Design Decisions and Data Structure Justification

Ryan Boucher

November 25, 2018

1 Main Data-Structure Changes from Milestone 2

The biggest change in our codebase for this milestone was the addition of an undo/redo functionality into the program.

To achieve this functionality, first, a "Deep Copy" of the data of the board had to be achieved, in order to allow for the state of the board to be written into memory correctly. To achieve this, at the end of every player turn, the game must write the contents of our board, consisting of squares, that consist of pieces, into memory. As such, we have added copy() methods into class Square, as well as every one of our Piece subclasses. These copy methods generate **new objects** that are proper copies of the current state of the board. We copy every square of the board into a new board object (as well as the pieces), and then proceed to store this information into an **Undo Stack**.

When the user clicks the undo button, the undo stack pops, and sets the state of the board to the popped board. Additionally, when this stack is popped, the object is pushed into another stack, the **Redo Stack**. For every sequential undo that is triggered, the redo stack will grow in size. When the redo functonality is called, the redo stack will pop its board object, set the current status of the board to the popped board, and then proceed to push the popped board into the undo stack.

When the user places a piece after performing an undo, the redo stack is wiped.

This functionality allows for a user to perform an undo operation until there are no more possible undos, but, also allow to perform a redo, if they are unhappy with the undo.

Other main changes in this Milestone include:

1. Improved graphics, in the form of grass, and of more plant icons
2. The addition of many more pieces into the game
3. The refactoring of some of our controller class code, to allow for increased readability
4. Complete testing for all new changes, and expanded testing into our classes

2 Java Class: Controller

This class functions as the controller for the project. Additionally, it models the board that the Plants VS Zombies game runs on, and was previously called GameBoard. It holds information on the amount of money that the user currently has, as well as the limit on the number of zombies that will occur on the board (a rough version of difficulty).

The board in this class has been modelled as a two-dimensional array, to allow for the use of rows, and columns. Two dimensional arrays are fast in regards to their access of information, and as the board is small (8 columns by 5 rows), the game has lost little in regards to efficency when it iterates over the board.

This class has been expanded to allow for the use of an undo/redo function, as discussed above.

As a controller, this class assigns an ActionListener to each square of the board, and listens for the user to click on the popup to place a plant on the board, before executing the appropriate code.

3 Java Class: View

This class functions as the view for this project, also known as the GUI. It renders the board, plants, and zombies that the player will play the game on, and is responsible only for rendering. It renders based off of the current state of the board, which is controlled by the Controller class.

There are no significant data-structures to discuss in this class.

1. Coordinate
2. Peashooter
3. Piece
4. PlantPieces
5. Square
6. Sunflower
7. Zombie

4 Java Class: PlantPieces

This class allows for text-based representation of the pieces on that will be used in the game. This is primarily used for milestone one, in which we use a text-based display of the basics of the game.

There are no significant data-structures to discuss in this class.

5 Java Class: Piece

This class models pieces used in this game, and includes information on individual piece health, damage, cost, and both the name, and short-name (char) that is used to represent the piece. Pieces will exists as objects in this design, and as such, it makes logical sense to model them here.

There are no significant data-structures to discuss in this class.

6 Java Class: Coordinate

This class models coordinates that will be used to identify locations on the board, through columns (X values) and rows (Y values). This is necessary to allow for easy identification of where pieces are in the game.

There are no significant data-structures to discuss in this class.

7 Java Class: Square

This class models the individual squares that make up the game board. It is used when adding and removing pieces, and will identify if a square is currently being occupied or not. This class contains the logic for adding and removing pieces on the board, and is integral for the game.

There are no significant data-structures to discuss in this class.

8 Java Class: Main

This class intializes the MVC modelling used for this game, and does the intial calls to View (the GUI) and Controller (the controller).

There are no significant data-structures to discuss in this class.