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

Introduction to Artificial Intelligence, Winter Term 2016
Project 1: Gotta Catch'em All

Due: October 19th at 23:59

1. Project Description

In this project, you will implement a search agent to play a maze-based Pokémon game. The maze is an $m \times n$ grid with Pokémons hiding at random locations along the maze. The agent starts at a random starting point in the maze with an egg containing a special rare pokémon. The agent can move one step forward (if there is no wall infront of it), and rotate right and left. For the egg to hatch, the agent should move at least x time units. A time unit is defined as the time needed to move forward. The agent catches a pokémon when it moves into the same cell where the pokémon hides. The objective of the game is to reach the end point in the maze after hatching the egg and collecting all the hiding pokémons in the maze.

Using search you should formulate a plan that the agent can follow to win the game, or announce that there is no solution. An optimal plan is one with as few steps as possible. Several algorithms will be implemented and each will be used to play the game:

a) Breadth-first search.
b) Depth-first search.
c) Iterative deepening search.
d) Uniform-cost search.
e) Greedy search with at least three heuristics.
f) A* search with at least three 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 `GenMaze` and `Search`:

- `GenMaze()` generates a random maze[1]. The locations of the pokémons, the start and end points, and the number of time units required to hatch the special pokémon egg are to be randomly generated.
- `Search(`*maze*`,` *strategy*`,` *visualize*`)` uses search to try to formulate a winning plan:
  - *maze* is a maze to perform the search on,
  - *strategy* is a symbol indicating the search strategy to be applied:

[1]Take a look at `https://en.wikipedia.org/wiki/Maze_generation_algorithm` for various maze generation algorithms.

    - * `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 $i \in \{1, 2, 3\}$ distinguishing the three heuristics, and
    - * `AS`i for A* search, with $i \in \{1, 2, 3\}$ distinguishing the three heuristics.
  - *visualize* is a Boolean parameter which, when set to `t`, results in your program's side-effecting a visual presentation of the maze 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 an abstract 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 algorithm presented in class, taking a problem and a search strategy as inputs.
- You should implement a Gotta Catch'em All instance/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.
- 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 Gotta Catch'em All 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.
- A comparison of the performance of the different algorithms implemented in terms of completeness, optimality, and the number of expanded nodes.
- 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

Source code. On-line submission by October 19th at 23:59 using the "Submission" link on the course web site.

Project Report. A hard-copy should be submitted. You have two options:

a) October 19th by 16:00;

b) October 20th by 16:00 provided that an *identical* on-line version is submitted with the code (by October 19th at 23:59).

Brainstorming Session. In tutorials.