Color Sweeper Documentation

Directory Structure

Color Sweeper has a basic directory structure for its files. There is a single package which stores all the files and directories. The package has a source folder which stores all the java class files and java files. The package also has an images folder which stores the images used in the game. There is also a scrum folder which holds some documentation of user stories and bugs in the main package. Apart from the source and scrum directories, the main package has some documentation files such as a README file and this documentation file.

Types of Objects

Color Sweeper follows an object-oriented programming style. The Model component is represented by the 3 Box classes and Mine Model. The Controller component is represented by the different BoxStrategies each Box has. Finally, the View component is represented by the different Screens and panels.

1. Box Class and Subclasses

There are 3 main types of Boxes which are square buttons in the game: Bomb, Number and WhiteSpace. There is a class for each of these and each inherits from the abstract class Box which inherits from the JavaFX Button class. These Box objects have the following methods: reveal, unflag and flag. The method reveal disables the button and reveals a possible image: a bomb for Bomb, a number indicating the number of adjacent bombs for Number and nothing for WhiteSpace. The methods flag and unflag add a flag and remove a flag image from the Box respectively. Each Box also stores the x and y coordinates corresponding to its position in the grid.

2. Mine_Model

The class Mine_Model makes the Boxes and puts them in a double array called Box_Grid. This class contains the main methods needed to store and update the game. These methods are: createAllBoxes, createGrid, assignBombs, assignNumbers, countAdjacentBombs, generateColorsets, getAverageColor and the getters and setters for all private variables. The Mine_Model class has reveal and flag so the game can update it's model.

createAllBoxes: This method is called to fill in Box_Grid and initialize any data. The method calls createGrid and assignNumbers to give each Box created the correct values. This method is called at the starting of the game.

createGrid: This method is called to create the WhiteSpaces boxes with the specified dimensions.

assignBombs: This method creates and assigns a random position for each Bomb needed in the game. The method calls generateColorsets and assigns each bomb to a random color set.

The method also hooks the Bomb Strategy class to the Box so the player can interact with the class.

assignNumbers: This method assigns the spaces around each Bomb the total number of adjacent Bombs nearby. This method calls on countAdjacentBombs and getAverageColor to get the appropriate configurations for a specific Box.

countAdjacentBombs: This method counts the adjacent Bombs around a single Box.

generateColorsets: This method generates a randomized set of colors that can be used to distinguish between different Bombs. Then in assignBombs.

getAverageColor: This method returns a colorset representing the average color of all the bombs around it.

3. ColorSet

This class stores the rgb values of a randomly determined color. More so, it stores an ArrayList of Bombs which have the color stored in the ColorSet. The class ColorSet uses the methods addBomb and removeBomb to set up the Bomb ArrayList.

4. BoxStrategy Class and Subclasses

The BoxStrategy class and its subclasses are meant to be handlers for mouse click events. There is a strategy or event handler object for each type of Box: BombStrategy, NumberStrategy and WhiteSpaceStrategy which inherit from the class BoxStrategy and take in the Mine_Model instance. The Mine_Model hooks up each of these strategies to each appropriate Box using BoxFactory. the mousePressed, mouseClicked, and mouseReleased methods update the Smiley to it's appropriate face. A surprised face when the button is being clicked, the glasses face when the player wins, the shocked face when the player loses or the default smile during any other time.

BombStrategy: The mouse events responsible for every Bomb. If the user left clicks, the class calls mine model to reveal all of the bombs and ends the game. Otherwise, If the user right clicks, the class calls mine model to check if flagging the bomb is a safe move.

NumberStrategy: The mouse events responsible for every Number box. If the user left clicks, the class calls mine model to reveal itself. Otherwise, If the user right clicks, the class calls mine model to flag itself.

WhiteSpaceStrategy: The mouse events responsible for the WhiteSpaces. If the user left clicks, the class recursively calls mine model to reveal itself and all the adjacent boxes only up to the first layer of Number boxes. If the user right clicks, the class calls mine model to flag itself.

5. Screens and Panels

There are several screens in the game: Start_Screen, Custom Screen,View, Header End_Screen and HintScreen.

Starter Screen: This panel is the title screen for the game. It contains 4 buttons: Easy, Medium, Hard and Custom which correspond to different game modes the player can play. Clicking on Easy, Medium or Hard sets the pre-made game configurations while Custom allows the player to make their own.

Custom Screen: Contains the features needed for the player to customize their own game.

View: The View is the main game screen which has a Header_Panel on top and a Grid_Panel on the bottom. Header_Panel stores and displays a flag count, a Smiley object and a current color box. The Mine_Model updates the flag count and the model is observed as an Observable class by the View which is an Observer.

Header Panel: Contains the current Color that's being flagged, the Smiley and the total number of flags unused. This class is observing Model and updates all of these components when the model changes.

End_Screen: The end game screen which lets the player exit or play again.

Hint Screen: Contains a special hint which makes the game easier for the player.

Extendability

If the player wants to add new Boxes to the game.

1) Create the class they want to add into the game and have it inherit from Box
2) Create the appropriate mouseEvents handler and have it inherit from BoxStrategy
3) Add the strategy information to the BoxFactory so the model can create the handlers
4) Implement the needed functions in Mine Model
5) Change createAllBoxes to be able to create the class being added

If the player wants to add new Smiley Faces to the game.

1) Add the correct face into the Images folder.
2) Label it String+Smiley so the Smiley class can locate the image
3) Call the updateImage class in Mine Model where the Smiley should change to the image being added