Jeff Liu, Michael Mao, Stanley Wang

George Peck

APCS 5

3 May 2018

Functional Specification: Lawn Mower Simulator

*Introduction*

Our program, Lawn Mower Simulator, is a two-dimensional grid-based puzzle game in which the player moves a lawn mower through a field of grass, aiming to mow every piece of grass only once without hitting obstacles, and finishing at the end coordinate. The game supports level creation as well as a short campaign mode where a player plays a few levels with increasing difficulty. Score is tracked in each level; the magnitude and quality of the score depends on the movement strategy of the player.

Lawn Mower Simulator’s first mode is free play, where a blank field of grass is presented for the player to be accustomed to the grid and the arrow key guided movement of the lawn mower sprite to convert grass tiles to dirt tiles.

Its level select opens up the campaign; four levels are preloaded into the game. These levels, in addition to the mower and the grass tiles, contain rock tiles as well as a finish tile. Rock tiles are the impassable obstacles and act similarly to the borders of the game board. The finish tile is how levels are completed; the player must mow all grass tiles to dirt and then navigate into the finish tile to finish the level and assess their score. A grading system A-F is given based on how high the amount of moves a player made to their final score is, as mowing over an already mowed dirt tile subtracts score while mowing fresh grass adds to it.

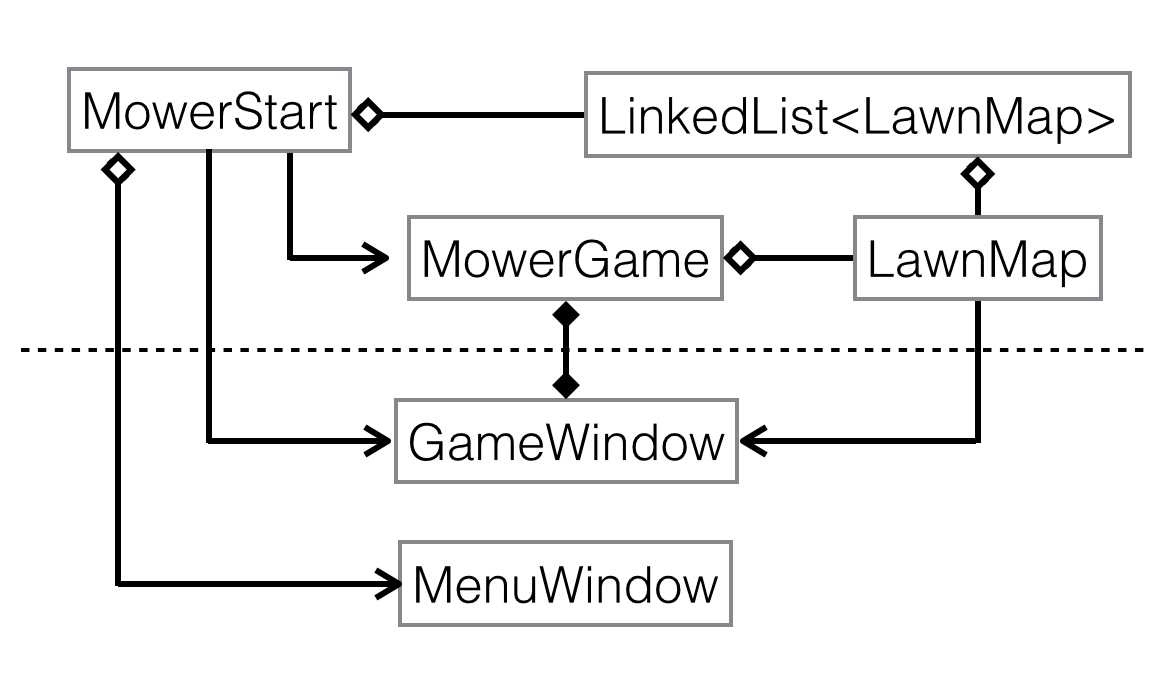
*Structural Design*

To store the data required for this project, we made use of several data structures. In the LawnMap class, we used a 2D array of characters to represent a lawn, with the characters ‘G’ symbolizing grass, ‘B’ symbolizing a boulder, ‘M’ representing the lawn mower, and ‘F’ representing the finish tile. The GameWindow class handles the GUI by managing a 2D array of JLabels alongside the 2D character array that a LawnMap would use.

Several levels were decided as the campaign or level select mode, and we created four to start with. The main class, MowerStart, contains a static linked list of LawnMaps that stores the pre-made levels. We are using Java's GUI, so we used two classes, MenuWindow and GameWindow, to handle the GUI of the game’s menu, and a functionality class, MowerGame, which handles the manipulation of the game such as the score, 2D array that symbolizes the board, and feeds the new information for the GameWindow to display to the player.

*Object-Oriented Design*

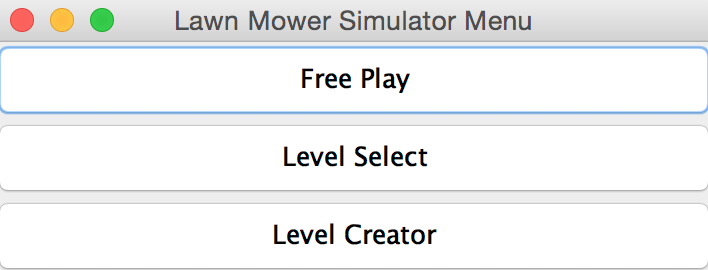
To realize the project, we created five classes: MowerStart, LawnMap, MowerGame, GameWindow, and MenuWindow. They can be separated into purely functional classes as well as GUI classes that handle mostly I/O to the player.

**

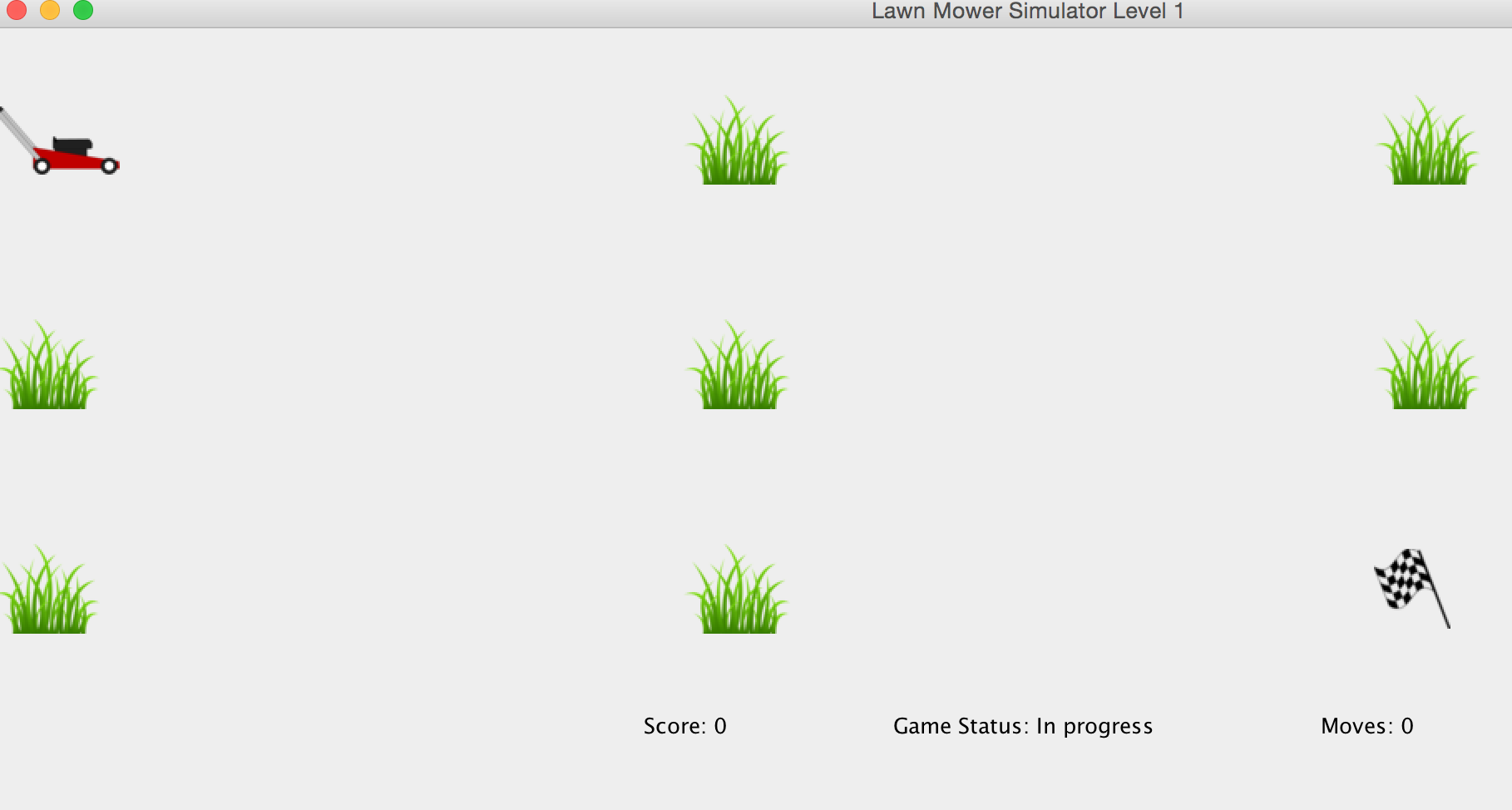
*Classes:*

Our first and most basic class is LawnMap, meant as an abstraction of a lawn represented as a grid of items. LawnMap is an independent class that stores relevant information regarding a lawn or the levels in the game. This information includes a 2D character array containing data of the lawn. The characters will correspond to different items in the grid: ‘G’ represents a square of grass, ‘M’ is the mower, ‘F’ is the flag, the ending position, and ‘B’ is a boulder. In addition, LawnMap has several integer parameters that specify where the mower should start, where the mower should end (startX, startY, endX, endY) and an integer that stores the number of patches of grass in the map. The number of patches of grass is important because the MowerGame class, which takes a LawnMap as a parameter, will be counting the number of squares of grass that have been mowed, and one of the end game conditions is when the number of squares of grass mowed is equal to the to total number of squares of grass. For convenience, we gave LawnMap a no args constructor that creates a default LawnMap consisting of a 10 by 10 grid with the start position in the top left and no end position, to have a blank area for just movement testing.

MenuWindow is the GUI class that creates the game menu. This class allows the user to select a game mode to play. The game menu is a simple GUI with three buttons. The three buttons are free play mode, level select, and level creator. When the free play button is clicked, MenuWindow opens a GameWindow using the default LawnMap as a parameter. A blank 10 by 10 field of grass with a lawn mower will be displayed. When the level select button is clicked, a window will pop up, prompting the user to enter in a level of their choice (enter an integer 1-4), and the level is loaded into a new GameWindow after the user makes their decision. When the level creator button is clicked, a pop-up window asks the user for information regarding the lawn dimensions, where the starting and ending positions are, and where to place boulders (can place up to three boulders). Then, a LawnMap with all the specified properties is loaded for the user to play. Coding the free play button was quite simple, but the other buttons that involved a pop up window were more complicated. We used code similar to the pop up window in StockExchange. The level select ActionListener continually checks the user’s input into a text field in the pop up window asking for the level the user wishes to play. We used a try-catch statement to check if the input was an integer. If not, this will prompt an error. If the user’s input is an integer, then the program checks to make sure that the integer is in bounds (in the range 1 to 4 inclusive) and loads the respective level. The level creator ActionListener works similarly. The pop up window asks the user for the number of rows and columns in the lawn, the mower starting and ending coordinates, and the locations of up to three boulders. Again, each of these fields are expected to be integers so we use try-catch statements to make sure this is true. If any of the fields are not integers, with one exception, then an error message will be displayed. The exception is with the boulder information. If the user does not wish to put any boulders on the lawn, they only need to input something that is not an integer and the boulder will not appear on the map. All the inputs are checked to make sure they are in bounds. MenuWindow also checks to make sure there are no discrepancies, such as the starting and ending positions being the same, or boulders in starting/ending positions. If all the information checks out, a new LawnMap is created, loaded into a “level 5”, and a new GameWindow of that level is created.



GameWindow is the GUI class that handles the graphics of the game itself along with mower movement and is an isolated class meant for pure GUI, heavily relying on input from MowerGame. Once the user selects free play mode or chooses a level, GameWindow opens up another window, taking a LawnMap as a parameter. GameWindow’s GUI consists of a 2D array of labels that store images of grass, mowers, and boulders. GameWindow has an update function that changes the GUI game grid when the player makes a move, called from the MowerGame whenever it changes its own grid. Because GameWindow is a pure GUI class, all of the update information must be passed to it by the underlying MowerGame class. The update method takes in three parameters, the x and y coordinates of the cell to update and the new character that should be in that position. This information is fed to GameWindow by an instance of MowerGame. Another update method takes in a 2D character array as a parameter and changes the entire GUI to match the given character array. Below the 2D lawn display, there are three labels that depict the current score, the number of moves, and the current game status. After each move, all this information will be updated. Again, MowerGame will feed the information to be updated to the GameWindow. When the game is finished, GameWindow displays the player’s rank, which is calculated using score divided by number of moves. The rank is based on the letter system for grading, with A as the best rank and F as the worst. Similarly to how other letter grades are done, A-F are organized by ratios incremeting by 0.1, like 0.9, 0.8, … Perhaps the most important function of GameWindow is to read in the player’s inputs. GameWindow has a key listener that tracks which keys the user types. The player’s downstrokes on keyboard keys are tracked, and if the downstroke matches to one of the four directional arrow keys, the GameWindow prompts MowerGame to move by one space in the direction pressed. When the current game ends (MowerGame will call GameWindow’s stopGame method), focusable will be set to false so that the the listener will not take more player input to make moves. At this point, the next level button appears, which the user can click to advance to the next level (only for levels 1 to 3).



MowerGame handles the flow of the game. It keeps track of how many patches of grass have been mowed so far, the number of moves made, the score, and detects when the game ends. MowerGame is responsible for moving the mower up, down, left, and right, and updates a grid of the lawn and GameWindow accordingly. MowerGame has a copy of the current state of the lawn stored as a 2D character array that updates every time the mower moves. MowerGame has a move function that takes in two integer inputs, horizontal displacement and vertical displacement (these integers will only ever be -1, 0, 1) and moves the mower in the specified direction. MowerGame first checks to make sure the move is valid and does not go out of bounds or make the mower crash into a boulder. If the move is valid, then MowerGame’s 2D character array will update, score and number of moves will update, and MowerGame will call the respective update functions in GameWindow to change the GUI.

MowerStart is the main class. Its constructor simply opens MenuWindow. MowerStart stores the important data for this project such as a specified LinkedList of LawnMaps designed by us. The LinkedList of LawnMaps is public static final so that all classes can access the LawnMaps but cannot change it. This class is quite simple and is only meant to store the premade LawnMaps and create the MenuWindow to get the program running.

*Detailed Design*

Design for individual classes and methods are attached in the html javadocs of the project.

*Testing*

The JUnitTest extensively tests all aspects of the methods that are possible to test, and shows exactly what is wrong if there is an error. For the code related to the graphics, a rigorous test plan was made that checked all of the different possible positions/outcomes of the game and made sure everything that happened was expected.

Refer to the Test Plan for details on testing.