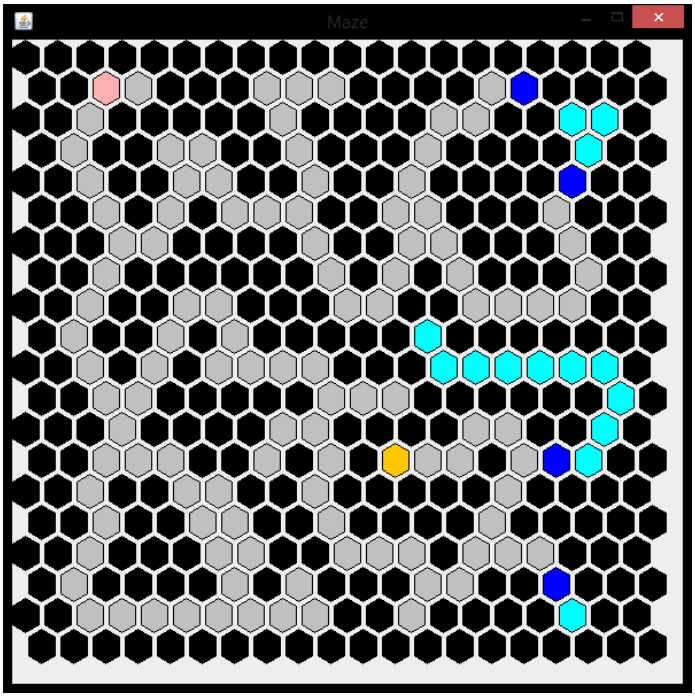
CS1027: Assignment 3

Due: Thursday, July 27th, 11:55pm.
Weight: 10%

Mazes can be solved by computers using a variety of algorithms. This time we will use some Artificial Intelligence (AI) to solve the maze more efficiently! This assignment uses many of the same concepts and files as assignment 2.

This picture shows a completed maze (maze2.txt) using this algorithm.

```
The end of the maze was found.
38 steps to the end.
There are still 4 Hexagons in
the queue.
109 steps were taken.
```



Functional Specifics

For this assignment you are given a number of classes to use, and you will also use some of the classes from Lecture. You will create classes: `MazeSolverToo`, `PriorityNode` (basically a node class with a priority attribute), and `LinkedPriorityQueue` (I recommend copying the normal queue class and then adding a special enqueue based on priority). You are provided with the interface `PriorityQueueADT` which will help you create the `LinkedPriorityQueue` class.

The classes you will mostly be interacting with are **Maze**, **Hexagon**, and the classes you make.

The API for the provided classes will orient you to their methods: API Reference. In other words, ***before you start***, familiarize yourself with the classes, specifically the altered **Hexagon** class!!! Be sure to read the description of the class near the top of the page and the available methods provided by the class.

There are a few differences from assignment 2 to note here. There is a different set of Hexagon Types, Hexagons also have a **stepsToMe** attribute, and there are a few important new methods you should investigate.

The tiles will display in different colours so you can track your `MazeSolverToo`'s progress.

You're going to actually use some AI to intelligently find the end of the maze! You will use something called A* . Basically, it's f(x) = g(x) + h(x), where x is a **Hexagon**, f(x) is a **priority** of x, g(x) is how far x is from the start (look at **stepsToMe**), and h(x) is some **heuristic** - how far we think x is from the end (you should look into **distanceToEnd()**). **WARNING**: Because I was lazy when I made the **distanceToEnd()** function, if you change the size of the window that the maze exists in, this function won't work properly. In other words, leave the window size alone.

So instead of using a stack, you will be using a queue. Actually, you will be using a **priority** queue where the priority will be f(x). If you want to know what a priority queue is, then I suggest you start Googling!

High Level Algorithm:

- Try to create a maze object reference
- Create a Hexagon reference and get the start Hexagon tile from the maze and add it to the priority queue
- while queue is not empty
    - dequeue
    - Is this the end?
    - Check node's neighbours and add them to priority queue based on f(x)
    - Note: Each time through we need to update the maze window with a call to repaint()
- Once we have searched the maze using the above algorithm, print out the following using a System.out.println or .format:
    - If the end was found, or if it was not found
    - The number of steps to get to finish
    - How many tiles were still in the priority queue
    - Number of steps taken

Classes to Submit:

- `MazeSolverToo.java`
- `PriorityNode.java`
- `LinkedPriorityQueue.java`

Exceptions

Your code must correctly handle thrown exceptions. To handle the exceptions you need only to inform the user through a console print statement what specifically has happened. These handlers must be specific (rather than one catch block for Exception itself). Be sure to be mindful of exception superclasses/subclasses.

Command Line Arguments

You must read the Maze file from the command line. The command line is what computers used to use to run programs rather than having graphical windows, and all systems still have the functionality available (like through the Terminal application on Macs, or the Command Prompt (cmd) application on Windows). The user could run the MazeSolverToo program with the following command from a command line (for example): `java MazeSolverToo maze1.txt`

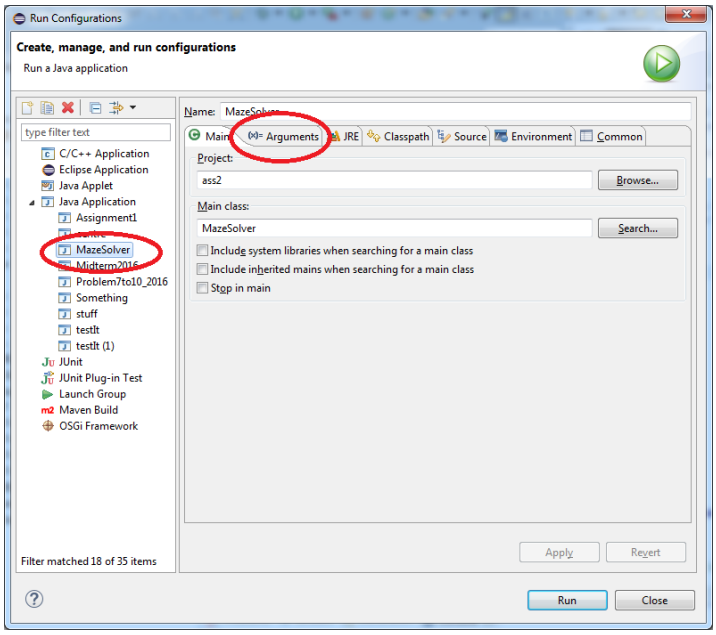
If you have ever wondered what the "`String[] args`" thing meant in the main method header, this is what it is for; It allows us to read in the text supplied on the command line. The args array of Strings will store any of the text supplied by the user, that is, any tokens following the application name (MazeSolverToo in the above example).

To read a single argument, we look in args[0], but first we want to check that the user has entered something. The following code example could be the beginning of your MazeSolverToo.java file. It will check the length of the args array and thrown an exception if there is not an argument provided. Then it will store a reference to the String using the reference variable mazeFileName.

```
public class MazeSolverToo {
      public static void main (String[] args) {
            try{
                  if(args.length<1){
                        throw new IllegalArgumentException("No Maze Provided");
                  }
                  String mazeFileName = args[0];
                  //...
```

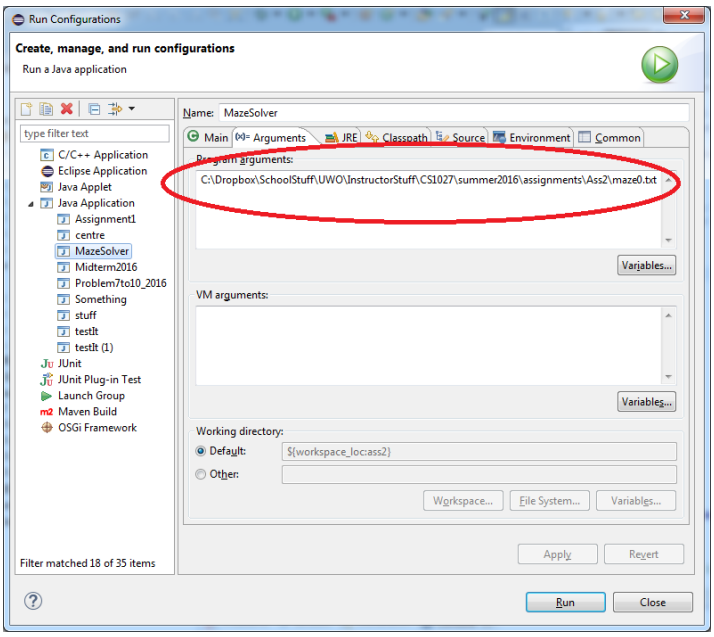
Setting up a Command Line Argument when running your program in Eclipse

To get Eclipse to supply a command line argument to your program when it runs, you will need to modify the "Run Configurations". On my computer it's located in "Run"->"Run Configurations". Something like the following should pop up.



Be sure the "Java Application->MazeSolverToo" (note, picture only says ‘MazeSolver’) is the active selection on the left-hand side.

Select the "Arguments" tab.



Enter the filename and location in the "Program arguments" text box. In my case I have it in "C:\Users\James\Dropbox\SchoolStuff\UWO\InstructorStuff\CS1027\summer2016\assignments\Ass3\maze0.txt", however you may have it somewhere completely different and that is to be expected.

If you are on Mac/Linux you will likely have something like "/Users/SomeName/Desktop/Ass3/maze2.txt".

FAQ:

- **Q:** I don't know where to start.
  - **A:** Did you read the whole assignment details?
- **Q:** I still don't know where to start.
  - **A:** Go over the high level description of the algorithm. It really does into the details of solving the problem**.** You should also make sure you're comfortable with the provided classes – read the documentation provided.
- **Q:** I don't know what A* is.
  - **A:** Read this. For real, read it! If you're still lost, check out YouTube, I'm certain there are videos there describing it.
- **Q:** Should I comment X?
  - **A:** Probably.
- **Q:** There is a bug in my code!
  - **A:** Look at the debugging lecture slides. There are a lot of tips in there.
- **Q:** Am I required to make any alterations to any of the provided code?
  - **A:** No, you shouldn't have to.
- **Q:** Did I catch *enough* exceptions?
  - **A:** No idea. Keep trying to figure out how to break your code. Make sure you have *at least* 4 being dealt with.
- **Q:** Can I change the specifications, even if it's to make it 'better'?
  - **A:** NO! Not even if you invent the best maze solver in the world.
- **Q:** When catching exceptions, does order matter?
  - **A:** Yes, it does actually. Remember subclasses are also the same type of parent classes.
- **Q:** I swear I did everything right, but for some reason my files won't open!
  - **A:** This isn't uncommon.
    - If you're using eclipse, try putting the .txt files in the project directory (the same folder containing the *bin* folder, *src* folder.
    - If you're running from command line, try putting the files in the *src* folder.
- **Q:** I don't know how to do X?
  - **A:** Try going to my favorite website: www.google.ca and then typing X into the big text box.
- **Q:** Can I email the TA or professor with questions?
  - **A:** Yes, but you should really check out my favorite website (above).

Non-functional Specifications:

1. Include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.
2. Assignments are to be done individually and must be your own work. Software may be used to detect cheating.
3. Use Java coding conventions and good programming techniques, for example:
   i. Meaningful variable names
   ii. Conventions for naming variables and constants
   iii. Use of constants where appropriate
   iv. Readability: indentation, white space, consistency
   v. private vs. public

Make sure you attach your files to your assignment; **DO NOT** put the code inline in the textbox. **DO NOT SUBMIT YOUR *.class* FILES. IF YOU DO THIS, AND DO NOT ATTACH YOUR *.java* FILES, YOU WILL RECEIVE A MARK OF ZERO!**

What You Will Be Marked On:

1. Functional specifications:
   - Does the program behave according to specifications?
   - Does it run with the main program provided?
   - Are your classes created properly?
   - Are you using appropriate data structures?
   - Is the output according to specifications?
2. Non-functional specifications: as described above
3. Assignment submission: via OWL assignment submission