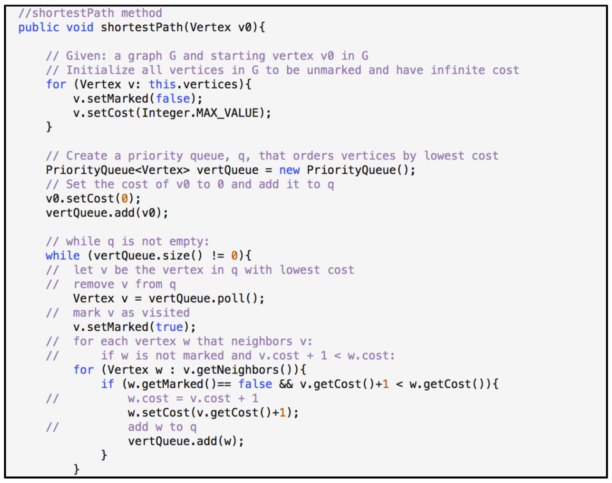
Project 9

PROJECT 9

For this project, we were ask to implement a new data structure called Graphs. We had to create a game called Hunt The Wumpus which is essentially similar to a Minesweeper type game. The only basic instructions of the project were to create 5 main classes: Vertex, Graph, Hunter, Wumpus, and HuntTheWumpus. We had to use our previous Cell, Landscape, LandscapeDisplay and LinkedList classes as well. The basic principles of the game are the Hunter roams the landscape looking for the invisible Wumpus. If you can shoot the Wumpus with an arrow you win, but if you enter the Wumpus lair, you will DIE. For my project, I decided to them the Hunter and Wumpus as Neo and AgentSmith from The Matrix!

==Task 1==

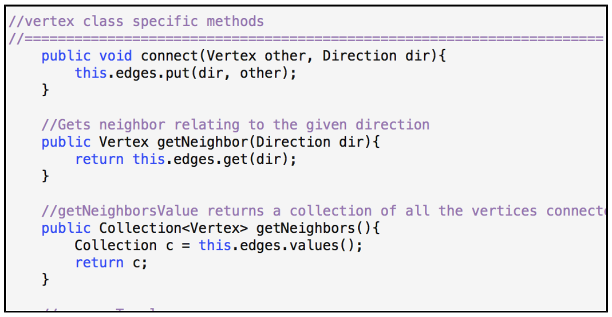
This first task involved creating and editing the Vertex and Graph classes. First, in the graph class I kept an ArrayList of the vertices. I also implemented a number of accessor methods for the vertices in the graph. Also, I implemented the required shortestPath and addEdge methods. Here is what my shortest path looked like:



```
//shortestPath method
public void shortestPath(Vertex v0){

    // Given: a graph G and starting vertex v0 in G
    // Initialize all vertices in G to be unmarked and have infinite cost
    for (Vertex v: this.vertices){
        v.setMarked(false);
        v.setCost(Integer.MAX_VALUE);
    }

    // Create a priority queue, q, that orders vertices by lowest cost
    PriorityQueue<Vertex> vertQueue = new PriorityQueue();
    // Set the cost of v0 to 0 and add it to q
    v0.setCost(0);
    vertQueue.add(v0);

    // while q is not empty:
    while (vertQueue.size() != 0){
    //  let v be the vertex in q with lowest cost
    //  remove v from q
        Vertex v = vertQueue.poll();
    //  mark v as visited
        v.setMarked(true);
    //  for each vertex w that neighbors v:
    //      if w is not marked and v.cost + 1 < w.cost:
        for (Vertex w : v.getNeighbors()){
            if (w.getMarked()== false && v.getCost()+1 < w.getCost()){
    //          w.cost = v.cost + 1
                w.setCost(v.getCost()+1);
    //          add w to q
                vertQueue.add(w);
            }
        }
```

Next, I also added three extra methods will the guidance of CP Majgaard that are used later in my HuntTheWumpus class for adding rooms to the landscape.

Now onto the Vertex class. I implemented a number of simple accessor methods. Also, I added the requires specific Vertex class methods--connect, getNeighbor, getNeighbors. Here are the three methods:



```
//vertex class specific methods
//======================================================================
    public void connect(Vertex other, Direction dir){
        this.edges.put(dir, other);
    }

    //Gets neighbor relating to the given direction
    public Vertex getNeighbor(Direction dir){
        return this.edges.get(dir);
    }

    //getNeighborsValue returns a collection of all the vertices connect
    public Collection<Vertex> getNeighbors(){
        Collection c = this.edges.values();
        return c;
    }
```

==Task 2==

This task was to create the Hunter class. This was very simple. It simply extended Cell and contained a couple accessor methods like getLocation and setLocation.

==Task 3==

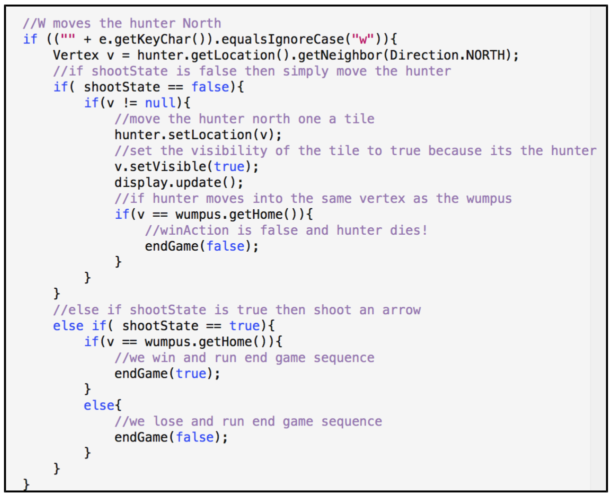
This task was to create the Wumpus class. This was also simple. It simply extended Cell and contained a couple accessor methods like getHome, isVisible and setVisible.

==Task 4==

This was the most tedious and time-consuming task --creating the HuntTheWumpus class. I was instructed to setup the UI and generate a game with the Hunter, the Wumpus, the Landscape and the rooms all in the HuntTheWumpus constructor, but instead I broke it up into a couple methods. I created at setupUI method. Then, I created a generateGame() method that basically generated all the necessary aspects to the HuntTheWumpus game. I called both of these methods in the constructor.

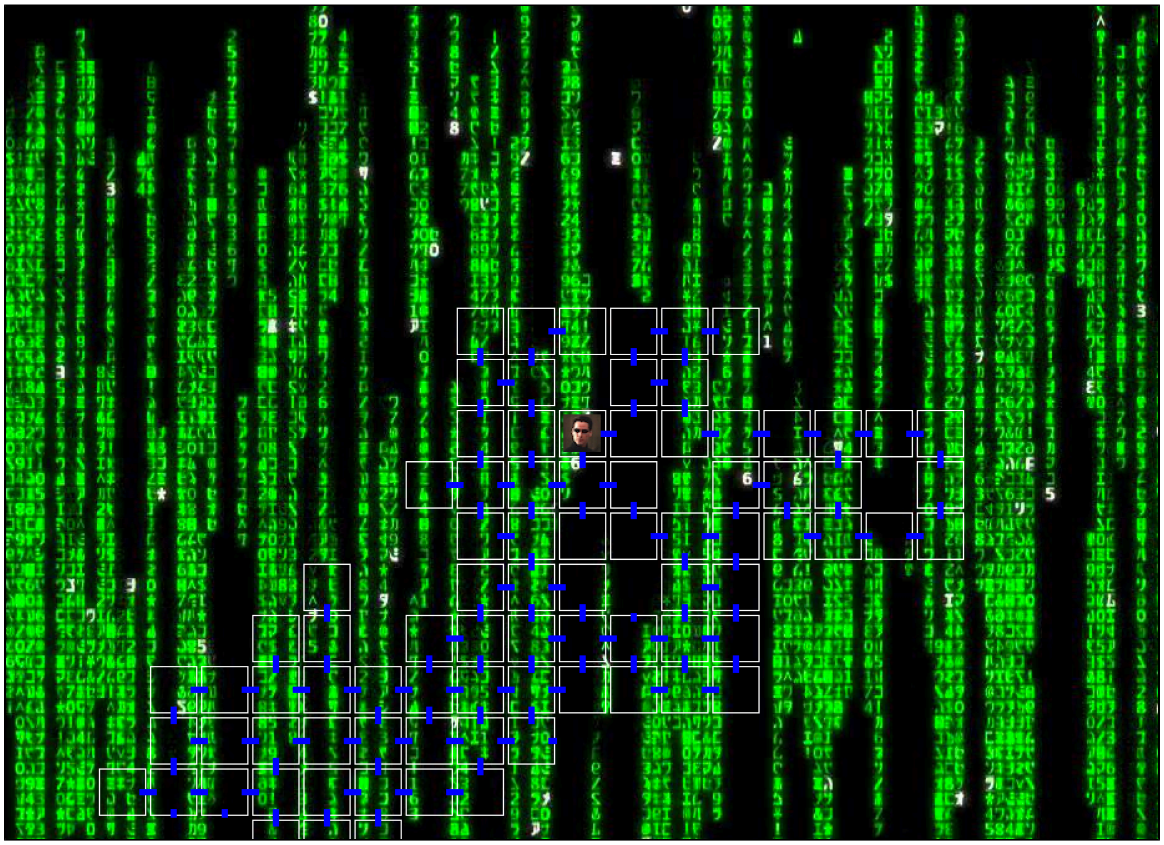
Then, I created two extra but useful methods with the help of CP called addRooms and endGame. The addRooms method basically filled the entire landscape with rooms! It generated a random direction and then checked the next vertex in that random direction. If there was already a room there, then just add an edge, but if there isn't a room there than add a new vertex and add an edge to connect it. The method also checks for the error case of trying to create a new vertex outside the landscape. The endGame method simply tells the computer what to do when the Hunter either wins or loses. In its different cases, true or false, the endGame class will generate a box stating the outcome, terminate the window, and regenrate and open an entirely new and random game!

Third, the most important part of this class was the Control class that extends KeyAdapter. I was able to use WASD as the movement keys and the SPACE bar as the shooting arrows key. I added a toggleShooter method for an extension that allows the user to toggle the arrow option. So, if you accidentally press the space bar, you can then press it again before pressing a WASD direction and the user will unselect the arrow. Also, in each Key stroke conditional, I added a shootState conditional that decided whether or not the Hunter has won after he shoots and calls the endGame function accordingly. Here is an example of one of my key code snippets:

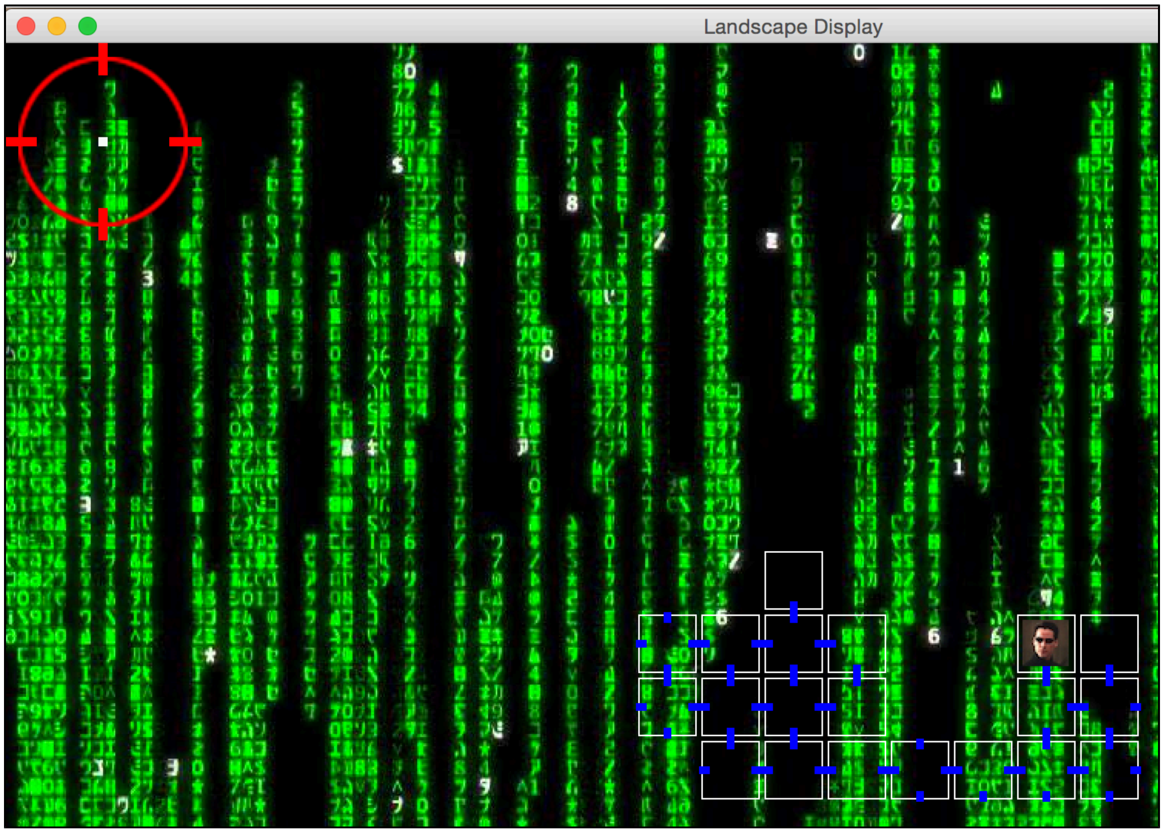


```java
//W moves the hunter North
if (("" + e.getKeyChar()).equalsIgnoreCase("w")){
    Vertex v = hunter.getLocation().getNeighbor(Direction.NORTH);
    //if shootState is false then simply move the hunter
    if( shootState == false){
        if(v != null){
            //move the hunter north one a tile
            hunter.setLocation(v);
            //set the visibility of the tile to true because its the hunter
            v.setVisible(true);
            display.update();
            //if hunter moves into the same vertex as the wumpus
            if(v == wumpus.getHome()){
                //winAction is false and hunter dies!
                endGame(false);
            }
        }
    }
    //else if shootState is true then shoot an arrow
    else if( shootState == true){
        if(v == wumpus.getHome()){
            //we win and run end game sequence
            endGame(true);
        }
        else{
            //we lose and run end game sequence
            endGame(false);
        }
    }
}
```

Extension 1. For this first extension, I simply made my game more visually pleasing by using Neo and Agent Smith icons and a Matrix background. Heres what it looked like:



Extension 2. My second extension was my toggleShooter class and adding a image to my toggleShooter class.* *When I pressed the space bar I was able to toggle the shooter and when I was in shooting mode a red cross-hairs will appear in the top right of the landscape so the user will know! Here what it looks like!:



Extension 3. My third extension included assistance from CP Majgaard and the creating of the addRooms method to create totally randomized landscapes for each game!

What I learned. I successfully learned how to implement a Graph and make a HuntTheWumpus game! I received more experience with GUI and KeyAdapter code as well as using .png images to make my projects more visually enjoyable. In the end, this was a fun final project where I felt that I learned the basics of a graph and reinforced my Control and GUI knowledge.

Who helped me. I worked alongside Steven Parrot and Brendan Doyle during this project. I received help from CP Majgaard throughout the Project 9. Also, I worked with Julia Saul.