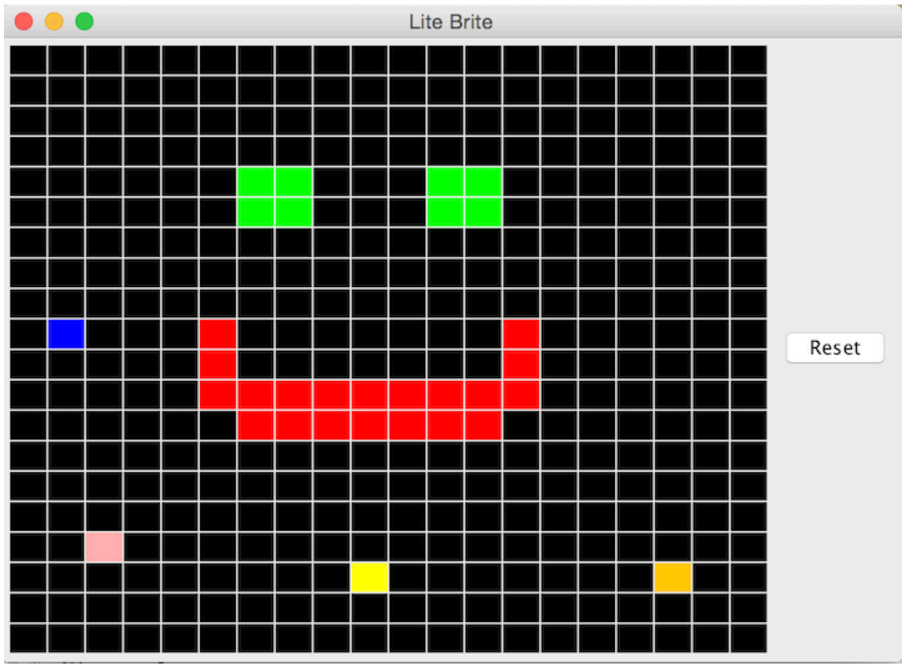
Activity 8.1 - LiteBrite

Overview

In this activity, you will create a GUI modeled after Lite-Brite. This activity also helps to reinforce 2D array functionality.



Objectives

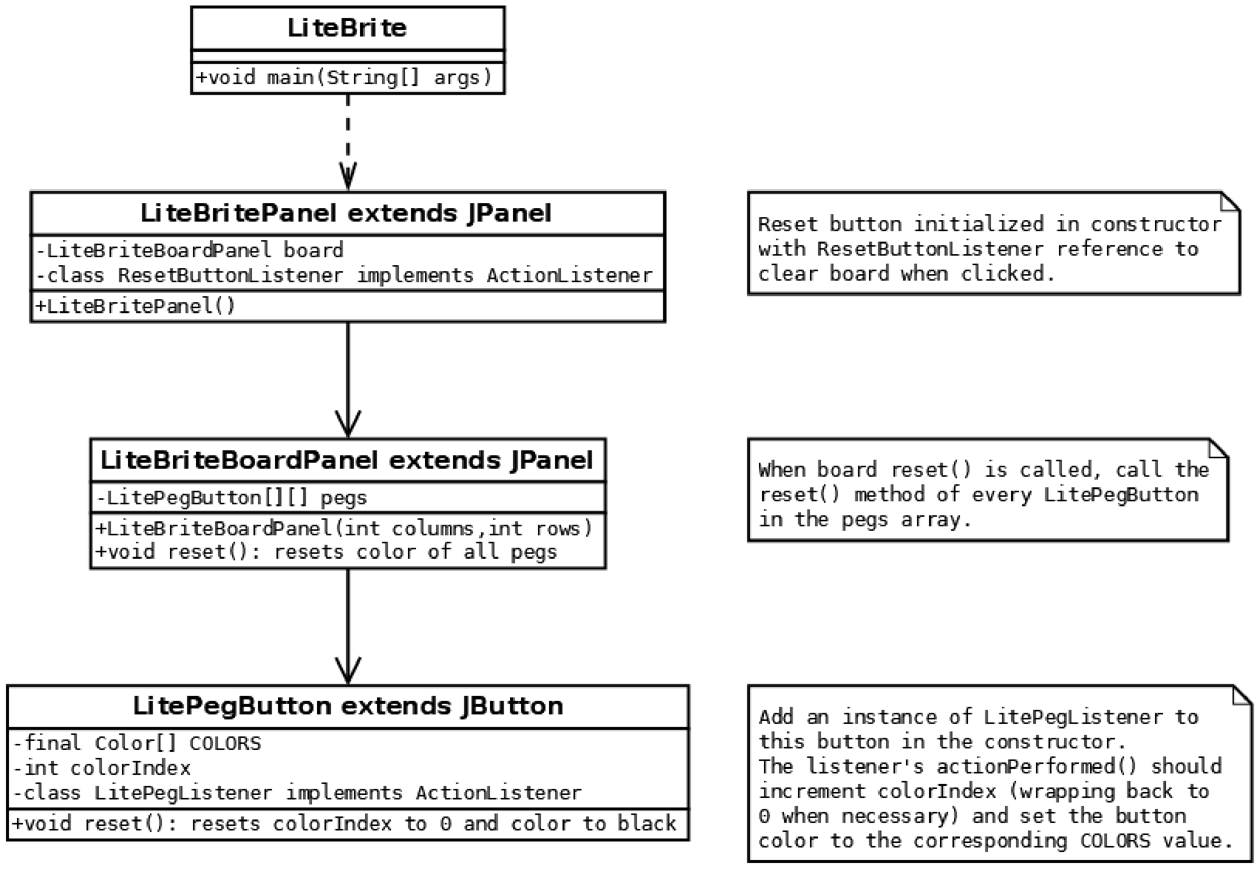
- Manage the GUI layout using sub-panels.
- Write classes that effectively manage program data and delegate responsibility for program logic.
- Use a `GridLayout` for displaying a 2D array of objects.
- Implement and add `ActionListeners` to GUI components to handle click events.

Instructions

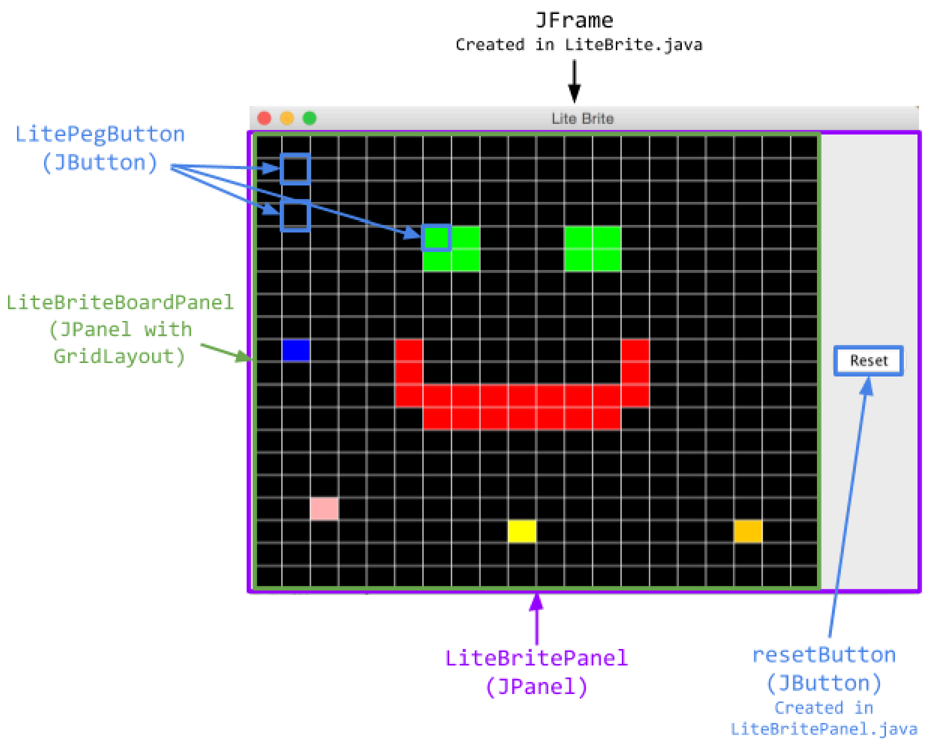
Getting Started

1. Create an "`Activity8.1`" project in VS code.
2. Import the existing `LiteBrite.java` driver class and `LiteBritePanel.java` class. Note: The `LiteBritePanel` class will not compile until you implement your other classes.
3. You will write your own `LitePegButton` and `LiteBriteBoardPanel` classes.
4. Review the following UML diagram and layout guide as you develop this program:

This UML diagram shows the main `LiteBrite` classes and their instance variables, methods, and inner listener classes. Dotted arrows indicate dependency/uses relationships and solid arrows indicate aggregation/has-a relationships between the classes. Private inner `ActionListener` classes are shown as attributes of their containing classes for simplicity.


LiteBrite
+void main(String[] args)
LiteBritePanel extends JPanel
-LiteBriteBoardPanel board
-class ResetButtonListener implements ActionListener
+LiteBritePanel()
Reset button initialized in constructor with ResetButtonListener reference to clear board when clicked.
LiteBriteBoardPanel extends JPanel
-LitePegButton[][] pegs
+LiteBriteBoardPanel(int columns,int rows)
+void reset(): resets color of all pegs
When board reset() is called, call the reset() method of every LitePegButton in the pegs array.
LitePegButton extends JButton
-final Color[] COLORS
-int colorIndex
-class LitePegListener implements ActionListener
+void reset(): resets colorIndex to 0 and color to black
Add an instance of LitePegListener to this button in the constructor.
The listener's actionPerformed() should increment colorIndex (wrapping back to 0 when necessary) and set the button color to the corresponding COLORS value.

Below is a screenshot of the final `LiteBrite` GUI layout with its various components labeled and outlined.



LiteBrite GUI Layout

The following example may be a helpful reference when completing this activity: [MiniColorChooser.java]

LiteBrite and LiteBritePanel classes

The `LiteBrite` driver class is already complete. It creates a new `JFrame` and adds a `LiteBritePanel` to the frame.

The `LiteBritePanel` class is mostly complete, but will be configured to first hold a `LitePegButton` during its development and then a `LiteBriteBoardPanel` in its final form. `LiteBritePanel` extends `JPanel`, which means it is a `JPanel`. Its purpose as the top-level panel is to organize and coordinate between sub-panels and components. Note: It will not compile until you have implemented the classes it references.

Part 1: LitePegButton class

- Create class `LitePegButton` that extends `JButton`. It will represent the state of a single colored peg. Implement the features shown in the class design UML diagram and as described here.
    - `LitePegButton` should have a constant `COLORS` array with several colors. Use an initializer list to create this array. The first color must be `Color.BLACK`. Choose three or four additional `Colors` for the rest of the values. Recommend red, green, blue, and yellow.
    - `LitePegButton` should have a single instance variable: int colorIndex. In the constructor, initialize this index to 0 and set the button's initial color to black.
    - In the constructor, use this.addActionListener() to register an instance of the button's `LitePegListener` to receive click events for this button.
    - Write private inner class LitePegListener implements ActionListener. Its `actionPerformed()` method should advance the colorIndex and set the button's color to the corresponding color from `COLORS`. The index should loop back to 0 when you've run out of valid indexes.
    - Write a public void reset() method that sets the color index to 0 and the color back to black.
- Add one `LitePegButton` to the `LiteBritePanel` to test that the functionality of a single button works correctly. Clicking this button should cycle through the colors defined in its `COLORS` array.

NOTE: If you are working on a MacOS system, you'll need to set the following options inside the constructor of the `LitePegButton` for the backgrounds to be displayed properly.

```
    this.setOpaque(true);
    this.setBorderPainted(false);
```

Part 2: LiteBriteBoardPanel class

- Create class `LiteBriteBoardPanel` extends `JPanel`. It will manage a grid of `LitePegButtons`.

- Update `LiteBritePanel` to add a new `LiteBriteBoardPanel` rather than a single `LitePegButton`.
- The only instance variable of `LiteBriteBoardPanel` should be a 2D array of `LitePegButtons`.
- The `LiteBriteBoardPanel` constructor should take the width and height of the grid as parameters, corresponding to the number of columns and rows in the grid. Set the layout manager of this panel to a `GridLayout` with the given dimensions. Also initialize the 2D array of `LitePegButtons` with these dimensions. Populate the array **and** the layout with `LitePegButtons`.
  Note: The array of buttons is needed because while the GridLayout organizes components into a grid, you cannot access those components by coordinates after they are placed. You need the array to be able to access your button references by their location later in the program.
- To get the buttons to be square, you need to set the preferred size for each `LitePegButton` to square dimensions (e.g. 30x30).
- Implement the board's `reset()` method to call the `reset()` method of every `LitePegButton` in the array.
- Confirm that the grid of `LitePegButtons` is correctly displayed, that every peg independently cycles through its colors when clicked, and that clicking the Reset button causes all pegs to reset to black, after which they still cycle correctly through colors.

Terminology Identification

In your code add comments identifying examples of the following: event, listener, component, inheritance. These should be identified with an @keyterm tag within the comment.

Code Review

When you are finished with this activity, pair up with a classmate and review each other's code to make sure it meets all the requirements.

Submission

After completing the assignment, use the assignment link in Canvas and follow the submission instructions there. You will upload your .java files and put your reflection in the “Comments” box.

Reflection Requirements

Write a one paragraph reflection describing your experience with this activity. The reflection should also include the name of your code review partner AND something interesting you found in their code. Please review the activity rubric for more details.