Brandon Haakenson
Final Project Proposal

**How To Run:**

If system has the ability use makefiles simply type "make run" to compile and start. Otherwise as long as Java is installed the jar file can be ran. To run jar file type "java -jar bhaakens_final_maze.jar" when in the directory with the jar file. This project uses a graphical user interface and cannot be run without the use of some sort of display server.

**Project Summary:**

Title: Maze Generation Game

Description: This project will generate and draw a maze on the screen for the user to then solve.

Intended User: Anyone but specifically someone who likes games

What problem this solves: This project doesn't really solve any important problems but it does provide a solution to boredom.

Technologies needed:

-Java Swing to render the maze on the screen

**Use Case Analysis:**

-Should have main screen for displaying maze
-Should have section below main screen with a button for generating a new maze
-Should be able to generate a maze of variable size
-Should be able to detect current position of user
-User should be able to move through maze using arrow keys

-Should have a win screen to display for when user finishes maze

**Data Design:**

The maze is really just a grid of cells that have walls on all 4 sides. To make the maze the algorithm would then go through and break down walls to make pathways. The cells could be represented as an object that contains data such as it's position on the grid and the status of each wall. The cell objects could then be collected into an array to represent a grid. It might also be a good idea to use something like a stack to keep track of the cells that have already been visited so that the algorithm can go back when it creates a dead end.

**UI Design:**

The GUI for this project will consist of a main page on the screen which contains the maze. At the bottom of this same page will be button for generating a new maze. The only other page needed will be a win screen that displays a congratulations as well as a centered button for playing again.

main page example:

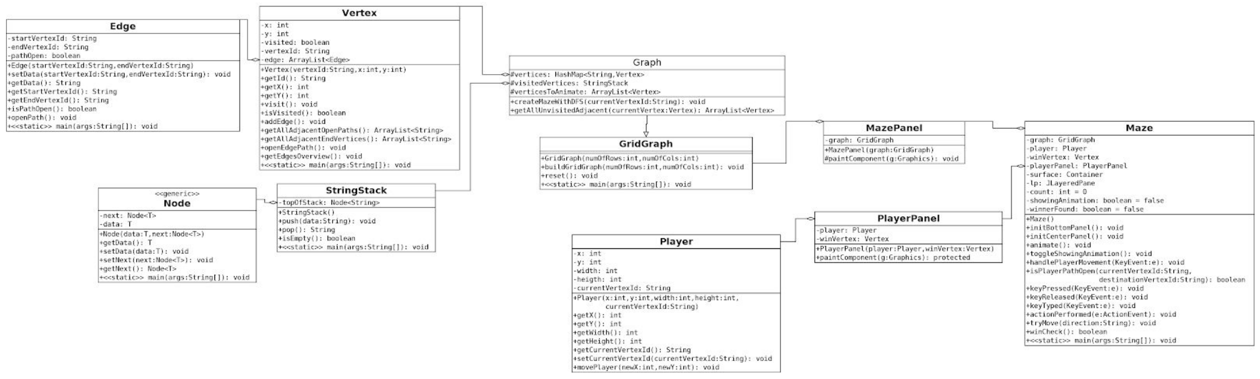
```
------------------------------------------------------
|                                                    |
|                                                    |
|                                                    |
|                       Maze                         |
|                                                    |
|                                                    |
|                                                    |
|----------------------------------------------------|
|                   | Generate|                      |
-----------------------------------------------------
```

win page example:

```
------------------------------------------------------
|                                                    |
|                                                    |
|                                                    |
|                   You Finished!!                   |
|                                                    |
|                  | Play Again? |                   |
|                                                    |
------------------------------------------------------
```

**UML:**

Can also be found as final_project_uml.png



Edge
-startVertexId: String
-endVertexId: String
-pathOpen: boolean
+Edge(startVertexId:String,endVertexId:String)
+setData(startVertexId:String,endVertexId:String): void
+getData(): String
+getStartVertexId(): String
+getEndVertexId(): String
+isPathOpen(): boolean
+openPath(): void
+<<static>> main(args:String[]): void
Vertex
-x: int
-y: int
-visited: boolean
-vertexId: String
-edge: ArrayList<Edge>
+Vertex(vertexId:String,x:int,y:int)
+getId(): String
+getX(): int
+getY(): int
+visit(): void
+isVisited(): boolean
+addEdge(): void
+getAllAdjacentOpenPaths(): ArrayList<String>
+getAllAdjacentEndVertices(): ArrayList<String>
+openEdgePath(): void
+getEdgesOverview(): void
+<<static>> main(args:String[]): void
Graph
#vertices: HashMap<String,Vertex>
#visitedVertices: StringStack
#verticesToAnimate: ArrayList<Vertex>
+createMazeWithDFS(currentVertexId:String): void
+getAllUnvisitedAdjacent(currentVertex:Vertex): ArrayList<Vertex>
GridGraph
+GridGraph(numOfRows:int,numOfCols:int)
+buildGridGraph(numOfRows:int,numOfCols:int): void
+reset(): void
+<<static>> main(args:String[]): void
MazePanel
-graph: GridGraph
+MazePanel(graph:GridGraph)
#paintComponent(g:Graphics): void
Maze
-graph: GridGraph
-player: Player
-winVertex: Vertex
-playerPanel: PlayerPanel
-surface: Container
-lp: JLayeredPane
-count: int = 0
-showingAnimation: boolean = false
-winnerFound: boolean = false
+Maze()
+initBottomPanel(): void
+initCenterPanel(): void
+animate(): void
+toggleShowingAnimation(): void
+handlePlayerMovement(KeyEvent:e): void
+isPlayerPathOpen(currentVertexId:String, destinationVertexId:String): boolean
+keyPressed(KeyEvent:e): void
+keyReleased(KeyEvent:e): void
+keyTyped(KeyEvent:e): void
+actionPerformed(e:ActionEvent): void
+tryMove(direction:String): void
+winCheck(): boolean
+<<static>> main(args:String[]): void
<<generic>>
Node
-next: Node<T>
-data: T
+Node(data:T,next:Node<T>)
+getData(): T
+setData(data:T): void
+setNext(next:Node<T>): void
+getNext(): Node<T>
+<<static>> main(args:String[]): void
StringStack
-topOfStack: Node<String>
+StringStack()
+push(data:String): void
+pop(): String
+isEmpty(): boolean
+<<static>> main(args:String[]): void
PlayerPanel
-player: Player
-winVertex: Vertex
+PlayerPanel(player:Player,winVertex:Vertex)
+paintComponent(g:Graphics): protected
Player
-x: int
-y: int
-width: int
-heigth: int
-currentVertexId: String
+Player(x:int,y:int,width:int,height:int, currentVertexId:String)
+getX(): int
+getY(): int
+getWidth(): int
+getHeight(): int
+getCurrentVertexId(): String
+setCurrentVertexId(currentVertexId:String): void
+movePlayer(newX:int,newY:int): void

**Algorithm:**

Class Edge
-------------

Method Name: Edge (constructor)
Goals: To be called when Edge object is created
Input: string startVertexId, string endVertexId
Output: N/A
Steps:
- -Set this object's startVertexId property to startVertexId parameter passed in
- -Set this object's endVertexId property to endVertexId parameter passed in

Method Name: setData
Goals: To set startVertexId and endVertexId to new values
Input: string startVertexId, string endVertexId
Output: N/A
Steps:

```
    -Set this object's startVertexId property to startVertexId parameter passed in
    -Set this object's endVertexId property to endVertexId parameter passed in
```

Method Name: getData
Goals: To return a formatted string to display startVertexId and endVertexId
Input: N/A
Output: string data
Steps:

```
    -Create string variable called data
    -concat data with startVertexId and text describing it
    -concat data with endVertexId and text describing it
    -concat data with pathOpen and text describing it
    -return string variable data
```

Method Name: getStartVertexId
Goals: To return the value of getStartVertexId
Input: N/A
Output: string startVertexId
Steps:

```
    -Simply return the value of this object's startVertexId
```

Method Name: getEndVertexId
Goals: To return the value of getEndVertexId
Input: N/A
Output: string endVertexId
Steps:

```
    -Simply return the value of this object's endVertexId
```

Method Name: isPathOpen
Goals: To return a boolean for whether or not the path is open
Input: N/A
Output: boolean true/false
Steps:
- Simply return the value of pathOpen

Class Vertex
---------------

Method Name: Vertex (constructor)
Goals: To be called when a Vertex object is created
Input: string vertexId, int x, int y
Output: N/A
Steps:
- Set object's x value to x value passed in
- Set object's y value to y value passed in
- Set object's vertexId value to vertexId value passed in
- Set visited variable to false
- Set edges variable to new ArrayList of Edges

Method Name: getId
Goals: To return the value of vertexId
Input: N/A
Output: string vertexId
Steps:
- Simply return the value of vertexId

Method Name: getX
Goals: To return the value of x
Input: N/A

Output: int x
Steps:
    -Simply return the value of x

Method Name: getY
Goals: To return the value of y
Input: N/A
Output: int y
Steps:
    -Simply return the value of y

Method Name: visit
Goals: To change the value of visited to true
Input: N/A
Output: N/A
Steps:
    -check to see if visited is false
    -if it is, then change it to true

Method Name: isVisited
Goals: To return boolean value of visited
Input: N/A
Output: boolean true/false
Steps:
    -Simply return the value of visited

Method Name: addEdge
Goals: To add an Edge object to the list of edges
Input: string startVertexId, string endVertexId
Output: doesn't return anything but will add to arraylist
Steps:
    -Create a new Edge object passing in the parameters of startVertexId and endVertexId

    -add this new edge object to edges arraylist

Method Name: getAllAdjacentOpenPaths
Goals: To return an ArrayList of strings that are the id's of all the adjacent vertices with open paths
Input: N/A
Output: ArrayList<String> endVertices
Steps:
    -Create an arraylist of strings to hold endVertices
    -loop through each vertex in the edges arraylist
    -for each edge in the arraylist check to see if it's path is open
    -if it's path is open, then add the endVertexId of that vertex to the endVertices arraylist
    -once the loop is done, return the arraylist endVertices

Method Name: getAllAdjacentEndVertices
Goals: To return an ArrayList of string the contain the id's of all the adjacent vertices
Input: N/A
Output: ArrayList<String> endVertices
Steps:
    -Create an arraylist of string to hold endVertices
    -loop through each of the vertex's edges
    -add each edge to the list regardless of it's path status
    -after looping return the arraylist

Method Name: openEdgePath
Goals: To open the path between two vertices
Input: string startVertexId, string endVertexId
Output: N/A
Steps:
    -loop through each of this vertex's edges

-for each edge check to see if it has the same startVertexId as the parameter passed in
-also check to see if the edge has the same endVertexId as the one passed in
-if it does, call openPath on it to open the path

Method Name: getEdgesOverview
Goals: To display the data for each edge for testing purposes
Input: N/A
Output: no return but will print to screen
Steps:
-for each edge in the edges arraylist call getData on it and print to screen

Class Node
-------------

Method Name: Node (constructor)
Goals: To be called when Node object is created
Input: parameter data that can be a generic data type, parameter next which is a Node of same generic type
Output: N/A
Steps:
-Set data to the data parameter passed in
-Set next to the next parameter passed in

Method Name: getData
Goals: To return the value of Node's data
Input: N/A
Output: data of whatever the generic type is
Steps:
-Simply return the value of data

Method Name: setData

Goals: To change the value of data to a new value
Input: data of generic type
Output: N/A
Steps:
  -Simply set the value of data equal to the new data parameter

Method Name: setNext
Goals: To change the value of next to new Node
Input: parameter next of generic Node type
Output: N/A
Steps:
  -Simply set the value of next equal to the parameter next

Method Name: getNext
Goals: To return the value of next
Input: N/A
Output: node next of generic type
Steps:
  -Simply return the value of next

Class StringStack
-------------------

Method Name: StringStack (constructor)
Goals: To be called when StringStack object is created
Input: N/A
Output: N/A
Steps:
  -Set the value of topOfStack equal to null

Method Name: push
Goals: To push a value to the top of the stack

Input: string data
Output: N/A
Steps:
    -Create a new string Node and pass it data parameter as well as topOfStack parameter
    -set the top of the stack equal to this new node

Method Name: pop
Goals: To remove and return the node on top of the stack
Input: N/A
Output: string value
Steps:
    -Create string to hold value of node
    -check to see if the stack is empty
    -if it is set value equal to a string describing this
    -otherwise if stack isn't empty get the value of the Node on the top of the stack and store it in value variable
    -Set the top of the stack equal to the next node on the stack
    -return the value variable

Method Name: isEmpty
Goals: To return boolean saying whether or not the stack is empty
Input: N/A
Output: boolean true/false
Steps:
    -Check to see if the top of the stack equals null
    -if it is null, then return true
    -otherwise if the top of the stack isn't null
    -then return false

Class Graph
--------------------

Method Name: createMazeWithDFS
Goals: To step through each vertex in the maze using a recurisve depth first search and carve out a maze by opening paths.

Method will end when all adjacent vertices are visited and it can't find anything unvisited by backtracking

Input: string currentVertexId
Output: N/A
Steps:
-Create and store the currentVertex using the currentVertexId provided as parameter
-Mark the current vertex as visited
-Check to see if the current vertex is already in the verticesToAnimate arraylist
-if it isn't, add it to the arraylist
-get all the unvisited adjacent vertices and store in variable
-Create vertex variable for currentEndVertex and set to null
-check to see if there are actually unvisited vertices
-if there is then randomly choose one and set currentEndVertex equal to it
-check to see if a currentEndVertex has been found (no longer null)
-if it has then push the currentVertexId onto the stack of visited vertices
-then open a path from currentVertex to currentEndVertex
-also open path in opposite direction of currentEndVertex to currentVertex
-recursively call createMazeWithDFS again and pass in the id of the currentEndVertex so that vertex will be visited next
-otherwise if the currentEndVertex is null still
-then check to see if there is anything on the stack
-if there is something on the stack then pop it off and store id
-recursively call createMazeWithDFS and pass in this popped off id so that the algo is sent back to most recent vertex to try again

Method Name: getAllUnvisitedAdjacent
Goals: To return an ArrayList of all the adjacent vertices that are unvisited
Input: Vertex currentVertex

Output: ArrayList<Vertex> allUnvisitedAdj
Steps:
-Get an arraylist with id's of all the adjacent vertices
-create empty arraylist for allUnvisitedAdj
-loop through each of the adjacent vertices
-check to see if the current adjacent vertex has been visited
-if it hasn't then add it to the ArrayList of unvisited adj
-after looping return allUnvisitedAdj

Class GridGraph
--------------------

Method Name: GridGraph (constructor)
Goals: To be called when GridGraph object is created
Input: int number of rows, int number of columns
Output: N/A
Steps:
-call buildGridGraph method and pass in parameters for number of rows/columns

Method Name: buildGridGraph
Goals: To create a rectangular grid graph of Vertices/Edges based on number of rows/columns passed in
Input: int numOfRows, int numOfCols
Output: N/A
Steps:
-Create int variable for total number of vertices by multiplying numOfRows and numOfCols
-Create int variable to keep track of currentRow
-Create int variable for x positioning
-Create int variable for y positioning

-for loop that starts at 0 and ends at the number of totalVertices, so that it loops once for each vertex

-Check to see if the currentRow should be incremented. This is done by checking to see if the
current iteration is evenly divisible by the number of columns and isn't zero.

-If the currentRow row should be incremented, then increment it and increase y positioning while reseting x positioning to zero

-create a new vertexId string by concating with current iteration number

-create a new vertex using this id and the positioning

-Create int variable for unique indices of grid graph
such as topLeftCorner, topRightCorner, bottomLeftCorner, bottomRightCorner, leftColumn, rightColumn.
topLeftCorner is always at zero index, topRightCorner is always numOfCols-1, bottomLeftCorner is always totalVertices-numOfCols,
bottomRightCorner is always totalVertices-1, leftColumn is always currentRow times numOfCols, rightColumn is always (currentRow times numOfCols) + (numOfCols - 1)

-check if index is topLeftCorner

-if it is then create edge to the right and below

-otherwise check if index is topRightCorner

-if it is then create edge to the left and below

-otherwise check if index is bottomLeftCorner

-if it is then create edge above and to the right

-otherwise check if index is bottomRightCorner

-if it is then create edge above and to the left

-otherwise check if index is in first row between corners

-if it is then creat edge to left, right and below

-otherwise check if index is leftColumn

-if it is then create edge above, to right and below

-otherwise check if index is rightColumn

-if it is then create edge above, to left and below

-otherwise check if index is in bottom row between corners

-if it is then create edge above, and to left/right

-otherwise the index must be in the middle of the graph
-this means that edges and be created for all directions above/below/left/right
-Add newly created vertex to hashmap of vertices
-increase x positioning
-end the for loop

Method Name: reset
Goals: To reset and rebuild that graph so that there is a new maze
Input: N/A
Output: N/A
Steps:
-Set the value of vertices to a new hashmap
-Set the value of visitedVertices to a new StringStack
-Set the value of verticesToAnimate to a new ArrayList
-call buildGridGraph
-call createMazeWithDFS

Class Player
--------------------

Method Name: Player (constructor)
Goals: To be called when Player object is created
Input: int x, int y, int width, int height, string currentVertexId
Output: N/A
Steps:
-Set value of x equal to parameter x
-Set value of y equal to parameter y
-Set value of width equal to parameter width
-Set value of height equal to parameter height
-Set value of currentVertexId equal to parameter currentVertexId

Method Name: getX

Goals: To return the x position of Player
Input: N/A
Output: int x
Steps:
    -Simply return the value of Player's x property

Method Name: getY
Goals: To return the Y position of Player
Input: N/A
Output: int Y
Steps:
    -Simply return the value of Player's y property

Method Name: getWidth
Goals: To return the width of Player
Input: N/A
Output: int width
Steps:
    -Simply return the value of Player's width property

Method Name: getHeight
Goals: To return the height of Player
Input: N/A
Output: int heigth
Steps:
    -Simply return the value of Player's height property

Method Name: getCurrentVertexId
Goals: To return the currentVertexId of Player
Input: N/A
Output: string currentVertexId
Steps:
    -Simply return the value of Player's currentVertexId property

Method Name: setCurrentVertexId
Goals: To change the value of currentVertexId to a new string id
Input: string currentVertexId
Output: N/A
Steps:
    -Set the value of Player's currentVertexId property equal to parameter currentVertexId

Method Name: movePlayer
Goals: To move the player to a new position on the screen
Input: int newX, int newY, string newVertexId
Output: N/A
Steps:
    -Set the value of Player's x property equal to newX
    -Set the value of Player's y property equal to newY
    -Set the value of Player's currentVertexId property equal to newVertexId

Class PlayerPanel
--------------------

Method Name: PlayerPanel (constructor)
Goals: To be called when PlayerPanel object is created
Input: Player player, Vertex winVertex
Output: N/A
Steps:
    -Set the value of PlayerPanel's player property equal to the player parameter passed in
    -Set the value of PlayerPanel's winVertex property equal to the winVertex parameter

Method Name: paintComponent

Goals: Swing calls this method for animation. This method will create two colored squares on the screen.

One square is for the player and the other represents the end of the maze

Input: parameter g of type Graphics that Swing will pass in automatically

Output: N/A

Steps:

-call super on paintComponent and pass in g (as per Swing documentation)

-set color of graphics to green for painting win vertex

-call fillRect on g and pass in the x,y,width,height properties of winVertex

-change color to red for player square

-call fillRect and pass in position info from player

Class MazePanel

--------------------

Method Name: MazePanel (constructor)

Goals: To be called when MazePanel object is created

Input: GridGraph graph

Output: N/A

Steps:

-Set the background color of this panel to black

-Set the value of graph equal to the graph parameter passed in

Method Name: paintComponent

Goals: To be called by Swing for animation. Will either show animation of Maze being generated or will just show maze as is

Input: g parameter of type Graphics provided by Swing

Output: N/A

Steps:

-call super on paintComponent and pass in g (as per Swing docs)

-typecast g to Graphics2D so that we also have this type of object for drawing lines with Swing

```
    -set color of graphics to white and set line stroke size
    -set width and height to the same as normal vertex
    -check if the maze is supposed to be showing an animation and there are verticesToAnimate
    -if there are, then start looping through the verticesToAnimate up to count variable
    -for each iteration use graphics object to draw the current vertex
    -also get all of the currentVertex's adjacent vertices and start looping through them
    -for each adjacent vertice draw a line between the current vertex and the adjacent vertex
    -otherwise if the animation isn't being shown and there are vertices
    -loop through every vertex stored in the graph's hashmap
    -for each vertex use graphics object to draw the vertex
    -also get all the adjacent verices for the currentVertex and draw a line between the two
```

Class Maze
--------------------

Method Name: Maze (constructor)
Goals: To be called when Maze object is created. Takes care of setting up JFrame and JPanels for animation
Input: N/A
Output: N/A
Steps:

```
    -call super with a string for the title of the JFrame
    -set default close operation and size for JFrame
    -set JFrame as visible and in focus
    -set layout as BorderLayout
    -add key listener for arrow key movement
```

-call initCenterPanel to create center panel that holds MazePanel and PlayerPanel
-call initBottomPanel to create panel for button on the bottom
-make sure that the MazePanel is set to visible
-call animate to start animation of maze being built right when Maze Object is created

Method Name: initBottomPanel
Goals: To create the panel that goes on the bottom of the screen containing a button for generating a new maze
Input: N/A
Output: N/A
Steps:
-create new Jpanel for bottomPanel
-set bottomPanel layout to grid layout
-create JButton called generate
-add action listener to the button so button presses can be handled
-add button to bottomPanel
-add bottomPanel to the bottom part of the JFrame surface

Method Name: initCenterPanel
Goals: To create the center panel that actually shows the maze
Input: N/A
Output: N/A
Steps:
-set the value of graph equal to a new GridGraph of desired size
-call createMazeWithDFS on the graph to carve out a maze
-set the value of surface to the JFrame's content pane
-set the value of layeredPanel equal to a new JLayeredPanel
-set the value of mazePanel to a new MazePanel that takes the graph as a parameter
-set bounds for the panel so that it will be the right size and be rendered by Swing

-add mazePanel to the layeredPanel at level 1 so that things can be overlayed on top of it
-set player value to a new Player Object that starts at the first vertex
-set winVertex equal to the last vertex from the graph
-set playerPanel equal to a new PlayerPanel that takes player and winVertex as parameters
-set bounds on playerPanel so that it will be rendered
-set the background on player panel to transparent because it will be overlayed on top of MazePanel and we don't need to see it
-add the playerPanel to the layeredPanel on top of mazePanel so that player can be seen on maze
-add the layeredPanel to the surface of the JFrame in the center
-make sure that mazePanel and playerPanel are invisible for now until proper animation timing

Method Name: animate
Goals: To show the passages of the maze being built as an animation rather than all just appearing at once
Input: N/A
Output: N/A
Steps:
-get the number of verticesToAnimate and store as an int
-check to make sure that the maze isn't already showing an animation
-if it isn't then toggleShowing animation on
-also make sure that count is reset to 0
-then create a new Timer object that will do something repeatedly after the desired amount of time has passed
-each tick of the timer increment the count and check to see if count has reached the end of the verticesToAnimate
-if it has reached the end then stop the timer, toggle the animation off, and set playerPanel to visible so that player now appears
-also explictly tell Swing to repaint the screen every tick of the timer
-finally tell the timer to actually start

Method Name: toggleShowingAnimation
Goals: To toggle the boolean value of showingAnimation
Input: N/A
Output: N/A
Steps:

```
-check if showingAnimation equals true
-if it is true, then set it to false
-check if showingAnimation is false
-if it is false, then set it to true
```

Method Name: actionPerformed
Goals: Called when a button is pressed, used to add functionality to "Generate" button
Input: ActionEvent e
Output: N/A
Steps:

```
-check if the action being passed in is for the "Generate" button
-if it is, check to make sure that maze isn't already in process of animating
-if it isn't already animating then make sure the playerPanel is hidden
-also call reset on the graph so it creates a new on
-also move the player back to the starting position
-make sure the win flag is reset
-make sure that mazePanel can be seen again
-call animate to start showing the new maze being generated
```

Method Name: keyPressed
Goals: Called when a key is pressed, used to move player position
Input: KeyEvent e
Output: N/A
Steps:

```
-check to make sure the player hasn't already won
-if they haven't then call handlePlayerMovement and pass on the event
```

-check to see if there is now a winner
-if there is, show win dialog

Method Name: handlePlayerMovement
Goals: Determines which key was pressed and calls method for moving in corresponding direction
Input: KeyEvent e
Output: N/A
Steps:
-get the key code from the event and store as int
-check the value of the keycode
-if it was the up arrow key call tryMove with "UP" parameter
-if it was the right arrow key, call tryMove with "RIGHT" parameter
-if it was the down arrow key, call tryMove with "DOWN" parameter
-if it was the left arrow key, call tryMove with "LEFT" parameter

Method Name: tryMove
Goals: To move the player in the specified direction if there is an open path to an adjacent vertex in that direction
Input: String direction
Output: N/A
Steps:
-get the player's current vertex and all the open paths from this vertex
-get the player's current positioning
-loop through each available path
-get the positioning of the current possible adjacent vertex
-if the direction is "UP" check if the possible y position of the vertex is less than the player's y position
-if it is, this means the possible vertex is above the player and the player's position can be moved to it
-if the direction is "RIGHT" check to see if the possible x position is greater than the player's x position
-if it is, then move the player to this new location

-if the direction is "DOWN" check to see if the possible y position is greater than the player's y position
-if it is, then move the player to this new location
-if the direction is "LEFT" check to see if the possible x is less than the player's x position
-if it is, then move the player to this new location
-if there are no adjacent vertices that meet the direction requirement then do nothing

Method Name: winCheck
Goals: To check and see if the player has made it to the end of the maze
Input: N/A
Output: boolean true/false based on whether or not the player has found the end of the maze
Steps:
-check if the player's x position equals the x position of the winVertex
-check if the player's y position equals the y position of the winVertex
-if these two things are true then set winnerFound to true and return true
-otherwise just return false

Method Name: isPlayerPathOpen
Goals: To check if a vertex has an open path to another specified vertex
Input: String currentVertexId, String destinationVertexId
Output: boolean true/false
Steps:
-get the currentVertex using currentVertexId
-get all adjacentOpenPaths for the current vertex
-check if the arraylist of open paths has the destinationVertexId in it
-if it does, then return true
-otherwise return false