Software Engineering Group Project
Maintenance Manual AUM Group

Author: Jty, nah37
Config Ref: SE_JC_MM_01
Date: 2018-05-04
Version: 1.0
Status: Release

Department of Computer Science
Aberystwyth University (Mauritius Branch Campus)
Flic-en-Flac
Mauritius

Contents

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to help maintainers improving and debugging the program.

1.2 Scope

This document should be read together with the design specification document and the test specification document.

1.3 Objectives

The objectives of this document are to:

- Give a detailed description of how the program and its different components work
- Suggest any improvements that can be done
- Suggest things that need to be taken into consideration when making changes
- State the physical limitations of the program
- Advise maintainers of what need to be done when rebuilding and testing the program

2. PROGRAM DESCRIPTION

The game gives the player the option to play either a new game of JoggleCube or load a previously saved game. The player also has the option to see the list of the 10 best players that have ever played the game.

Upon loading a new game, the player will be presented 3 grids which have randomly generated letters, 2 buttons, one which clears the user selection and another one which adds the currently selected word to the list of words selected. There is also a list of the already input words that the user has played. A countdown of 3 minutes is also started.

The player can select a letter by typing it on the keyboard or clicking the letter on one of the grids. When a letter is selected, the valid neighbours of that selected tile will be highlighted. The user can then select another letter from one of those highlighted letters and so on. When the user is satisfied with the selection, the player clicks on the button adding the word to the list of selected words. If the word is a valid one, the word will be added to the list and the score will be updated.

When the countdown is over, the player will input their name and can then select to either play a new game, play a previously saved game, see the score board or go to the main menu where the player will be presented with the options mentioned above.

2.1 Program Structure

The program is divided into 2 sub packages:

1. gameFrames

   It contains all the frames needed for the program and the main class.

2. gameLogic

   It contains all the classes that deal with logic of the game. The board is made up of grids which are made of tiles which represent the letter used to make up words. Each letter has a set of properties and they are defined by the enum LetterPopulation. The PositionInGrid has all the methods and properties to set the position of a Tile in a grid and to know the position of a Tile in a grid. The Dictionary class reads the wordlist.txt file and the word to an array. It has a search method that uses binary search to check if a word entered by the player is found in the dictionary. The Player class is used to set and get

the name and score of player. It is useful when sorting the players according to their score to update the scoreboard or a saved game file. That's why the Player class implements the Comparable Interface.

Linear search is used to highlight all the letters that are adjacent to the currently selected letter.

2.2 Algorithms

The significant algorithms in the program are the one used for highlighting the selectable tile in the grid in which the letter a letter has been selected and in the neighbouring grids. They are used by the handleTileAction method in the main class.

```
int row = clickedTile.getPos().getRowNumber();
int col = clickedTile.getPos().getColNumber();
Grid ownerGrid = clickedTile.getOwnerGrid();
// new and improved algorithm to set neighbours to selectable to true
//get the number of rows in a grid
int row_limit = Grid.NO_OF_ROWS_IN_GRID - 1;
// no tiles need to be selected if the number of rows is zero
if (row_limit > 0) {
        //get the number of columns in a grid
        int column_limit = Grid.NO_OF_COLUMNS_IN_GRID - 1;
        for (int x = max(0, row - 1); x <= min(row + 1, row_limit); x++) {
                for (int y = max(0, col - 1); y <= min(col + 1, column_limit); y++) {
                        //do not highlight the selected letter
                        if (x != row || y != col) {
                                setSelectableIfNotAlreadySelected(new PositionInGrid(x, y), ownerGrid);
                          }
                }
        }
}
setSelectableInNeighbourGrids(clickedTile, selectedTiles, ownerGrid);
```

First, it has to get the position of the selected tile in the grid. The rules of the game allow the player to select another tile that is adjacent to it either horizontally, vertically or diagonally.

The nested for loops are to make sure that the constraints of the grid are respected and the indexes are not out of bounds. The row index starts at either 0 or one less than the selected tile's row index while the column index starts at either 0 or one less than the selected tile's column index. The row index also caps out at either the length of a row or one more than the selected tile's row index while the column index caps out at either the length of the column or one more than the selected tile's column index.

It starts highlighting at a row above the selected tile or in the same row and stops when row_limit is reached or the next row. The conditions for the next for loop is the same. It highlights tiles in the same column or the column before or after the selected tile and it stops when the column limit is reached or the next column.

In order to select tiles in the neighbouring grids, the program has to know in which grid the selected tile is found. The selectable grids will depend on the position of the grid in the board and the position of the selected tile in the grid. The algorithm for highlighting the selectable tiles in the neighbouring is the same as the above except that it does not check if the x is not equal to row and y is not equal to column. For each of the tiles in a neighbouring grid it check if it is selectable and if it has not been selected already, it selects it.

```
public void setSelectableInNeighbourGrids(Tile tileInOriginGrid, List<Tile> selectedTiles, Grid
ownerGridOfTile) {
	List<Grid> neighbourGrids = getNeighbourGrids(ownerGridOfTile);
	for (Grid eachGrid: neighbourGrids) {
		eachGrid.setSelectableTiles(tileInOriginGrid, selectedTiles);
	}
}

public List<Grid> getNeighbourGrids(Grid ownerGrid) {
	List<Grid> neighbourGrids = new ArrayList<>();
	  switch (ownerGrid.getGridNo()) {
		case 1:
			neighbourGrids.add(allGrids.get(1));
			break;
		case 2:
			neighbourGrids.add(allGrids.get(0));
			neighbourGrids.add(allGrids.get(2));
			break;
		case 3:
			neighbourGrids.add(allGrids.get(1));
			break;
		default:
			System.err.println("This should never happen");
			break;
	  }
	  return neighbourGrids;
}
public void setSelectableTiles(Tile tileInFirstGrid, List<Tile> selectedTiles) {
	int row = tileInFirstGrid.getPos().getRowNumber();
	int col = tileInFirstGrid.getPos().getColNumber();
```

```
        int row_limit = Grid.NO_OF_ROWS_IN_GRID - 1;
        if (row_limit > 0) {
                int column_limit = Grid.NO_OF_COLUMNS_IN_GRID - 1;
                for (int x = max(0, row - 1); x <= min(row + 1, row_limit); x++) {
                        for (int y = max(0, col - 1); y <= min(col + 1, column_limit); y++) {
                                for (int indexVariable = 0; indexVariable < NO_OF_TILES_IN_GRID;
indexVariable++) {
                                        if (allTiles[indexVariable].getPos().getRowNumber() == x) {
                                                if (allTiles[indexVariable].getPos().getColNumber() == y)
{
                                                        setSelectableIfNotSelected(new
PositionInGrid(x, y), selectedTiles);
                                                }
                                        }
                                }
                        }
                }
        }
}
```

2.3 Main data areas

An ArrayList is used for storing Player object used the ScoreMenu class so that Collections can be used to easily sort them according to their score in descending order. Lists are used in the main class for storing the grids, the generated letters, the correct words, the tiles that belong to a grid and the currently selected tiles and their letters.

2.4 Files

All files that the program needs to access is found in the resources folder. It contains the “highScore.txt” file which stores the name and score of the ten best players, the file “wordlist.txt” is the dictionary file and the folder “savedGames” keeps the files of all the games that the players have decided to save. The “wordlist.txt” is the only that the program cannot update.

3. SUGGESTIONS FOR IMPROVEMENTS

In the SelectGame frame, the user has to scroll down to find a particular saved game, as an improvement, a search option can be added. Thus, as the user would type in the name of the file, it would give he/she the available files to choose from.

We wanted to add sound to give the player a better user experience and of course to mute the sound when he/she wants while playing the game. Animations in the ScoreMenu frame can be added to make the display more attractive and entertaining.

In order to give the letter better look and feel, the buttons shape can be changes to rounded buttons. We have not been able to do that due to the lack of time.

It was suggested by Danshil that an interface could have been used for playSavedGame and playGame to avoid repetition of the same methods that are common to these two classes.

The display of the frame where the board is seen as if it is being viewed from its face can be improved with a 3D rotation of the panels containing the grids. Unfortunately, we did not enough time to implement that.

4. THINGS TO WATCH FOR WHEN MAKING CHANGES

All the files needed by the program are stored in the resources folder, if ever, the maintainers want to put the different files in separate folders, then all the paths must be changed in the program. It has to be noted that the File object is being used to access the files.

If the maintainers want to increase the complexity of the game by increasing or decreasing the number of rows and columns in a grid then the algorithms for loading and saving a saved game must be changed and maybe the files for the saved game must be deleted.

5. PHYSICAL LIMITATIONS OF THE PROGRAM

This program was designed to be used on Windows PCs within the Department of Computer Science of Aberystwyth University. The PCs also need to have a minimum requirement of Java 8 installed as well at least 512 MB of RAM and a Intel Pentium 4 or equivalent.

6. REBUILDING AND TESTING

All identified errors are found in the test report. After having made changes to the program the test to be carried out and the test results are all found in the Test Specification document.

REFERENCES

[1] QA Document SE.QA.10 Producing a Final Report C.J Price

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	*N/A*	*2018-05-04*	*Creation of document*	*jty*
1.1	*N/A*	*2018-05-09*	*Outline, Files*	*nah37*
1.2	*N/A*	*2018-05-11*	*Part 2 to 6*	*Nah37*