Design Decisions and Data Structure Justification

Ryan Boucher

November 8, 2018

1 Main Data-Structure Changes from Milestone 1

The biggest change in our codebase for this milestone was the conversion of our project to the Model, View, Controller methodology of coding design. This required significant changes in the structuring of our code in regards to how our controller would work, the creation of a view class, and the proper use of a model.

In regards to our controller, we changed the class that was previously known as GameBoard, into controller. As it already controlled the board actions, this made sense in regard to getting our project into MVC format.

For our model, we used the plant pieces to hold our data.

For our view, we used a standard implementation of a java GUI, with approrpriate uses of buttons, and popups.

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.

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.

4 A Note on Model

This project does not currently have a model as a class in itself. However, this project still follows the MVC format, using the following classes as the ”model” component of MVC:

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

5 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.

6 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.

7 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.

8 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.

9 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.