German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

Introduction to Artificial Intelligence, Winter Term 2019
Project 1: Endgame [1]

Due: October 17^{th} *2018*

1. Project Description

At the dawn of the universe, there was nothing. Then, BOOM! The Big Bang sent six elemental crystals, known as the infinity stones, hurtling across the newly-born universe. Each infinity stone controls an essential aspect of existence and possesses unique immense capabilities that have been enhanced by various alien civilizations for millennia. The power of the six stones, if combined, can grant their possessor ultimate control over the entire universe. The mighty titan, Thanos, has long believed that the massive population of the universe would inevitably use up the entire supply of available resources leading to the universe's eventual demise. He believed that the only hope for the universe's survival is to annihilate half of the population utilizing the power of the infinity stones. The Avengers - a team of super heroes whose main aim is to protect the universe from detrimental threats - swore to thwart Thanos' plan. However, after a long and ferocious battle with the Avengers, Thanos eventually succeeded in collecting the stones and snapped his fingers wiping out half of all life in the universe. Having accomplished his mission, Thanos fled with the stones and his warrirors to a deserted planet to retire. He hid each stone in a different place on the planet until he figures out a way to destroy them.

After losing the war against Thanos, the surviving Avengers separated thinking that all hope was gone. Iron Man, however, had a plan to save the universe. He boarded his space ship and set out to find Thanos and retrieve the infinity stones which he planned to use to reverse what Thanos did and kill Thanos once and for all. After locating Thanos' planet, Iron Man used an advanced technology he invented to freeze Thanos and his warrirors in place.

The planet can be thought of as an $m \times n$ grid of cells where $5 \leq m, n \leq 15$. A grid cell is either free or contains one of the following: Iron Man, Thanos, one of Thanos' warrirors, or an infinity stone. In this project, you will use *search* to help Iron Man defeat Thanos by first collecting the six infinity stones, then heading to the cell where Thanos is frozen, and finally snapping his fingers to return the universe to the way it used to be. Iron Man can move in the four directions as long as the cell in the direction of movement does not contain Thanos or a living warrior. He can collect an infinity stone only if he is in the same cell with the stone. Since the stones possess immense powers, collecting a stone causes 3 units of damage to Iron Man. At any point with one move, Iron Man can kill all warrirors lying in adjacent cells. An adjacent cell is a cell that lies one step to the north, south, east, or west. Killing a single warrior will increase Iron Man's received damage by 2 units since they will attempt to fight back. Once a warrior is killed, Iron Man can move through the cell where the warrior was. Since Thanos and his warriors are very powerful, they can cause damage to Iron Man as long as he is in an adjacent

[1] This project's theme is based on the MCU's Avengers: Infinity War and Avengers: Endgame.

cell to them. Being in an adjacent cell to Thanos increases Iron Man's received damage by 5 units for each time step, while being in an adjacent cell to a warrior increases Iron Man's received damage by 1 unit for each time step. A time step is the time required to do an action. After collecting the stones, Iron Man will be able to enter the cell where Thanos is and snap his fingers only if his received damage is less than 100 units.

Using search you should formulate a plan that Iron Man can follow to defeat Thanos if one exists. An optimal plan is one where Iron Man's damage is minimal. The following search algorithms will be implemented and each will be used to help Iron Man:

a) Breadth-first search.

b) Depth-first search.

c) Iterative deepening search.

d) Uniform-cost search.

e) Greedy search with at least two heuristics.

f) A* search with at least two *admissible* heuristics.

Each of these algorithms should be tested and compared in terms of the number of search tree nodes expanded.

You must only use Java 1.8 to implement this project.

Your implementation should have the following function in a class named `Main.java` placed in the *default package* of your project.

- `public static String solve(String grid, String strategy, String visualize)` uses search to try to formulate a winning plan:
  - *grid* is a string representing the grid to perform the search on. This string should be in the following format:
    `m,n;ix,iy;tx,ty;`
    $\texttt{s}_1\texttt{x},\texttt{s}_1\texttt{y},\texttt{s}_2\texttt{x},\texttt{s}_2\texttt{y},\texttt{s}_3\texttt{x},\texttt{s}_3\texttt{y},\texttt{s}_4\texttt{x},\texttt{s}_4\texttt{y},\texttt{s}_5\texttt{x},\texttt{s}_5\texttt{y},\texttt{s}_6\texttt{x},\texttt{s}_6\texttt{y};$
    $\texttt{w}_1\texttt{x},\texttt{w}_1\texttt{y},\texttt{w}_2\texttt{x},\texttt{w}_2\texttt{y},\texttt{w}_3\texttt{x},\texttt{w}_3\texttt{y},\texttt{w}_4\texttt{x},\texttt{w}_4\texttt{y},\texttt{w}_5\texttt{x},\texttt{w}_5\texttt{y}$

    where
    - `m` and `n` represent the width and height of the grid respectively.
    - `ix` and `iy` represent the x and y starting positions of Iron Man.
    - `tx` and `ty` represent the x and y positions of Thanos;
    - $\texttt{s}_i\texttt{x},\texttt{s}_i\texttt{y}$ represent the x and y position of stone $\texttt{s}_i$.
    - $\texttt{w}_i\texttt{x},\texttt{w}_i\texttt{y}$ represent the x and y position of warrior $\texttt{w}_i$.

    Note that the string representing the grid does not contain any spaces or new lines. It is just formatted this way above to make it more readable. All x and y positions are assuming 0-indexing. *The minimum dimensions of the input grid must be at least* 5×5 *with at least 5 warriors.*
  - *strategy* is a string indicating the search strategy to be applied:
    - `BF` for breadth-first search,
    - `DF` for depth-first search,
    - `ID` for iterative deepening search,
    - `UC` for uniform cost search,
    - `GR`i for greedy search, with $i \in \{1, 2\}$ distinguishing the two heuristics, and

* `AS`i for A* search, with $i \in \{1, 2\}$ distinguishing the two heuristics.

– *visualize* is a boolean parameter which, when set to `true`, results in your program's side-effecting a visual presentation of the grid as it undergoes the different steps of the discovered solution (if one was discovered).

The function returns a `String` of the following format: `plan;cost;nodes` if a solution was found where

– `plan` is a string representing the operators Iron Man needs to follow separated by commas. The possible operator names are: `up, down, left, right, collect, kill,` and `snap`.

– `cost` is a number representing the cost of the found plan.

– `nodes` is the number of nodes chosen for expansion during the search.

If the problem has no solution, the string `There is no solution.` should be returned.

2. Sample Input/Output:

Example Input Grid: `5,5;1,2;3,1;0,2,1,1,2,1,2,2,4,0,4,1;0,3,3,0,3,2,3,4,4,3`

The following is a visualization of the above grid to make things easier for you.

	0	1	2	3	4
0			S	W	
1		S	I		
2		S	S		
3	W	T	W		W
4	S	S		W	

I represents Iron Man, T represents Thanos, S represents a stone, and W represents a warrior. The first row and column just shows the indices and are not part of the grid.

Example Output:
`up,collect,left,down,collect,down,collect,right,collect,kill,down,down,left, collect,left,collect,right,up,snap;63;52333`

This is a breakdown of the computed cost above: `up(1),collect(3+1),left(0),down(0), collect(3),down(5),collect(3+5),right(1),collect(3+1),kill(2),down(5), down(1),left(5), collect(3+5),left(1),collect(3+1),right(5),up(5+1),snap`.

2. Groups: You may work in groups of *at most* three.

3. Deliverables

a) Source code.

- You should implement a data type for a search-tree node as presented in class.
- You should implement an abstract data type for a generic search problem, as presented in class.
- You should implement the generic search procedure presented in class, taking a problem and a search strategy as inputs. *You should make sure that your implementation of search does not allow repeated states and that all your search strategies terminate in under 1 minute.*

- You should implement an EndGame subclass of the generic search problem.
- You should implement all of the search strategies indicated above together with the required heuristics. A trivial heuristic (e.g. $h(n) = 1$) is *not* acceptable. You should make sure that your heuristic function runs in maximum *polynomial* time in the size of the state representation.
- Your program should implement the specifications indicated above.
- Part of the grade will be on how readable your code is. Use explanatory comments whenever possible

b) Project Report, including the following.

- A brief description of your understanding of the problem.
- A discussion of your implementation of the search-tree node class.
- A discussion of your implementation of the search problem class.
- A discussion of your implementation of the EndGame problem.
- A description of the main functions you implemented.
- A discussion of how you implemented the various search algorithms.
- A discussion of the heuristic functions you employed and, in the case of A*, an argument for their admissibility.
- At least two running examples from your implementation.
- A comparison of the performance of the implemented search strategies on your running examples in terms of completeness, optimality, and the number of expanded nodes. You should comment on the differences in the number of expanded nodes between the implemented search strategies.
- Proper citation of any sources you might have consulted in the course of completing the project. *Under no condition*, you may use on-line code as part of your implementation.
- If your program does not run, your report should include a discussion of what you think the problem is and any suggestions you might have for solving it.

4. Important Dates

Teams. Make sure you submit your team members' details by September 26th at 23:59 using the following link `https://forms.gle/2G2Bjc6eVQFHczBD8`. Only one team member should submit this for the whole team.

Source code. Online submission by October 17th at 23:59 using the following link `https://forms.gle/qmeZ6jA4i6M5ctr28` . We will assign IDs to all teams and post them on the MET website one week before the submission deadline. You will need to include your team ID in your submission form.

Brainstorming session. In tutorials.