic:** when the state resulting from an action is determined by an action and the prior state
Preference:	**achievement goal** is a goal to achieve. LaserTank's goal is player should arrive flag
Number of agents:	**single agent reasoning,** where the agent assumes that any other agents are just part of the environment. There are no other agents.
Interaction:	**reason offline:** All states are generated on the local computer without interacting with the environment

Question 2 (Complete your full answer to Question 2 on page 2)

State space: Does not consider map elements: land, water, brick, ice, bridge, anti-tank (with orientation) and mirrors (with orientation) OR Does not consider the coordinates and heading of the Tank.

First, we notice that LaserTankMap is a **fully observable** game, meaning that the agent always knows the exact state of the game from observing the board; there is no hidden information that the agent cannot perceive. This means that the percept space is the same as the state space; P = S. Following from this, the perception function Z simply maps each state to itself. Thus, P and Z are not required, and we only need to define A, S, T, and U.

- **Action Space (A)**: The action space only consists of 4 actions that are:

  LaserTankMap.MOVES = ['f', 'l', 'r', 's']

- **Percept Space (P)**: the problem is fully observable, and thus we do not need to formally specify the percept space.,

  P = S

- **State Space (S)**: The LaserTankMap contains y_size * x_size cells stored in a grid_data. This can be represented by input file symbols
- **World Dynamics/Transition Function (T : S x A -> S')**: The world dynamics describe how the environment reacts in response to the agent's actions. Here in LaserTank, the transition function is apply_move(). It will change the game_map based on different action is given as parameter.
- **Perception Function (Z : S -> P)**: Since the problem is fully observable, a percept function is not required. Equivalently, it can be described by the identity map,

  Z = S -> P

- **Utility Function (U : S -> IR)**: Since the agent is supposed to find shortest paths, we can represent this in the form of utility by assigning the agent disutility (i.e. negative utility) proportional to the total_cost (cost_so_far) of the grid cells that it has traveled. This can be represented by a function c : S -> IR; which specifies the cost of cell of grid data as a real number, e.g. in steps. The agent can then associate its total utility with the total cost of a sequence of states,

  $$\text{i.e. } \mathrm{U}((\mathrm{s1}, \mathrm{s2}, \mathrm{s3}, \ldots, \mathrm{sn})) = -\sum_{i=1}^{n} \mathrm{c}(\mathrm{si})$$

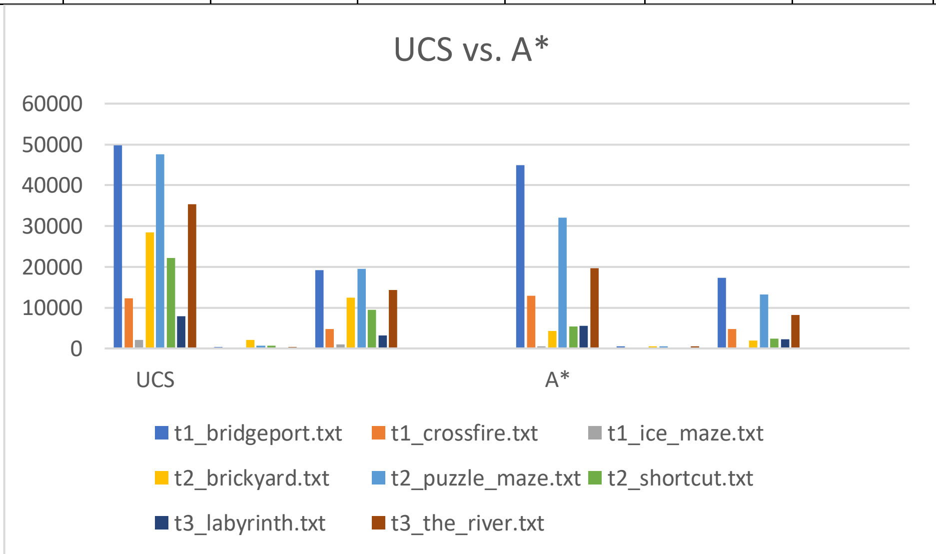
Question 3 (Complete your full answer to Question 3 on page 3)

In your explanation, it would be good to comment on why your A* performs better on some testcases than others (and how this relates to your choice of heuristic).

Run t1_bridgeport.txt

Search type	UCS	A*
The number of nodes generated	49725	44953
The number of nodes on the fringe when the search terminates	419	541
The number of nodes on the explored list when the search terminates	19115	17331
The run time of the algorithm (in unit secs). report run-times from my own machine.	1.1349396705627441	1.005347490310669

map	t1_bridgeport.txt	t1_crossfire.txt	t1_ice_maze.txt	t2_brickyard.txt	t2_puzzle_maze.txt	t2_shortcut.txt	t3_labyrinth.txt	t3_the_river.txt
UCS	49725	12320	2141	28478	47594	22168	7933	35310
	419	1	131	2052	653	719	68	363
	19115	4683	965	12474	19497	9434	3189	14315
	1.1349396705627441	0.28623414039611816	0.10870933532714844	0.6672408580780029	1.5827827453613281	0.8946084976196289	1.3633553981781006	2.1851577758789062
A*	44953	12829	442	4235	32095	5422	5500	19618
	541	1	61	445	504	217	17	457
	17331	4683	178	1976	13193	2356	2256	8165
	1.0053474903106690	0.30916333198547363	0.02597975730895996	0.09374880790710449	1.0123109817504883	0.19548320770263672	1.0751252174377441	1.2277193069458008



Discuss and interpret these results	UCS	A*
	By adding the heuristic, the overall performance of A* is better than the overall performance of UCS	
	Uniform-cost search is uninformed search: it does not use any domain knowledge. It expands the lowest cost node, and it does so in every action because no information about the goal is provided. It can be regarded as a function f(n) = g(n) where g(n) is the path cost	A* search is informed search: it uses a heuristic function to estimate how close the current state is to the flag (are we getting close to the flag?). Therefore, our cost function f(n) = g(n) is combined with the cost from n to the flag, the h(n) - heuristic function for estimating the cost, giving us

	("path cost" itself is a function that assigns a numeric cost to a path based on performance measure - number of moves).	$f(n) = g(n) + h(n)$. Both methods have an expanded node list, but A* search will try to minimize the number of expanded nodes (path cost + heuristic function).

Question 4 (Complete your full answer to Question 4 on pages 4 and 5, and keep page 5 blank if you do not need it)
Minor concept error regarding admissability of Manhattan distance.

A* uses f (p) = g(p) + h(p)
The heuristic function h(n) tells A* an estimate of the minimum cost from any cells in grid data n to the goal.

- At one extreme, if h(n) is 0, then only g(n) plays a role, and A* turns into Dijkstra's Algorithm, which is guaranteed to find a shortest path.

Manhattan distance

The better heuristic for overall performance across all the testcases. The Manhattan distance is an admissible heuristic in our testcases. Admissible heuristics must not overestimate the number of moves to solve this problem. Here the player can only move the block 1 at a time and in only one of the 4 directions, the optimal scenario for each coordinate is that it has a clear, unobstructed path to the flag. This is a Manhattan Distance of 1.

```
def man_distance(self, a, b):
  return abs(a[0] - b[0]) + abs(a[1] - b[1])
```

However, note that although an admissible heuristic can guarantee final optimality, it's not necessarily efficient.

Manhattan distance * constant value

It finds the optimal steps by using shorter time and visited smaller number of nodes than by using just Manhattan distance in certain testcases.
For example, the time taken of using (Manhattan distance * constant value) as heuristic, 0.7011277675628662 seconds and the number of explored nodes is 8095. However, the time taken of using pure Manhattan distance is 1.067188024520874 seconds and the number of explored nodes is 13193.

```
def man_distance(self, a, b):
  return const * abs(a[0] - b[0]) + abs(a[1] - b[1])
```

This heuristic is revised version of Manhattan distance, since the LaserTankMap contains special symbol (e.g. ice and teleport) may cause the estimate cost is not accurate.

Teleport Special case

In some test cases, such as t2_shortcut.txt. We may overestimate the cost between the player and the target and make the cost inaccurate, because teleporting will move the player closer to the flag, which cannot be considered in the Manhattan distance.

So here we are can use teleport special case heuristic to estimate the cost accurate.

In this heuristic method, we will still use Manhattan distance as an auxiliary function. We create a list to store the locations of all teleporters, and then we compare the distance between the player and each teleporter to find the shortest path to one teleport. Then delete it from the list, and then calculate the distance between the paired teleport and our target. We add these two distances together as the cost.

Then determine one more step, compare the Manhattan distance between the player and the target to see with the cost if the Manhattan distance is small, then we use the Manhattan distance.

Ice Special case

In some test cases, such as t1_ice_maze.txt. We may overestimate the cost between the player and the target and make the cost inaccurate, because ice will move the player closer to the flag, which cannot be considered in the Manhattan distance.

So here we are can use ice special case heuristic to estimate the cost accurate.

In this heuristic method, we will still use Manhattan distance as an auxiliary function. We create a list to store the locations of all ices, and then we compare the distance between the player and each ice to find the shortest path to one ice.

Then determine one more step, compare the Manhattan distance between the player and the target to see with the cost if the Manhattan distance is small, then we use the Manhattan distance.