**GRID TICKET SYSTEM**

**Quick Note**

My code is viewable in the “src” folder of the unzipped “GridTicketSystem” directory.

**Installation Instructions**

**Windows**

1. Unzip the folder
2. In the Windows search menu search “cmd” and hit enter
3. Using the “*cd”* command navigate to the directory where you’ve unzipped GridTicketSystem.
4. Type “*cd* GridTicketSystem” to enter into the directory.
5. Type “*cd* bin”.
6. Type “java GridTicketSystem”, this will initialise the program
7. You may now enter your x and y coordinates in the form “x,y”, doing so will list the cheapest priced ticket of the five nearest events calculated in Manhattan distance.

**Mac/Linux**

1. Unzip the folder.
2. Open up terminal.
3. Using the “*cd”* command navigate to the directory where you’ve unzipped GridTicketSystem.
4. Type “*cd* GridTicketSystem” to enter into the directory.
5. Type “*cd* bin”.
6. Type “java GridTicketSystem”, this will initialise the program
7. You may now enter your x and y coordinates in the form “x,y”, doing so will list the cheapest priced ticket of the five nearest events calculated in Manhattan distance.

**Assumptions I’ve Made**

1. Each event can hold a maximum of 5 differently priced tickets.
2. The range of a ticket price varies between $1 and $400.
3. 10 different events are randomly generated across the grid each time the program runs.

**How might you change your program if you needed to support multiple events at the same location?**

If there was need to support multiple events at the same location, the *Coordinate* class could be altered so that the variable *event* was changed to an array. The array could hold multiple events as opposed to one.

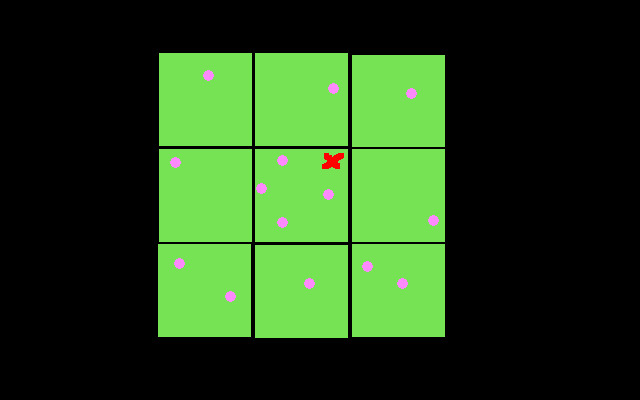
**How would you change your program if you were working with a much larger world size?**

The simplest way of accommodating this in my code would be to change the values of the *min* and *max* variables in the main method from -10 & 10 to whatever number was necessary. However after a certain point doing this would be very inefficient, calculating the distance from your coordinates to the nearest events in a much larger world size would be very computationally taxing.

My proposal would be to split the large world into smaller sub-worlds e.g if you were dealing with a world that was 10000x10000, split the world into 1000 worlds of 100x100, each world with it’s own unique identifier. When the user enters a coordinate the system would know which world the coordinate exists to, therefore it could drastically narrow the scope required to search for nearest events.

The main consideration with this implementation would be how you would deal with coordinates that are on the boundaries between 2 subworlds. Essentially the sub-worlds would have to have a knowledge of where they are in the overworld and the sub-worlds that they are adjacent to.

When searching for the 5 nearest events, all 8 adjacent subworlds should be searched to determine the closest events e.g.

****

In this diagram green squares represent 9 100x100 subworlds, pink dots represent events and the red cross represents the input coordinates. The system would check the distance from the coordinates to all the events in these 9 subworlds to list the 5 nearest. This diagram would look slightly different if we were dealing with subworlds that were on the edge or corner of the overworld as we’d have to deal with 6 or 4 subworlds respectively.