Assignment 1

1 Learning Objectives

By the end of this assignment you should have:

- become fluent with the mechanics of class definitions in Python, including constructors and other special methods
- practised implementing a class, given its interface
- practised identifying suitable classes, given the description of a problem in English
- faced decisions about which classes should be responsible for what, and made reasonable choices
- faced decisions about what data structure to use to implement a class, weighed some options, and made reasonable choices
- implement and use an ADT to solve a problem

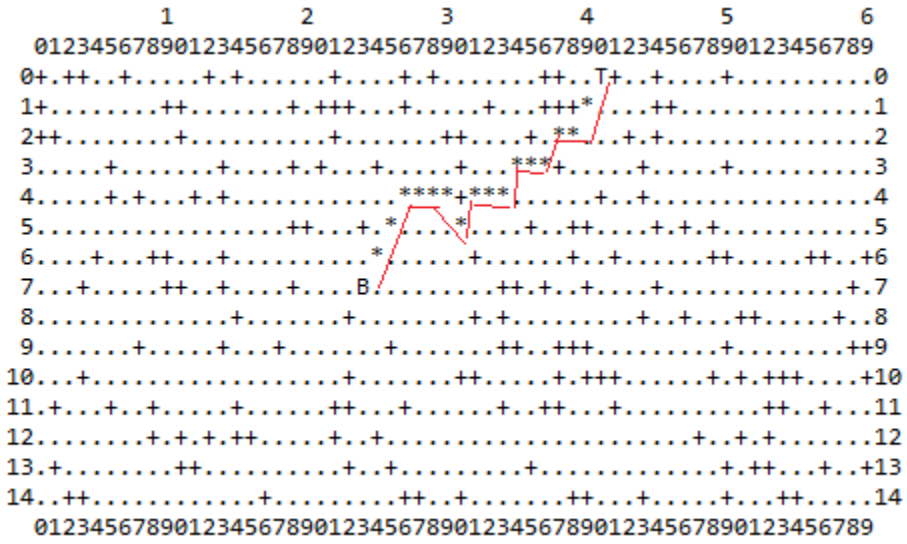
2 The Redbeard Treasure Hunt Game

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 (one square at a time). 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.

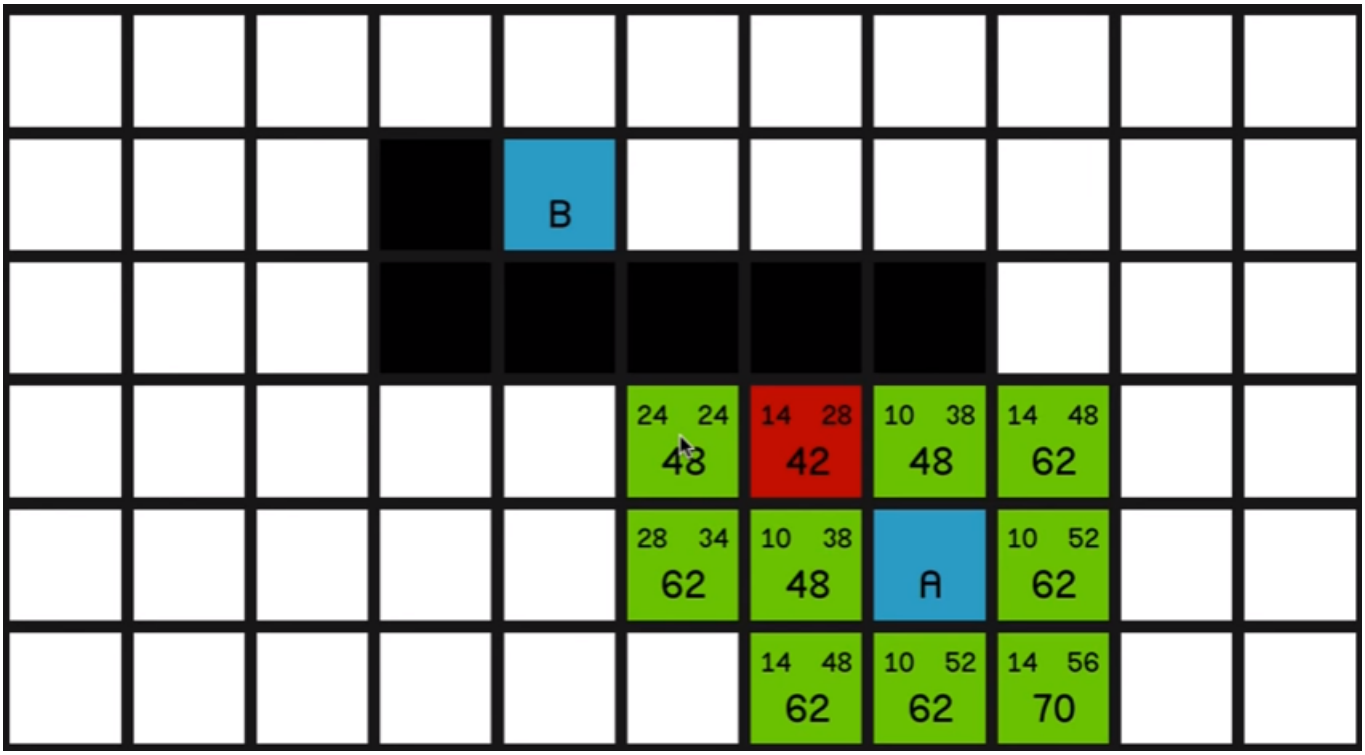


3 The A-star path search algorithm

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 **priority queue**. In fact, this is not very efficient. We will learn later that the best structure use for this purpoise is a **heap**.

4 Priority Queue

In the A-star path search algorithm, you need to maintain a set of open cells and chose the one with minimum f-cost. While we could use a Python list, it has no way to give us items in order according to our priority (f-cost).

Instead we will use something called a PriorityQueue. It is a kind of container that allows you to add and remove items. But a PriorityQueue removes items in a very specific order: it always removes the item with the highest priority. We define what the priority is to be when we construct the PriorityQueue.

A PriorityQueue supports the following actions:

determine whether the priority queue is empty insert an item with a given priority remove the item with the highest priority Look at container.py, which contains the Container class, as well as a new, partially-complete PriorityQueue class. PriorityQueue is a child class of Container, which means that it inherits attributes and methods from Container.

You must complete the required methods according to the provide docstring. (Notice that we're using a sorted list to implement this class; in later courses, you'll learn about a much more efficient implementation called heaps.)

5 The Node, Grid, and TreasureHunt

Please study the starter code carefully. It contains a lot of comments explaining many aspects of your coding work.

The `Node` class is used to represent grid node. Each cell of the grid will be represented by a `Node` object. A `Node` object has grid coordinates, and also can be `navigable` 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 `Grid` class represents the game world: it has a map and the main paremeters needed to maintain the game - width, height, the position of the boat, and the position of the treasure. Make sure to use a priority queue object to maintain the set of open cells in the `find_path` method.

The `TreasureHunt` class sets up the game. Each game can have the status "STARTED", "WON" or "OVER".

The game can be played from keyboard if you build a user interface for it, however for the purpose of this assignment, we will allow commands of a play session to be read from a text file. Sample test files and some test code will be provided a few days after the Assignment has been published, so you can test.

Please note we have NOT prepared a user interface (and we won't). If you prefer to play your game, you may consider to build a user interface. In case you do, please do not submit it.

Make sure to test your work carefuly before submitting! Submit the completed code in `container.py`, `grid.py` and `treasurehunt.py`.