Assignment 4

Objective

Practice writing unit tests

Practice writing GUI wrappers for an application with core logic separated from the user interface

Practice effectively using data structures in algorithm implementations

Practice pair programming as a component of XP

Practice Maven as build tool

Gain more experience with object oriented design.

Marking

This assignment will be graded as follows:

- 30% of the grade will be computed based on the quality of test cases that you will write and submit, namely:
  - Did you create at least one test case for each relevant method? (we do understand that it is probably an overkill to test a simle getter and/or a simple setter)
  - How thorough is your testing? Did you account for all categories of values that your method parameters can take?
  - Did you test for edge cases?
- 5% of the grade will come from correctly preparing the `pom.xml` to include all necessary dependencies and project properties.
- 10% of the grade will come from the code inspection by our TAs to make sure you have properly implemented the A-star path searching algorithm.
- 25% of the grade will come from a well designed GUI.
- 30% of the grade will come from our autotesting.

It is worth 15% of your final grade.

The submission deadline is Feb 19, 2017, 11:59pm on Github.

It is highly recommended to work in groups of two in order to practice XP.

How to submit your work

1. As usual, please do use this invitation: `https://classroom.github.com/group-assignment-invitations/1f3c4e0c1a621b85`
2. Once you have cloned your project, make sure to run an `mvn clean` in the folder `A4` (where `pom.xml` is).
3. Make sure to update `readme.md` with the names and student IDs of the participants of your group.
4. Also make sure maintain the `.gitignore` so you do not submit unncessary items or files that you might have used for testing.

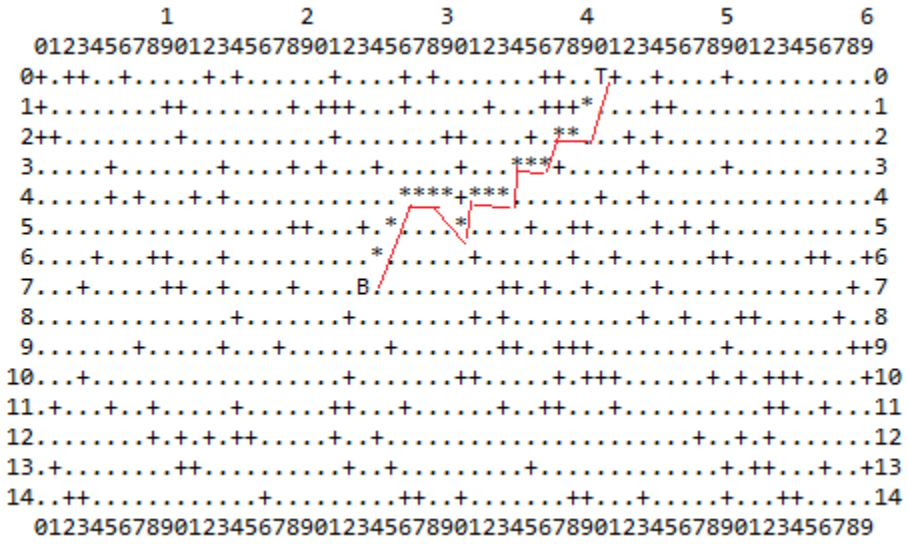
The Redbeard Treasure Hunt Game

The famous pirate Redbeard master ship sunk in 17-th century in the Sea Of 1000 Islands after a fierce battle with Bluebeard's ship. The sunken ship contains a large bounty. You are navigating the Sea Of 1000 Islands in your ship provided with modern technology. The Sea Of 1000 Islands has been mapped into a grid where a navigable square is denoted by a dot ("."), and a square that belongs to one of many islands is denoted by a "+".

You have a number of sonar devices. If you drop a sonar device from your ship, if the treasure is in the range of sonar, the device will give you the grid coordinates of the treasure. Once you have the coordinates, your AI engine will run a clever algorithm, called A-star search algorithm, that is capable of plotting the shortest way from the current position of your ship to the detected treasure avoiding the obstacles (in this case, the obstacles are the numerous islands).

If the sonar device detects nothing, you can navigate in any direction you like (north, south, east, west, north-east, north-west, south-east, south-west) one navigable square at a time. You can move as many squares as you like. Whenever you feel like it, you drop another sonar. And so on until the treasure has been found, or you run out of the sonars. Please note once a sonar has been dropped, it is not recoverable. The goal of the game is to find the treasure and plot the navigation route to the treasure. If you run out of the sonar devices before finding the treasure, you lose the game.

A copy of the map, including the plotted trip to the treasure, is shown below. The dots indicate navigable squares, the "+" signs indicate obstacles, the letter "B" indicates the position of the ship, the letter "T" indicates the position of the treasure. The sequence of "*" signs indicates the path plotted by the A-star search algorithm. The jagged red line just follows the "*" symbols to emphasize the path.

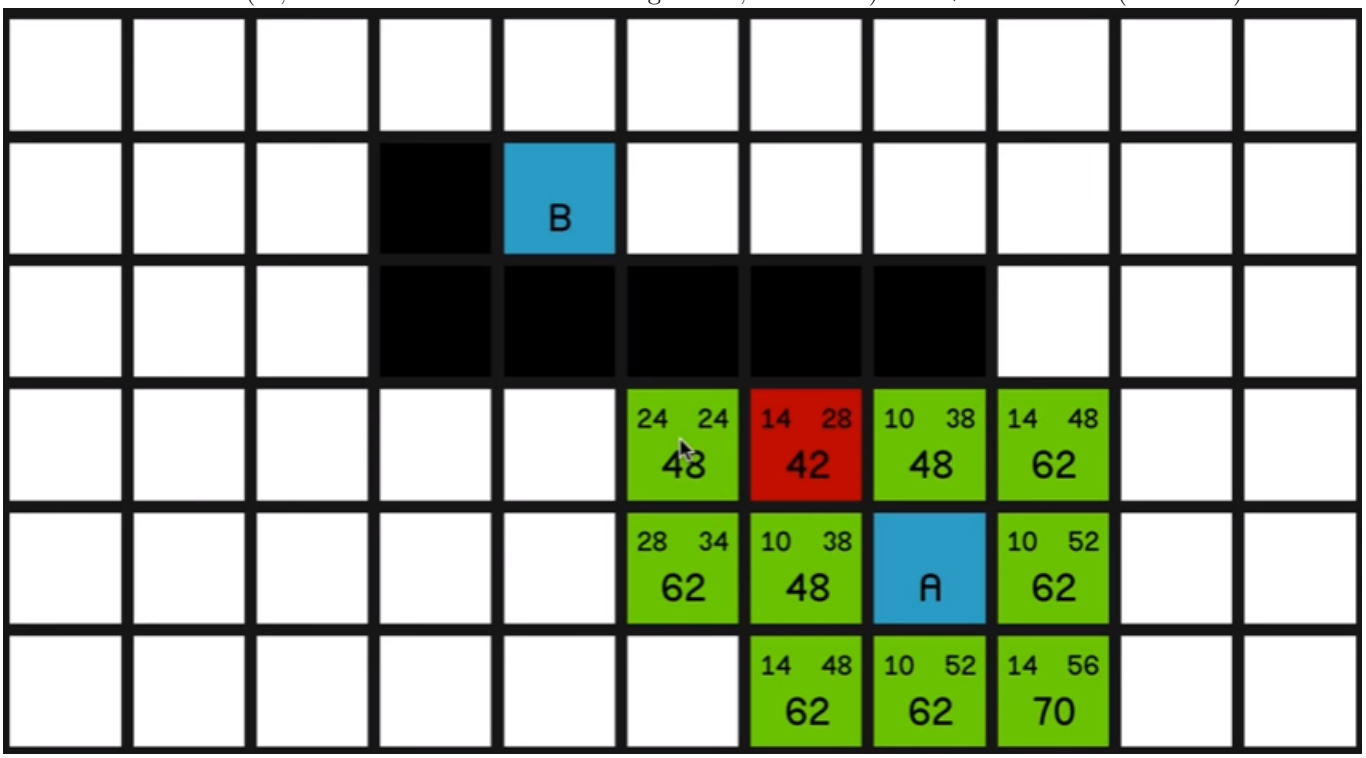


The A-star path search algorithm

The A-star path search adjoins next cell to the path by chosing a cell that is "open" (see the links below) and satisfies the following condition: the sum of distances of that cell from the origin and the destination must be minimal. The details of the algorithm including the pseudocode can be found on the internet. Here is a couple of recommended links:

- `http://web.mit.edu/eranki/www/tutorials/search/`
- `http://www.policyalmanac.org/games/aStarTutorial.htm`

The next image shows how the actual sum of distances is computed. Please note if we assume one cell has side length one unit, a vertical or horizontal distance from one cell to next is one unit, whereas a diagonal distance (for example from current cell to the cell loacted north-east) is $\sqrt{2}$ or approx. 1.4 units. In order to avoid decimals from or computation, we will multiply everything by 10, so for instance the distance of the cell painted in red is computed as follows: the red cell is one (diagonal) square far from cell labelled "A", that is 14 unit, and two (diagonal) squares from cell labelled "B" therefore 28 units, so its total distance (or, as it is called in the A-star algorithm, its **f-cost**) is 14+28=42 units (as shown).



Please note on each iteration we need the open cell with minimal f-cost (please see the pseudocode in the MIT weblink above). To avoid a costly search on each iteration, we will maintain the list of the open cells using a data structure called **heap** implemented using a complete binary tree stored in an array.

Heaps

Heaps are complete binary trees that satisfy these conditions:

1. The smallest value (or highest priority) element sits at the root.
2. Every subtree of a heap is also a heap.

Since a heap is a complete tree, all its levels but the last are complete. The last level may miss some leaves in the rightmost position. An example of a heap is shown in the next image.

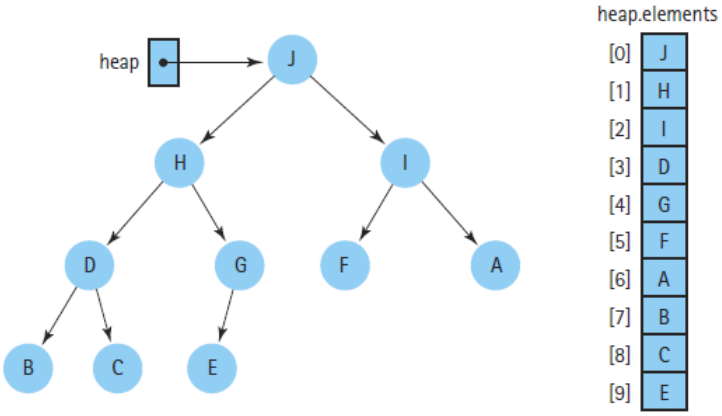
Observe that the root (that is the lowest value element - in that case the letter "J" of course assuming the highest value letter of the alphabet is "A" and the lowest value letter of the alphabet is "Z") is mapped (in the implementation) to the lement with index 0 of the array to the right. Its immediate children occupy the cells 1 and 2 of the array. The children of the level 1 occupy elements 3,4,5,6. And so son.

When we use this representation of a binary tree, the following relationships hold for an element at position index:

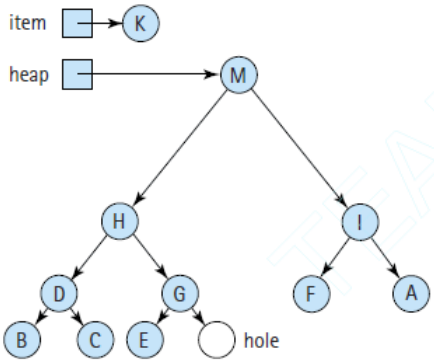
- If the element is not the root, its parent is at position (index 1) / 2.

- If the element has a left child, the child is at position (index * 2) + 1.
- If the element has a right child, the child it is at position (index * 2) + 2.

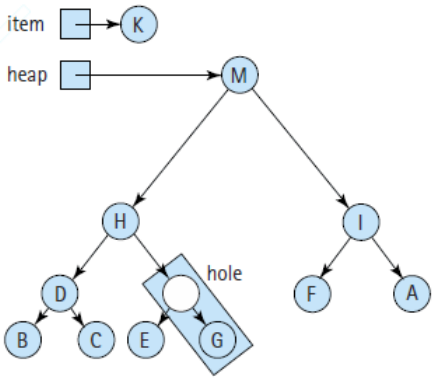
These relationships allow us to efficiently calculate the parent, left child, or right child of any node! And since the tree is complete we do not waste space using the array representation. Time efficiency and space efficiency! We make use of these features in our heap implementation.



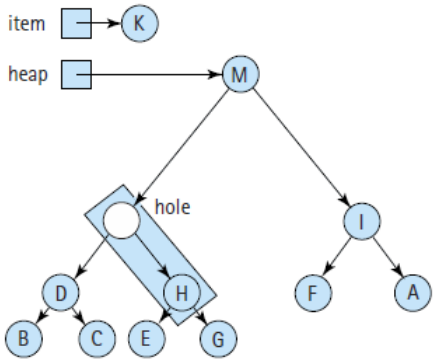
Here is how we add an element to the heap (in this case letter "K"). We add it at the bottom rightmost position, and sort it up (sortUp in the starter code) to the correct position.

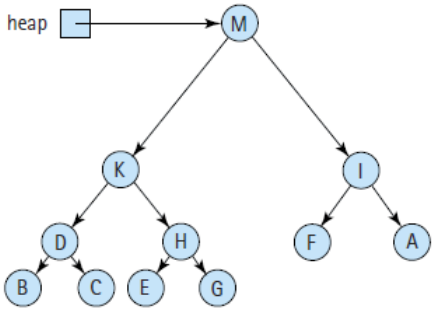


(a) Add K



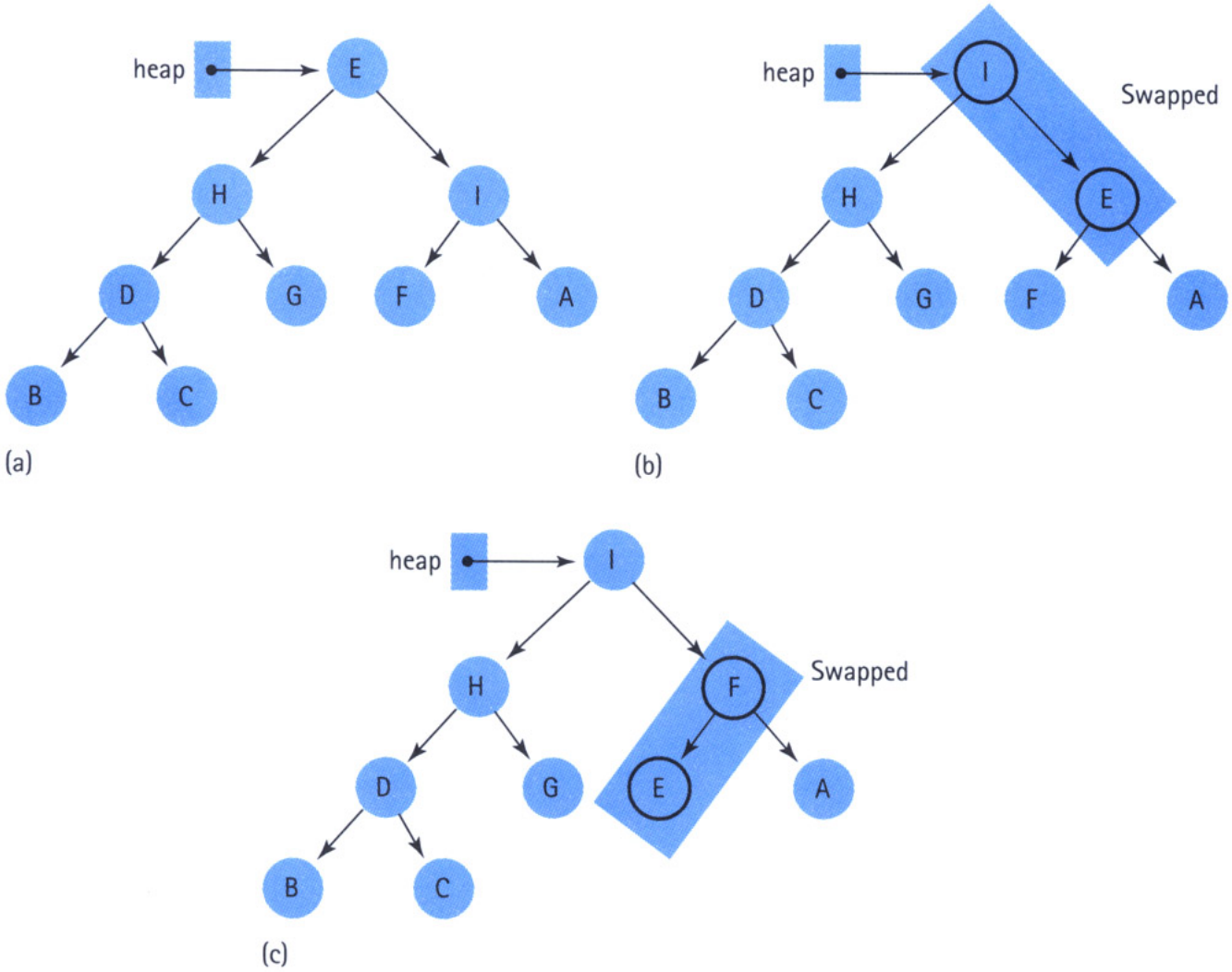
(a) Move hole up





Removal of the top element (remember - we only remove the top element - highest priority (in our case

lowest f-cost!) is done by substituting it with the rightmost element and then sorting it down. The example below illustrates the removal of topnode "J" (see the first image in the previous page), replacing it by "E" (the rhightmost element in the lowest level) and moving the element "E" down to is right place:



All these manipulations rely on the fact that we are able to **compare** the information stored in the nodes of the heap. That is each node should implement the comparable interface.

The starter code

Please study the starter code carefully. It contains a lot of comments explaining many aspects of your coding work. Make sure to implement heap carefuly and use a heap object of this implentation to maintain the set of open cells.

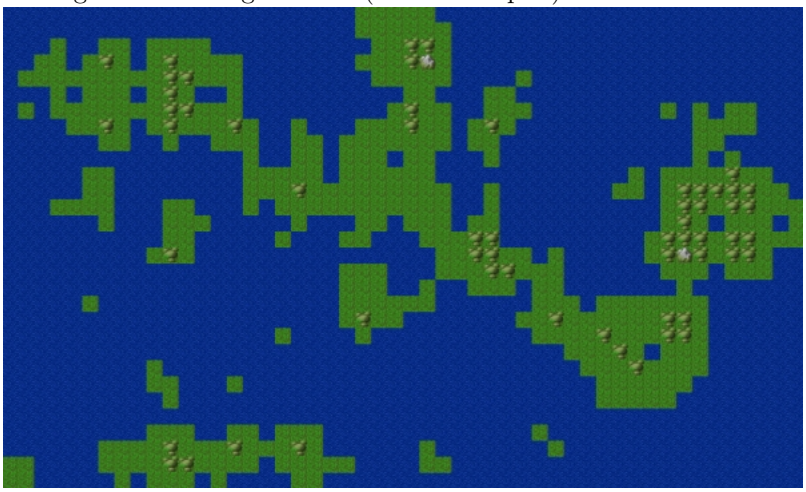
Each cell of the grid will be represented by a `Node` object. A `Node` object has grid coordinates, and also can be `walkable` or not (that is can be an obstacle or not), it has a parent (that is a prior cell in the A-star path) and can be a member of the A-star path or not, as determined by the A-star algorithm.

The `world` main attribute is a `map` as described above. In order to find and plot the A-star path, we need a heap to maintain the open cells. As such we will implement a heap, where each heap item implements the `HeapItem` interface.

The `TreasureHunt` class sets up the game. Each game can have the status "STARTED" or "OVER". Once the game is "OVER" we determine if the player won by checking the path. The game can be played from keyboard, however for the purpose of this assignment, we will allow commands of a play session to be read from a text file (a sample has been provided with the starter code). The `GameTest` class is optional -it has been provided in the starter code for your convenience so you can test the game before submitting it.

GUI

The Graphical User Interface must display an image map that shows a grid where the cells of the grid representing the sea are blue and the cells of the grid that represents the islands are a different color (mabe green but we will trust this detail to your design capabilities). The Redbeard's ship should be represented with a distinguished color or other small image that fits in a single grid.
Yes, we do understand that generating a world map is not something you do every day, however we are looking for something like this (or even simpler):



(Source: `http://stackoverflow.com/questions/2520131/looking-for-a-good-world-map-generation-algorithm`)
A tutorial on drawing in Java can be found here:
`https://docs.oracle.com/javase/tutorial/2d/images/drawimage.html`
When the game starts,the player must be presented with the map, and must able to set up the number of available sonars and the initial position of Redbeard's ship (simply by indicating the row and column number; of course you may make it fancier if you like). That means it is probably a good idea to draw the coordinates so player knows where to put the ship; of course a solution using the mouse clicks is nice, but it may be harder to implement.
The GUI window must contain appropriate buttons to allow the player navigate North/South/East/West and also a button that allows to drop the sonar. Once the sonar is dropped and it has detected the treasure, the path (generated by A-star search alogorithm) must be plotted on the map, a winning message must be displayed, and a small pop-up must sho up asking the player if the player wants to play again. In case the player runs out of sonars before finding the treasure, he/she must get the "Game Over" message and a pop-up asking if the payer wants to play again.

Pair Programming

In this assignment it is highly recommended to use and practice pair programming where one member plays the role od the driver and the the role of the navigator, switching roles frequently. You may find more information about pair progrmaming in this article:
`https://en.wikipedia.org/wiki/Pair_programming`

Sample run (using the provided game.txt file)

```
C:\UofT\csc301\maven\a4\A4>java -cp target/A4-1.0-SNAPSHOT.jar org.csc301.GameTest
               1         2         3         4         5         6
       0123456789012345678901234567890123456789012345678901234567890123456789
      0...+..........+...+....+.+........++....+.....++.........+.0
      1..+..........+.++...+............+...+...+..+...+.+++.++.+1
      2+....+.+......+....+.............+..+...+.....+.......+....2
      3..B+++..+......+..+.++.++.++.+...+............+............3
      4.*+....+.........+..............+...++......+...+.+.....+..4
      5.*.+..++............+.+.+..+.......+.............+.+.+..+..5
      6.*..++.++..+...+...+.++++...++..+.+.++...........+......+...6
      7.*...+........+.++...+++.......+.........+++..++.++.+.+..++.7
      8.*....+..++......++..............+.............+++........8
      9+*......+..................++.........+........+..++..+.....9
     10+T...........+..++..+.+..............+....+..........++...10
     11..+.....................+...++++..++.+....+.+...........+.11
     12..+......+...+..+...+++.................+..+.....++++..+....12
     13+....+.+.+....+..........+..+........+..........+..+.....+.+13
     14.....+.....+....+..+.............+..+...........++.+.+.+..14
       0123456789012345678901234567890123456789012345678901234567890123456789

OVER
7
```