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

Introduction to Artificial Intelligence, Winter Term 2022
Project 1: Coast Guard

Due December 2nd by 23:59

1. **Project Description** You are a member of the coast guard force in charge of a rescue boat that goes into the sea to rescue other sinking ships. When rescuing a ship, you need to rescue any living people on it and to retrieve its black box after there are no more passengers thereon to rescue. If a ship sinks completely, it becomes a wreck and you still have to retrieve the black box before it is damaged. Each ship loses one passenger every time step. Additionally, each black box incurs an additional damage point every time step once the ship becomes a wreck. One time step is counted every time an action is performed. You reach your goal when there are no living passengers who are not rescued, there are no undamaged boxes which have not been retrieved, and the rescue boat is not carrying any passengers. You would also like to rescue as many people as possible and retrieve as many black boxes as possible.

   The area in the sea that you can navigate is an $m \times n$ grid of cells where $5 \leq m, n \leq 15$.

   The grid elements are restricted to the following.

   - **Coast Guard Boat:** The coast guard boat is the agent; *it is the only element that can move* on the grid. The coast guard can enter any cell. It has a fixed capacity $30 \leq c \leq 100$ which indicates the number of passengers it can carry at one time. So it may have to make multiple visits to one ship to save all the passengers on it.
   - **Ships:** Each ship has a certain number of passengers and, at every time step, one of them expires. The ship is considered sunk when all of them are expired or all are picked up In that case, it becomes a wreck. Each ship contains a black box that can be retrieved after it sinks as long as it has not been completely damaged.
   - **Wrecks:** Once the ship no longer has any passengers (all expire or all are picked up), it becomes a wreck. When the ship becomes a wreck (once its last passenger either expires or is picked up), in the next time step, the black box starts counting damage from 1 all the way up to 20. Once damage reaches 20, the black box is no longer retrievable.
   - **Stations:** Stations are fixed and do not have any capacity limits. To count a passenger saved, they need to be dropped at a station.

   Initially, the grid is configured as follows.

   - The coast guard boat is at a random location
   - Several ships are scattered at random locations, and each has a random initial number of passengers p, where $0 < p \leq 100$.
   - Several stations are at random locations.
   - There are no wrecks.
   - No two items are in the same cell.

The coast guard can perform the following actions.

- **Pick-up:** The coast guard picks up as many passengers off a ship as its remaining capacity allows. This can be done on a ship that is in the same cell as the coast guard and it only affects this ship. Once a passenger is picked up by the coast guard, they will not expire and will stay on the coast guard boat until they are dropped at a station.
- **Drop:** The coast guard drops all passengers it is currently carrying at a station. This can only be done when the coast guard and the station are in the same cell and it resets the remaining capacity of the coast guard boat to 0.
- **Retrieve:** The coast guard boat retrieves a black box. This can only be done when the coast guard boat is in the same cell as a wreck with a black box which has not been completely damaged yet. This action does not affect the coast guard's remaining capacity at all.
- **Movement in any of the 4 directions** (up, down, left, right) within the grid boundaries.

In this project, you will create an agent that acts as the coast guard boat captain using search. The agent has succeeded when there are no more un-expired passengers to save and no more undamaged black boxes to retrieve. Optimally, you would like to minimise the number of expired passengers and secondarily retrieve as many black boxes 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) Greedy search with at least two heuristics.

e) A* search with at least two *admissible* heuristics. A trivial heuristic (e.g.$h(n) = 1$) is not acceptable.

Different solutions should be compared in terms run-time, number of expanded nodes, memory (RAM) utilisation and CPU utilisation. **You are required to implement this agent using Java.**

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

- `GenGrid()` randomly generates a grid. The dimensions of the grid, the locations of the coast guard, stations, ships, number of passengers for each ship and the coast guard capacity are all to be randomly generated. A minimum of 1 ship and 1 station have to be generated. However, there is no upper limit for the number of ships or stations generated as long as no 2 objects occupy the same cell. `GenGrid()` should return a string representation of a grid; the format of the string is indicated below.

- `Solve(`*grid*`,`*strategy*`,` *visualize*`)` uses search to find a sequence of steps to rescue the passengers and retrieve the black boxes.
  - `grid` is a grid to perform the search on; it is passed as an input in the following format.
    $M, N; C; cgX, cgY;$
    $I_1X, I_1Y, I_2X, I_2Y, ...I_iX, I_iY;$
    $S_1X, S_1Y, S_1Passengers, S_2X, S_2Y, S_2Passengers, ...S_jX, S_jY, S_jPassengers;$
    where
    * M and N represent the width and height of the grid respectively.
    * C is the maximum number of passengers the coast guard boat can carry at time.
    * cgX and cgY are the initial coordinates of the coast guard boat.
    * I_iX, I_iY are the x and y coordinates of the i^{th} station.
    * S_jX, S_jY are the x and y coordinates of the j_{th} ship.
    * $S_jPassengers$ is the initial number of passengers on board j_{th} ship.

    Note that the string representing the grid does not contain any spaces or new lines. It is just formatted this way to make it more readable. All x and y positions assume 0-indexing.
  - `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,
    * `GR`i for greedy search, with $i \in \{1, 2\}$ distinguishing the two heuristics.
    * `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 board as it undergoes the different steps of the discovered solution (if one was discovered). *A GUI is not required, printing to the console would suffice.*

  The function returns a `String` of 4 elements, in the following format:
  `plan;deaths;retrieved;nodes`
  where:
  - `plan`: the sequence of actions that lead to the goal (if such a sequence exists) separated by commas. For example: `left,right,pickup,up,drop,down,retrieve`. (These are just all of the possible operator names.)
  - `deaths`: number of passengers who have died in the solution starting from the initial state to the found goal state.
  - `retrieved`: number of black boxes successfully retrieved starting from the initial state to the found goal state.
  - `nodes`: is the number of nodes chosen for expansion during the search.

2. **Examples:**

   - Grid generation string: 3,4;97;1,2;0,1;3,2,65;
   - Grid generated by the above string:

	Station	
		Coast Guard
		Ship(65)

   - An example plan that could be devised by the agent that leads to the goal for the above grid: down,down,pickup,retrieve,up,up,up,left,drop
   - number of deaths if that plan is executed: 2
   - number of black boxes retrieved if that plan is executed: 1

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

4. **Deliverables**

   a) Source Code

      - You should implement an abstract data type for a search-tree node as presented in class. (*This does not necessarily mean that your implementation should include Java abstract classes.*)
      - 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 minimises or eliminates redundant states and that all your search strategies terminate within the time limits of the automated public test cases that will be posted.
      - You should implement a CoastGuard subclass of the generic search problem. This class must contain `genGrid()` and `solve` as static methods.
      - You should implement all of the search strategies indicated above (together with the required heuristics).
      - Your program will be subject to both public and private test cases.
      - Your source code should have two packages. A package called `tests` and a package called `code`. All your code should be located in the `code` package and the test cases (when posted) should be imported in the `tests` package.
      - 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 discussion 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 CoastGuard 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 greedy or A*, an argument for their admissibility.
- A comparison of the performance of the different algorithms implemented in terms of completeness, optimality, RAM usage, CPU utilization, and the number of expanded nodes. You should comment on the differences in the RAM usage, CPU utilization, and 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.
- If you use code available in library or internet references, make sure you *fully* explain how the code works and be ready for an oral discussion of your work.
- 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.

5. Important Dates

Team Submission Make sure you submit your team member details by October 31st at 23:59 using the following link `https://forms.gle/Y4Dw7V1hVAEvfQDDA`. Only one team member should submit this for the whole team. After this deadline, we will be posting on the CMS a team ID for each submitted team. You will be using this team ID for submission.

Source code and Project Report On-line submission by December 2nd at 23:59. The submission details will be announced after the team submission deadline.

Brainstorming Session. In tutorials.