PROJECT REPORT

Viagogo Coding Challenge

Abstract

Randomly populating a map with events and measuring the distance between them and a given user value.

Owen O'Donnell

owenodonnell87@gmail.com

Statement of System Purpose / Brief:

The purpose of this program is to take two coordinates given by a user and cross-reference them with the location on a randomly-populated map. The program will then tell the user what the five closest events to that location are, the distance between their location and the event, and inform the user of the ticket price for each event.

Event List:

1) The system creates a 2D array, comprised of Event Objects.
2) Using the Modulus operator and a randomly generated number, each location is populated with either an Event built from the Event class, or a non-event is placed in the array, created from the Event class's no-argument constructor.
3) Each event's location in the array is stored as a value, using the Event class's getDistance() method.
4) The user is asked to enter an X coordinate. This value is used as a variable to know how far into the outer-array to loop.
5) The user is asked to enter a Y coordinate. As with the previous event, this is used to know how far into the inner-array to loop.
6) The system informs the user of their location and if there is an event there.
7) The system then prints out all of the events on the map, and their distance from the user's location.
8) The events are placed in an ArrayList and are sorted in an ascending order, based on the value stored in the distance field.
9) The system prints the five events closes to the user's coordinates to the console.

Event Response table:

EVENT	**SYSTEM RESPONSE**
Program is executed	2D array of Event Object is generated
	Events are placed at random throughout 2D array
	Event's location in 2D array is stored in the Event's object instance as a variable
	The 2D array is printed to the console as a grid / map
User is asked to enter an X axis co-ordinate	System uses a Scanner to take in user's input
	User's input is stored as a variable
	First array of the 2D array is iterated through that amount of times
User is asked to enter a Y co-ordinate	System uses a Scanner to take in user's input
	User's input is stored as a variable
	Second array of the 2D array is iterated through that amount of times
	The system then prints the user's chosen co-ordinates to the console, and reveals that location's event name, using the Event object's getEventName() method

	The system searches through the 2D array and adds all events to an ArrayList. This is done by checking to see if the event has an ID number.
	The user's location is compared to the locations of the other events. The distance between the two is measured in "hops", and stored as a variable in the other events.
	The events in the ArrayList are sorted in ascending order, based on the value stored in the distance field.
	The sorted ArrayList values are placed in a HashSet, to remove duplicate entries
	The HashSet values are then placed back into an ArrayList, so the Object values can be used
	A loop is used to return the five closest events.

UML Class Diagrams

Event
-eventID : int -eventName: String -ticketPrice: float -xLocation: int -yLocation: int -distance: int

+getEventID( )
+setEventID( )
+getEventName( )
+setEventName( )
+getTicketPrice( )
+setTicketPrice( )
+getXLocation( )
+setXLocation( )
+getYLocation( )
+setYLocation( )
+getDistance( )
+setDistance( )

Run
-rows: int -columns: int -i: int -j: int -eventList: Event[ ] -eventMap: Event[ ][ ]

+getRandomEvent( )
+main( )

Code Snippets / Logic Explained

Building the Map

```
for(i=0; i < rows; i++){
```

```
            for(j=0; j < columns; j++){
                  Random rn = new Random();
                  int randomNumber = rn.nextInt(9-0+1)+0; // Create a random
number between 0 and 10
                  if(randomNumber % 4 == 0){ // If it is divisible by 4, then
                  Event re = getRandomEvent(eventList); // Select a random
event from the event array
                  eventMap[i][j] = re; // then populate this location with
the event
                  System.out.print(re.getEventID() + " ");
                  }else{ // Otherwise, put a non-event in the location
                        eventMap[i][j] = n;
                        System.out.print(n.getEventID() + " ");
                  }
            }
            System.out.println("");
        }
        System.out.println("");
```

Finding All Events on the Map

```
// LOCATING ALL OF THE EVENTS ON THE MAP
        for(i=0; i < rows; i++){
            for(j=0; j < columns; j++){
                  if(eventMap[i][j].getEventID() > 0){ // If the event's ID
number isn't 0, then it's an event
                        eventMap[i][j].setXLocation(i); // Store the event's
location as a field in the event
                        eventMap[i][j].setYLocation(j);
                        events.add(eventMap[i][j]); // add the event to the
"events" ArrayList
                  }
            }
        }
        System.out.println("");
```

Find the Distance Between User Location and All Events

```
for(Event event:events){ // for each event in the events arraylist
            if(eventMap[yLocation][xLocation].getXLocation() >
event.getXLocation()){ // if the X location in the eventmap event is bigger
than the X location stored in the event
                  xSteps = eventMap[yLocation][xLocation].getXLocation() -
event.getXLocation(); // subtract
            }else{
                  xSteps = event.getXLocation() -
eventMap[yLocation][xLocation].getXLocation();
            }
            if(eventMap[yLocation][xLocation].getYLocation() >
event.getYLocation()){ // perform the same with the Y locations
                  ySteps = eventMap[yLocation][xLocation].getYLocation() -
event.getYLocation();
            }else{
                yScteps = event.getYLocation() -
eventMap[yLocation][xLocation].getYLocation();
            }

            int distance = xSteps + ySteps; // the distance between
            event.setDistance(distance);
            distanceList.add(event);
        }
```

Sorting Events by Distance

```
Collections.sort(distanceList, new Comparator<Event>(){
                  @Override
                  public int compare(Event eventDistance1, Event
eventDistance2) {
                        for(int searchDistance = 0; searchDistance <
distanceList.size(); searchDistance++){
                              if(eventDistance1.getEventID() ==
eventDistance2.getEventID() && eventDistance1.getDistance() ==
eventDistance2.getDistance()){
                                    continue;
                              }
                        }
                        return eventDistance1.getDistance() -
eventDistance2.getDistance();
                  }
        });
```

Put the sorted events into a Hash Set to remove duplicate entries

```
LinkedHashSet<Event> uniqueEvents = new LinkedHashSet<Event>();
        for(int ue = 0; ue < distanceList.size(); ue++){
            uniqueEvents.add(distanceList.get(ue));
        }
```

Find the Five Closest Events

```
System.out.println("THE FIVE NEAREST EVENTS TO YOUR LOCATION ARE: ");
        ArrayList<Event> sortedEventList = new
ArrayList<Event>(uniqueEvents);
        for(int p = 1; p < 6; p++){
            System.out.println("The " + sortedEventList.get(p).getEventName()
+ " is " + sortedEventList.get(p).getDistance() + "km away.");
            System.out.printf("Tickets are $%.2f",
sortedEventList.get(p).getTicketPrice());
            System.out.println("\n");
        }
```

Assumptions Made

The project requirements asked for an X and Y axis ranging from -10 to 10. For a map program, a user's position would generally be 0,0. Since this project asked that the user be able to enter their position, I used a 2-dimensional array to create the map instead, which places index 0,0 in the top-left position of the map. To give it the specified size, I have set the rows and columns to 21 (-10, a 0 position, and +10).

For the ticket prices, I gave each event one price. More could be added easily, and the cheapest price could be returned with simple value comparison.

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

A simple solution would be to have the index location store an array of the events being held there, and display the contents to the user. They could then enter the ID number for the event they wish to learn more about, and be shown the information.

The Event class itself could also have more fields, such as eventDate and startingTime, to make the Event's time even more specific.

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

I have set the "row" and "column" values to variables, so the simplest way to increase the size of the map is to change their values. The project requirements stating it wanted a map ranging from -10 to 10, so I have set their values to 21 (this includes a value for what would be 0 in an X and Y map).

For this project, I simply created a handful of Event objects. On a bigger scale, it would be better to incorporate a database, as you could potentially be storing thousands of Event objects.