(http://www.boisestate.edu)

{ CS121 }
Fall 2017
(/~cs121/)

P5: Minewalker

Dec 06, 2017 (by 11:59pm)

Note: There may be *minor* changes to the write-up over the weekend, but the general project idea will be the same.

Contents

Project Overview

In this project, you will create a MineWalker game that lets a user start from the lower-left corner on a grid and attempt to walk to the upper-right corner while avoiding hidden mines.

Classes that you will create

- `MineWalker.java` *(driver)*
- `MineWalkerPanel.java` *(and, potentially, other classes)*

Existing class that you will use

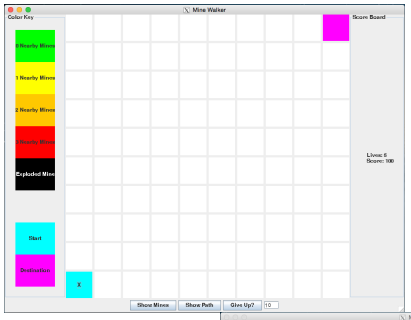
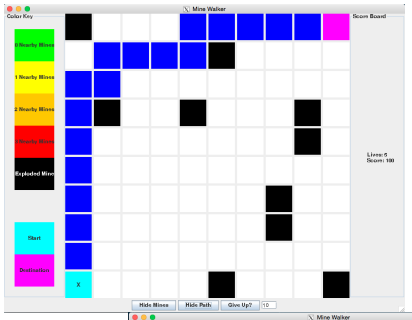
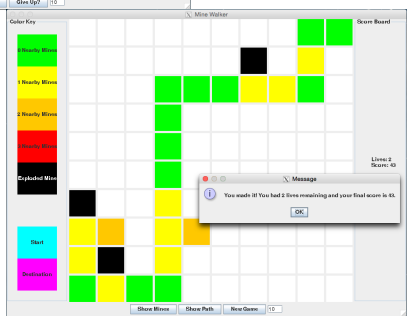
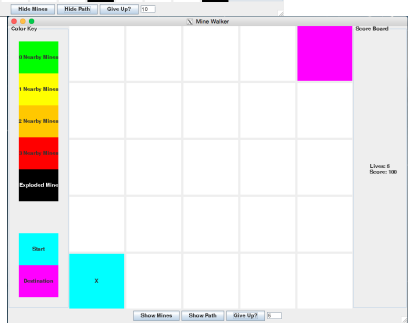
- `RandomWalk.java` (you will use your version from p3 (http://cs.boisestate.edu/~cs121/projects/p3/))

Objectives

- Write a program with a Graphical User Interface (GUI)
- Design a good user interface
- Use classes from the AWT and Swing packages
- Use existing classes
- Use events and listeners

Specification

- **User interface design.** Design a layout for the user interface. This is best done on a white board or on paper!
  - The main visual element will be the grid of tiles that represents the mine field. This can be a two-dimensional array of `JButton` objects.
  - You will also need buttons for starting a new game, a text field for the grid size, buttons for showing/hiding the mines and labels for the current score and number of lives as well as other optional elements.
  - Plan on changing labels on buttons and disabling them when it doesn't make sense to use them. For example, the grid size cannot be changed while the game is going on. Once the user clicks on the *Show Mines* button, its label should change to *Hide Mines*. Once the game starts, the label for the *New Game* button should change to *Give Up*. Note that the labels can be different if that makes more sense to you.
  - Sample screenshots:


(images/new-game.png)

(images/show-mines-path.png)

(images/win.png)

(images/small-grid.png)

(Note that you can design your game to look different than these screenshots!)

- **Creating the game**. The *New Game* button creates a new game as follows.
  - Generate a new grid of the size specified by the user (have reasonable limits on the size). The default size is 10x10.
  - Generate a random walk on this grid from the lower-right corner to the upper-left corner. This path will be used to ensure that there is a solution to the game.
  - Generate the hidden mines on the grid. Use a random number generator to place mines. 25% of the tiles that are not in the random walk should be mines. You may allow the user to change from the 25% default value but that is extra credit and not required.
- **Game play**.
  - At each step, the user can move one step to the north, south, west or east of the current position (except around the edges). If a user clicks on a tile that isn't valid, the program doesn't let them move there (no jumping allowed!). The game hints at the number of mines near by with a color code defined as follows:

    - Green: There are zero mines in the four adjacent tiles
    - Yellow: There is one mine in the four adjacent tiles

    - Orange: There are two mines in the four adjacent tiles
    - Red: There are three mines in the four adjacent tiles

  - If the user reaches the goal, the game is over. Notify the user appropriately using a *JOptionPane* popup window . When the game is over, show all the mines on the grid along with the path that the user took. Do not allow the user to play anymore unless they start a new game.

  - Show current square by a blinking tile color or label. For this you need to create a timer thread for the animation (similar to one we used in Project 1 or Project 3 (in the GridMap class)). Use the following code for the *startAnimation()* method and call it from the constructor in *MineWalkerPanel* class. The animation method creates a *Timer* that will periodically fire an ActionEvent that needs to be handled by an *ActionListener*. You need to toggle the color of the button (or its label) in the actionPerformed method. Here is a template for the *startAnimation* method and the corresponding *ActionListener*:

```
        /**
         * Performs action when timer event fires.
         */
        private class TimerActionListener implements ActionListener
        {
                public void actionPerformed(ActionEvent evt)
                {
                        // toggle color/text for current tile
                        // do other periodic checks if needed
                }
        }

       /**
        * Create an animation thread that runs periodically
        */
        private void startAnimation()
        {
            TimerActionListener taskPerformer = new TimerActionListener();
            new Timer(DELAY, taskPerformer).start();
        }
```

  - If the user steps on a mine, they die and lose one life. *The game is over after number of lives reaches zero*. The user gets five lives in total. After each death, the user is reborn on the last tile they were on before stepping on a mine. They can continue walking from that spot. Stepping on an already exploded mine should not result in an additional death, but it is not a valid move. Stay out of the craters.

  - The user can give up on a game anytime by pressing the *Give Up* button.

  - The score and the number of lives are updated after every move in a label somewhere on the display.

- **Scoring**. Come up with a scoring scheme. For example, the initial score can be 500. Each life costs the user a 100 points. Each step costs them 1 point. The goal is to lose as few points as possible. Or invent your own scoring scheme!

- **Show/hide mines**: Add a button that allows the user to show or hide the mines. This is very useful for debugging!

- **Show/hide walk**: Add a button that allows the user to show or hide the original random walk used to create the game.

- **Color key**: Add a useful key so the user knows what the colors mean in the game.

Extra Credit (up to 10 points)

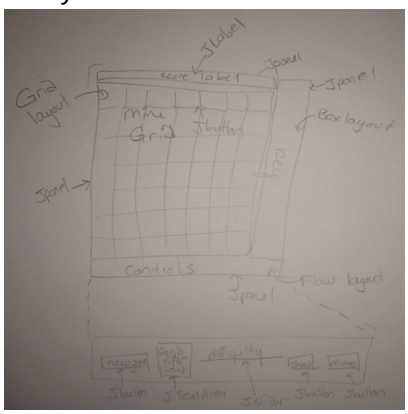
- **Settable level of difficulty**: Provide a sliding bar that allows the user to set the level of difficulty for the game.
- **Sound effects**: Set sound effects for walking, invalid moves, reaching the goal and, of course, for explosions!

Getting Started

- Create a new Eclipse project for this assignment. Import the *RandomWalk.java* class (from your project 3 (http://cs.boisestate.edu/~cs121/projects/p3/)) into your project.

- Create new classes: the driver class, *MineWalker.java*, and main panel class, *MineWalkerPanel.java*.

- Make a rough sketch of your GUI. Remember the provided screenshots and movies are general guidelines and you should not turn in an exact duplicate of what you see. Be creative. Have fun.



(images/sketch.jpg)

- Determine the Java classes for the components in your design and label your sketch.

- Group your sketch into sections (breaking down the design) that will be implemented in the same Java container. Make notes on your diagram about the Layout Manager and Border style you will use for each container.

- Implement the Java code to produce the GUI you sketched above. Working on one section at a time helps to reduce the overall complexity as you can validate smaller sections of code before putting everything together. Since you will have a large number of components in your GUI, choose variables names that will help you to quickly identify which visual element you are currently dealing with in the code.

- Once you have the visual framework in place consider the listeners and how they will interact with each component in your design. Decide if you want a single event listener with a compound conditional or if you want separate listeners to handle each event.

- Implement the game logic and integrate the controls and scoring into the code. Plan early on how you will test and debug each section of the game and ensure you are testing regularly.

- Ensure you have the basic functionality implemented before improving or adding to the game. Extra credit will not make up for problems with the basic functionality!

These examples (available in your Eclipse chap06 project that you should have checked out) should cover most of the technical requirements of the project. The game logic and putting it all together, of course, is up to you.

- MiniColorChooserV1 (http://cs.boisestate.edu/~cs121/examples/chap06/MiniColorChooserV1.java): 1D GUI controls array, getting properties from JButtons and updating another control accordingly
- MiniColorChooserV2 (http://cs.boisestate.edu/~cs121/examples/chap06/colorchooser): 2D GUI controls array, indexes corresponding to button coordinates in a GridLayout, coordinating with a parallel 2D array
- ButtonTextCycler (http://cs.boisestate.edu/~cs121/examples/chap06/ButtonTextCycler.java): cycles through a sequence of button names as a button is pressed
- LightBulb: the toggling light bulb example from the book, controls enable/disable, coordinating events between panels, mnemonics, tool tips
  - LightBulb.java (http://cs.boisestate.edu/~cs121/examples/chap06/LightBulb.java)
  - LightBulbControls.java (http://cs.boisestate.edu/~cs121/examples/chap06/LightBulbControls.java)
  - LightBulbPanel.java (http://cs.boisestate.edu/~cs121/examples/chap06/LightBulbPanel.java)
  - LightBulbOff.gif (http://cs.boisestate.edu/~cs121/examples/chap06/lightBulbOff.gif)
  - lightBulbOn.gif (http://cs.boisestate.edu/~cs121/examples/chap06/lightBulbOn.gif)
- GameBoard.java (http://cs.boisestate.edu/~cs121/examples/chap06/GameBoard.java) and GameBoardPanel.java (http://cs.boisestate.edu/~cs121/examples/chap06/GameBoardPanel.java) : replacing a panel in a running application, 2D array of controls corresponding to coordinates in a GridLayout, borders, getting/setting numbers in JTextFields, validating text field input

Submitting Your Project

Documentation

Javadoc Comments

If you haven't already, add **javadoc comments** to your program. They should be located immediately before the class header and before each method. If you forgot how to do this, go look at the Documenting Your Program (http://cs.boisestate.edu/~cs121/labs.php?lab=01#docs) section from lab.

- Have a class javadoc comment before the class.
  Your class comment must include the `@author` tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Have javadoc comments before *every* method that you wrote. Comments must include `@param` and `@return` tags as appropriate.
- To build and view your comments, run the following commands.

```
javadoc -author -d doc *.java
google-chrome doc/index.html
```

README

Include a plain-text file called **`README`** that describes your program and how to use it. Expected formatting and content are described in README_TEMPLATE (https://raw.githubusercontent.com/BoiseState/CS121-resources/master/projects/README_TEMPLATE.md). See README_EXAMPLE

(https://raw.githubusercontent.com/BoiseState/CS121-resources/master/projects/README_EXAMPLE.md) for an example.

Submission

You will follow the same process for submitting each project.

1. Open a console and navigate to the project directory containing your source files,
2. Remove all the `.class` files using the command, `rm *.class`.
3. In the same directory, execute the submit command for your section as shown in the following table.
4. Look for the success message and timestamp. If you don't see a success message and timestamp, make sure the submit command you used is EXACTLY as shown.

Required Source Files

Required files (be sure the names match what is here exactly):

- MineWalker.java
- MineWalkerPanel.java
- RandomWalk.java
- any other files necessary to compile and run your program
- README

Section	Instructor	Submit Command
1	Jerry Fails (TuTh 1:30pm - 2:45pm)	`submit JERRYFAILS cs121-1 p5`
2	Greg Andersen (TuTh 9:00am - 10:15am)	`submit GregoryAndersen cs121-2 p5`
3	Luke Hindman (TuTh 10:30am - 11:45am)	`submit LukeHindman cs121-3 p5`
4	Marissa Schmidt (TuTh 4:30pm - 5:45pm)	`submit MarissaSchmidt cs121-4 p5`
5	Marissa Schmidt (TuTh 3:00pm - 4:15pm)	`submit MarissaSchmidt cs121-5 p5`
6	Jim Conrad (TuTh 12:00pm - 1:15pm)	`submit JimConrad cs121-6 p5`

After submitting, you may check your submission using the "check" command. In the example below, replace <instructor> with your instructor and <x> with your section.

```
submit -check <instructor> <section> p5
```