Final Project Documentation

Examples are based on the sample maze.txt file included in the repo.

Public Methods:

class maze

{

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

bool maze::createMaze(vector<string> rooms, vector<vector<int>> connections)

This function takes in a vector or strings (the names of the rooms in the maze), and a vector that holds vectors of INTs. The double vector acts just like a 2D array, it stores all the connections between rooms in the maze. It returns true if there is an error in the maze.txt file and false if the graph was created successfully.

Example:

maze m; //creates maze class before sending to createMaze.

//Boolean(holds true/false), rooms(vector of string room names), connections(vector of vector of risk values).

Boolean = m.createMaze(rooms, connections)

Precondition:

A graph must be initialized before calling the function. In this case m is the graph. maze.txt should be read in and the rooms vector and connections vector should contain the names and connections to each other. Each risk value in the maze.txt file that connects the rooms must be valid. Rooms with one way connections (-1 one way and a positive number the other way) or rooms that are not accessible from the rest (aka in different districts) will fail the error checking at the end of the function.

Postcondition:

The private variable vertices will contain every room listed in the maze.txt file and each of those rooms will contain a vector of its adjacent rooms (edges). -1 and -2 values in the maze.txt file will be ignored when creating connections between rooms. In addition each vertices will have an ID assigned which is done with an incrementing loop starting from 1. If the rooms and connections are valid, the function will return false. This function calls 3 other functions in order to complete graph creation and error checking.

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

void addVertex(std::string name)

This function takes a string from the rooms vector and creates a vertex in the private maze variable vertices. This Is the first helper function called from the createMaze method.

Example:

addVertex(rooms[0]) //grabs entrance and creates a vertex.

Precondition:

A vector of the room names have been passed into createMaze. Also the room does not already exist in the private variable vertices.

Postcondition:

Private variable vertices now contains a vertex of entrance. If the vertex already exists in vertices the function will skip adding it again.

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

void addEdge(std::string room1, std::string room2, int risk, int option)

This function is the second helper function called from the createMaze method. This function takes in two room names, the risk value (connection) between them, and an integer of either 1 or any other number.

Example:

addEdge(entrance,woods,30,0)

//Adds a connection between entrance and woods with a risk of 30%. Adds connection one way only.

Precondition:

The private variable vertices must be complete with all the rooms (vertexes) before this is called. The connections vector that is passed in to createMaze is also used. If calling from createMaze, an integer that is not equal to 1 must be passed int to addEdge as well. If calling from addRoom, an integer that is equal to 1 must be passed to addEdge.

Postcondition:

entrance is now connected to woods and has a 30% risk traveling between them. If int option is 1, the function will create an edge both ways. This is not needed when calling from createMaze as a for loop will add the connections both ways manually. However this is needed when calling from addRoom as there is no loop to add an edge both ways.

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

void maze::mazeErrorCheck()

This function assigns ID's the same way as assigning districts in a graph. This is the last helper function called in createMaze. After it is called, a loop through the vertices can check this value.

Example:

mazeErrorCheck();

Precondition:

The private variable vertices has been fully constructed. All vertexes in vertices contain a mazeCheck value of -1

Postcondition:

All vertexes in the vertices vector contain mazeCheck values not equal to -1 if the graph is all interconneted.

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

void maze::mazeErrorCheckBFT(string startingRoom, int errorCheck)

This function is called by mazeErrorCheck to perform a breadth first search through vertices.

Example:

mazeErrorCheckBFT(ballroom,errorCheck);

// entrance is the current vertex in the for loop going through vertices. errorCheck is the mazeCheck integer of the vertex (all mazeChecks are initialized to -1).

Precondition:

The vertex has a mazeCheck value of -1.

Postcondition:

The vertex will have a positive mazeCheck value if it has some connection to the rest of the graph.

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

void maze::findSafestPath()

This function will utilize dijkstra's algorithm and find the safest possible path from the entrance and exit vertexes. It will print the path in order and will also divide the total sum of risk values by the number of rooms on that path to get an average risk.

Example: (using the sample maze.txt)

entrance->balcony->observatory->basement->passage->exit

average risk = 33%

Precondition:

Graph has been created. Error checking has passed successfully.

Postcondition:

A vector containing a path that has the lowest sum of risks in the entire graph will be displayed in order.

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

void maze::findShortestPath()

This function is very similar to findSafestPath but will find the shortest path from the entrance and exit vertexes and will disregard the risk.

Example: (using the sample maze.txt)

entrance->ballroom->basement->passage->exit

Precondition:

Same as findSafestPath.

Postcondition:

Same as findSafestPath but instead of the path with the lowest total risk, the path with the shortest amount of hops from entrance to exit will be displayed.

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

void maze::addDoorAndKey()

This function is called from main menu. This will search the vertices vector, find the rooms, and place a locked door in one and place a key in another.

Example:

(first user input) = kitchen

(second user input) = basement

//This will add a door to kitchen and a key In the basement.

Precondition:

kitchen's bool door is false. basements bool key is false.

Postcondition:

kitchen's bool door is true. basements bool key is true.

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

void maze::addRoom()

This function is called from the main menu. It will prompt the user to input the name of a room to add to the vertices vector. It will also prompt the user to input the name of rooms they want the new room to be connected to as well as the risk values.

Example:

(user input) = newroom

(second user input) = woods

(third user input) = 10

(forth user input) = y

(fifth user input) = kitchen

(sixth user input) = 0

(secenth user input) = n

//This will create a new room called newroom, and connect it to woods with a risk of 10%, it will also connect it to kitchen with a risk of 0%.

Precondition:

newroom does not exist in vertices.

Postcondition:

vertices vector now contains newroom along with its connections to woods and kitchen.

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

void maze::printRooms()

This function will simply loop through the vertices vector and print the names of the rooms, their connections to other rooms and the risk values going to those rooms.

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

}

class player

{

void game(std::vector<vertex> maze);

This function will take in a copy of the vertices vector and run the maze game. This function uses two helper functions.

Precondition:

The player will start in the entrance room and will be given options on where they can go next. currentRoom will be set to entrance and the winGame variable will be set to the ID of the exit room.

Postconditions:

This function will return and end the program after the user reaches 0 lives, or the currentRoom ID matches the ID of exit.

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

int player::gameShowChoices(int choices)

This function takes in an integer (choices) that resides in the game function (it will always be 0 when passed to this function) and returns that same variable. It will also determine the current path options based on what the current room the user is at.

Example: (from maze.txt)

currentRoom = entrance

Precondition:

choices = 0, currentRoom = entrance, currentOptions and currentRisks (vectors) will be empty.

Postcondition:

choices = 3,

currentRoom = entrance,

currentOptions = {hallway,balcony,ballroom}

currentRisks = {40,30,20}.

The function will also check and see if any of the rooms the player can travel to have locked doors that require.

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

void player::setDifficulty()

This function will prompt the user to select 1 of 3 difficulty options for the game. Then it will alter the private variables lives, and keys accordingly.

Example:

* Easy

Precondition:

lives = 3, keys = 0 //This is the default if this function is not called before starting the game.

Postcondition:

lives = 5, keys = 1

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

void player::gameHelp()

This function displays how the game works as well as what lives,keys,risks etc mean.

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

}