Artificial Intelligence, Summer Term
Project 1: United We Stand

1. Project Description A number of micro-organisms inhabit a rectangular grid of squares. Each micro-organism occupies a single square of the grid. The grid is surrounded by harmful chemical deposits. The organisms are controlled by a single station outside the grid. If two organisms occupy contiguous squares, then they fuse into a single, larger organism. Each of the organisms can move as per instructions issued by the controller. The controller issues instructions of the form $\langle o,d \rangle$, where o is an organism and $d \in \{North,South,East,West\}$, indicating that organism o is to move in direction d. The motion proceeds continuously in the indicated direction and ends in one of the following ways.

    a) o collides with another organism. In this case, motion stops and the two organisms fuse into a single new organism.
    b) o collides with an obstacle. In this case, motion simply stops.
    c) o collides with the chemical boundary. Such collision causes major damage to o, resulting in failure.

    In this project, you will create part of the controller which is designed to create a plan for the micro-organisms to fuse into a single organism, if possible. The plan will be formulated using search. 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. The cost of the movement of a part p is given by $\sum_{m \in M(o)} s(u)$, where $M(o)$ is the set of micro-organisms making up o and $s(m)$ is the number of squares which micro-organism m moves across.

    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 the following classes:

    - GenericSearch, which has the generic implementation of a search problem (as defined in Lecture 2).
    - UnitedWeStandSearch, which extends GenericSearch, implmenting the Ünited We Standßearch agent.
    - Node, which implements a search-tree node (as defined in Lecture 2).

    You can implement any additional classes you want. Inside UnitedWeStandSearch you will implement genGrid and solve as the key function which will be the basis for testing:

    - genGrid() randomly generates a grid. The dimensions of the grid, the locations of the micro-organisms, and the locations of obstacles are all randomly generated. The returned grid should be a string in the same format described below.

- Search(*grid*,*strategy*, *visualize*) uses search to try to fuse the micro-organisms into a single organism:
  - *grid* is the initial configuration of the grid to perform search on in terms of grid dimensions, obstacle locations and organism locations (initially all organisms are of the same size of 1). This information will be represented as a string formatted as follows:

    $Width$;$Height$;
    $organismX_1$,$organismY_1$,$organismX_2$,$organismY_2$,...$organismX_i$,$organismY_i$;
    $obstacleX_1$,$obstacleY_1$,$obstacleX_2$,$obstacleY_2$,...$obstacleX_i$,$obstacleY_i$;

    where $organismX_i$ and $organismY_i$ represent the horizontal and vertical coordinates of the *i*th organism respectively. And $obstacleX_i$ and $obstacleY_i$ represent the horizontal and vertical coordinates of the *i*th obstacle. The actual provided string will have no spaces or new lines, they are just added here for readability. The grid coordinates start from (X=0, Y=0) which is located at the top-left corner. X increases as we go right (east) and Y increases as we go down (south). The with and height have an upper limit of 10 and the number of organisms are at most 20.
  - *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, $*$ ID for
    iterative deepening 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 board 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. The return string should follow the format of the following example:

  Example: WEST_2_1,NORTH_0_2;3;7

  Where the action WEST_2_1 means the organism at cell (2, 1) was moved west. The cost is 3 and the number of expanded nodes is 7.

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 united-we-stand instance/subclass of the generic search problem. This class must contain solve as a static method.
- You should implement all of the search strategies indicated above (together with the required heuristics).
- No additional assumptions should be made about the initial state.
- Your program should implement the specifications indicated above.
- 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 united-we-stand problem ADT.
- 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.
- 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.

4. Important Dates

Source code. On-line submission by July 26 at 23:59. Directions for submissions will be posted.

Project Report. You should submit a soft copy of the report with the code.

Brainstorming Session. In tutorials.