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

Introduction to Artificial Intelligence, Winter Term 2018
Project 1: Winter has come! [1]

Due: October 18th 2018

1. Project Description

More than eight thousand years ago, the longest winter in history fell on the continent of Westeros. During this dark time, humanoid ice creatures known as the white walkers swept through Westeros riding on their dead horses and hunting everything living. Eventually, the people of Westeros rallied against the white walkers and forced them to the north. A great wall was built to bar the return of the white walkers ever again. The wall stood tall for thousands of years protecting all living things until the white walkers finally succeeded in bringing it down. Now a long and deadly winter has come to Westeros once again.

As a last strive for survival, the King in the North Jon Snow gathered all remaining magic in Westeros to freeze the white walkers in their places in a big field right after they crossed the fallen wall. The white walkers must be exterminated before they find a way to unfreeze bringing unstoppable doom to Westeros. In order not to provoke the white walkers causing them to break free, only Jon Snow must enter the field where they are frozen in place to kill them. White walkers can be killed by stabbing them with a form of volcanic glass commonly known as dragonglass which can be obtained from a special stone called the dragonstone.

In this project, you will use search to help Jon Snow save Westeros. The field where the white walkers are frozen in place can be thought of as an $m \times n$ grid of cells where $m, n \geq 4$. A grid cell is either free or contains one of the following: a white walker, Jon Snow, the dragonstone, or an obstacle. Jon Snow can move in the four directions as long as the cell in the direction of movement does not contain an obstacle or a living white walker. To obtain the dragonglass by which the white walkers can be killed, Jon has to go to the cell where the dragonstone lies to pick up a fixed number of pieces of dragonglass that he can carry. In order to kill a white walker, Jon has to be in an adjacent cell. An adjacent cell is a cell that lies one step to the north, south, east, or west. With a single move using the same piece of dragonglass, Jon can kill all adjacent white walkers. If Jon steps out of a cell where he used a piece of dragonglass to kill adjacent walkers, that piece of dragonglass becomes unusable. Once a white walker is killed, Jon can move through the cell where the walker was. If Jon runs out of dragonglass before all the white walkers are killed, he has to go back to the dragonstone to pick up more pieces of dragonglass. Using search you should formulate a plan that Jon can follow to kill all the white walkers. An optimal plan is one where Jon uses the least number of pieces of dragonglass to kill all the white walkers. The following search algorithms will be implemented and each will be used to help Jon:

[1]This project's theme is based on Game of Thrones.

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 may use the programming language of your choice.

Your implementation should have two main functions `GenGrid` and `Search`:

- `GenGrid()` generates a random grid. Jon Snow always starts at the bottom right cell of the grid. The dimensions of the grid, the number of dragonglass pieces Jon is allowed to carry, as well as the locations of white walkers and the dragonstone are to be randomly generated. You need to make sure that the dimensions of the generated grid are at least 4×4.
- `Search(`*grid*`, `*strategy*`, `*visualize*`)` uses search to try to formulate a winning plan:
  - *grid* is a grid to perform the search on,
  - *strategy* is a symbol 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 `t`, 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 list of three elements, in the following order: (i) a representation of the sequence of moves to reach the goal (if possible), (ii) the cost of the solution computed, and (iii) the number of nodes chosen for expansion during the search.

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 implement a SaveWesteros 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 ADT.
- A discussion of your implementation of the search problem ADT.
- A discussion of your implementation of the SaveWesteros 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.
- A list of complete and precise instructions on how to run the program and interpret the output.
- 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 27th at 23:59 using the following link `https://goo.gl/forms/T5nCrwD7xu9WW8Rw1`. Only one team member should submit this for the whole team.

Source code. Online submission by October 18th at 23:59 using the following link `https://goo.gl/forms/DAJ6Yu9f8VAMJ1uG2` . 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.