Hunt The Wumpus--Adithya Shastry

Summary:

The goal of this project was to create a hunt the Wumpus game. The game uses the graph data structure to create rooms that the hunter and the Wumpus occupy. The solution for this project was developed using a number of classes to handle the hunter moving, end game conditions, and the display. These classes were specifically the hunter class, HuntTheWumpus and Wumpus, landscape and LandscapeDisplay. The combination of all of these classes produces a game where the player (the hunter) is tasked with slaying the Wumpus.

Vertex

The Vertex class was mainly used to make the rooms in the game which blocked the hunter from moving to inaccessible rooms. The neighbors were implemented using my own hash table(extension) where the keys were enumerations of the cardinal directions and the values were the vertexes that they connected to.

```
// Run Dijkstra's algorithm to determine the length of the shortest path
// to each VertexSolution.
public void computePathLengths(int startingFrom) {

    // Mark the vertices as unmarked and set the cost to infinite

    for (Vertex v : this.vertices) {
        if(v!=null){
        v.setCount(Integer.MAX_VALUE);
        v.setMarked(false);
        }
    }

    // Now I will set the cost of the starting vertex to 0

    Vertex v = this.vertices.get(startingFrom);
    v.setCount(0);

    // create a priority queue
    PriorityQueue<Vertex> pq = new PriorityQueue<Vertex>(new VertexComparator());
    // Add the value of v to the priority queue

    pq.offer(v);

    while (pq.peek() != null) {
        // remove the head of the priority queue
        Vertex v0 = pq.poll();
        v0.setMarked(true);
        for (Vertex w : v0.getNeighbors()) {
            if (!w.isMarked() && w.getCount() > v0.getCount() + 1) {
                w.setCount(v0.getCount()+1);
                pq.offer(w);

            }

        }

    }
    // Uncomment this line to get the point statements that match the
```

Graph

The graph class was used to make the connections between the vertexes based on their position in the landscape. The graph is also responsible for handling the shortest path algorithm, Dijkstra's algorithm, and assigning the correct cost values for the vertices in the graph based on how far they are from the chosen vertex. This method can be seen below:

Landscape

The Landscape class dealt with the actual location of the vertices in the game. I used a grid structure in order to do actually populate the landscape and used the graph class to handle the connections and other things like that. Random layouts are generated by allowing for a density condition when the vertices are actually created, making the layout change every time the user plays the game. (Extension)

Hunter

The Hunter class handled the key pressed and released events as well as the move function of the game. All of these functions are controlled by the player when he/she presses a key. In order to actually assign the hunter a location on the graph, the hunter will hold the vertex object of the room that it is in. This is then used to draw the hunter and handle all the movements and shooting events. The code for the key events can be seen below:

```
@Override
public void keyPressed(KeyEvent arg0) {
    if (arg0.getKeyCode() == (KeyEvent.VK_W)) {
        d = DIRECTION.NORTH;
    } else if (arg0.getKeyCode() == (KeyEvent.VK_S)) {
        d = DIRECTION.SOUTH;
    } else if (arg0.getKeyCode() == (KeyEvent.VK_A)) {
        d = DIRECTION.WEST;
    } else if (arg0.getKeyCode() == (KeyEvent.VK_D)) {
        d = DIRECTION.EAST;
    } else if (arg0.getKeyCode() == (KeyEvent.VK_SPACE)) {
        if (this.shoot) {
            this.shoot = false;
        } else {

            this.shoot = true;
        }
    }
    this.move();
    d = null;

}

@Override
public void keyReleased(KeyEvent arg0) {
    if (arg0.getKeyCode() == (KeyEvent.VK_W) && this.d == DIRECTION.NORTH)
        d = null;
    else if (arg0.getKeyCode() == (KeyEvent.VK_S) && this.d == DIRECTION.SOUTH)
        d = null;
    else if (arg0.getKeyCode() == (KeyEvent.VK_A) && this.d == DIRECTION.EAST)
        d = null;
    else if (arg0.getKeyCode() == (KeyEvent.VK_D) && this.d == DIRECTION.WEST)
        d = null;
    else if (arg0.getKeyCode() == (KeyEvent.VK_SPACE))
        ;
}
```

Wumpus

The Wumpus class is mainly used to handle the conditions and draw methods of the Wumpus in the game. These are handled using various fields like the alive Boolean that holds if the Wumpus is actually alive or not. The Wumpus object is also an attribute that the hunter class has because this will allow the move methods and the shooting events the ability to actually work.

HuntTheWumpus

The Hunt the Wumpus class sets up the game and handles the end game situations. The game is set up by generating a landscape and graph, making the connections in the graph, and finally assigning the Wumpus and the hunter locations on the graph. I ran into a lot of errors when doing this therefore I needed to create a while loop that will only break when the indexes, both for the hunter and the Wumpus, pass all of the criterion. This can be seen below:

```
}
while(hunterIdx==wumpusIdx || land.getGraph().getVertex(hunterIdx)==null ||
        land.getGraph().getVertex(wumpusIdx)==null ||
        land.getGraph().getVertex(hunterIdx).getNeighbors().size()<=0 ||
        land.getGraph().getVertex(wumpusIdx).getNeighbors().size()<=0);
 this.hunter.setVertex(land.getGraph().getVertex(hunterIdx));
 Wumpus wumpus=new Wumpus(land.getGraph().getVertex(wumpusIdx));
 this.hunter.setWumpus(wumpus);
```

Extensions

Hash table

I used a hash table to handle the neighbors in the vertex class. This is explained in the Vertex section above

Random Game Generation

I generated random landscapes for the game to played in. This is explained in the HuntTheWumpus class section above.

Making the game a .jar file

After some research, I figured out how to make the game into a standalone executable! I think this is a very cool extension because it means I can share my game with all of my friends and family, which I have! The file can be found in the project folder!

Acknowledgements

I would like to thank Dr. Codabux, Dr. Taylor, and Ethan Pullen for helping me on this project.