**German University in Cairo**
**Department of Computer Science**
**Assoc. Prof. Haythem O. Ismail**
**Eng. Nourhan Ehab**
**Eng. Ammar Yasser**

**Introduction to Artificial Intelligence**, Summer Term 2021
# Project The Artificial Olympics

*Due: August* $20^{th}$

## 1. Project Description

The artificial Olympics are similar to the traditional Olympic games except that the participants are artificial robots. One robot in particular is of interest to the organizing committee. The robot is named *Jarvis*. *Jarvis* is tasked with the most important task in the Olympics: lighting the Olympic flame. The artificial Olympic flame is lighted when all the necessary components are gathered then brought to the location of the flame. The Olympic arena hosts all athletes, the location of the flame, as well as all the components that are necessary to create the flame. All of this is well and fine except that there is a group of conspirators that want the Olympics to fail and hence they want to spoil the flame lighting ceremony. They wanted to spray the arena with a poisonous material that will damage anyone that gets in contact with it. Luckily, they were not able to spray the entire arena but only parts of it.

The arena can be thought of as an $m \times n$ grid of cells where $5 \leq m, n \leq 15$. Initially, a grid cell is either free, poisonous (in which case *Jarvis* can not enter this cell), or contains one of the following: *Jarvis*, a component for the flame, or the flame location. In this project, you will use *search* to help *Jarvis* complete the task by finding all the components and then going to the flame location to light the flame. *Jarvis* can move in all four directions, can pick up a component only if both of them are in the same cell, and can light the flame if it is at the same cell where the flame location is and all components are with it. Using search you should formulate a plan that *Jarvis* can follow to light the flame. An optimal plan is one where the moves are at a minimum. The following search strategies will be implemented and each will be used to help *Jarvis*:

a) Breadth-first search.
b) Depth-first search.
c) Uniform-cost search.
d) Greedy search with at least two heuristics.
e) A* search with at least two *admissible* heuristics.

Each of the aforementioned strategies should be tested and compared in terms of RAM usage, CPU utilization, the optimality of the output, and the number of search tree nodes expanded. **You must only use Java to implement this project.**

Your implementation should have two main functions `genGrid` and `solve`:

- `genGrid()` generates a random grid. The dimensions of the grid, the starting position of *Jarvis* and the flame location, as well as the number and locations of the poisonous cells and the components are to be randomly generated. You need to make sure that the dimensions of the generated grid is between $5 \times 5$ and $15 \times 15$, the number of poisonous cells is between 2 and 25, and the number of generated components is between 5 and 10.

- `solve(String grid, String strategy, boolean 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; jx,jy; fx,fy;`
    xc$_1$,yc$_1$, ...,xc$_k$,yc$_k$;
    xp$_1$,yp$_1$, ...,xp$_z$,yp$_z$;
    where:
    - ∗ `m` and `n` represent the width and height of the grid respectively.
    - ∗ `jx` and `jy` represent the x and y starting positions of *Jarvis*.
    - ∗ `fx` and `fy` represent the x and y positions of the flame.
    - ∗ xc$_i$,yc$_i$ represent the x and y position of component $i$ where $1 \leq i \leq k$ and $k$ is the total number of components.
    - ∗ xp$_i$,yp$_i$ represent the x and y position of poisonous cell $i$ where $1 \leq i \leq z$ and $z$ is the total number of poisonous cells.

    **Note that the string representing the grid does not contain any spaces or new lines. It just formatted this way above to make it more readable. All x and y positions are assuming 0-indexing.**
  - *strategy* is a symbol indicating the search strategy to be applied:
    - ∗ `BF` for breadth-first search,
    - ∗ `DF` for depth-first 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;nodes` where
  - `plan` is a string representing the operators *Jarvis* needs to follow separated by commas. The possible operator names are: `up, down, left, right, pick` and `light`.
  - `nodes` is the number of nodes chosen for expansion during the search.

**2. Sample Input/Output:**

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

| | | | | |
|---|---|---|---|---|
| | P | P | C | |
| | | J | | |
| | C | | | |
| | P | F | P | C |
| C | | P | C | |

J represents *Jarvis*, F represents the flame, P represents a poisonous cell and C represents a component.

**Example Output:**
right,up,pick,down,left,left,down,pick,left,down,down,pick,up,up,right,right,right,right, down,pick,down,left,pick,right,up,up,left,left,down,light;1165836

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

4. **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 `Olympics` 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. 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 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 Olympics 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, 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. *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.

## 5. Important Dates

**Source code and Report.** Online submission by August 20th at 23:59. The submission details will be announced after the team submission deadline.

**Brainstorming session.** In tutorials.