FINAL PROJECT DOCUMENTATION

Michelle Du, Kalvin Lam, Joanne Ma, Vinh Truong

BHATNAGARABBLE | May 16th, 2016

TABLE OF CONTENTS

1. Final Program (source code) , Test Programs, Testing Results (test plan)
2. Design Specifications for all classes: Javadocs and hierarchy drawings
3. User manual
4. Post project reflections

INTRODUCTION:

We recreated the game SCRABBLE™ (re-coined at Bharnagarbble due to intellectual property reasons), which is a word game in which two to four players score points by lining up tiles containing a single letter onto a 9x9 grid. The goal of Scrabble is to form English words (similar to a crossword) that flows left to right in rows, or top to bottom in columns.

I. FINAL PROGRAM

SOURCE CODE

TEST PROGRAMS

Class: Board

Method Name	Functions	Possible errors	How we tested
Constructor	Initialize all instance variables	Incorrectly instantiating different variables	Put print statements inside and check only constructor & checkpoints
PlayWord	Plays the word from newBoard	Incorrectly checking the word, not counting points, not copying correctly	Try playing invalid words, play different points, print statements to check the different loop checkpoints
skipTurn	This skips the player's turn	Not many possible errors, just incrementing two	Use the skipTurn method, and it works

		instance variables	
printBoard	Prints out the board	Not many, this was mainly for me to see what the board was like after playing a word	Used the function and it worked. Not much room for error
endGame	Returns true if game should end, false if not	Not much room for error here either, just checking skip count and the letterBag size	Tested the endgame, checked skipping and no letters left and it worked
printWord	Again, for me, checking if the dictionary array was correctly instantiated	Only in the construction of the dictionary array	I used this to test the dictionary array
getCurrentPlayer	Used as helper method to get the current player based on turn count	Not much room for error, maybe just the turn count being wrong	Played a word and then checked the turn count, and it was correct
getPlayer(int x)	Returns the player at the index x	Getter method, used as helper	No room for error; just returns a player
printScores	Used to print the scores and to check my functionality	Not room for error, only error would be with the scores being wrong	Used this to check the other methods
binary	Used this method to search the dictionary using binary	Boundary cases, such as very first and very last word	I used binary on the first and last words in the dictionary and then I tested other random words
isValidWord(String word)	Checks if the word passed through argument is valid using binary	No room for error	Just used the method; it worked
isValidWord(Tile[][] oldBoard, Tile[][] newBoard)	Checks if the new letters in newBoard come together to	Lots of room for error; I set up print statements for when	Tested new words that are both valid and invalid

	make a valid word	I'm debugging to check different checkpoints in my code	
giveLetters	This gives letters to the current player until their hand is full or the letterBag is empty	I had one error where it would only copy 1 letter for their entire hand	I fixed this using different print checkpoints in my code and then using those to check where i was going wrong
wordToString(List<Letter> word)	This is a helper method; since most of the tiles have singular letters, i used this to return a list of letters as a concatenated String	Not much error; simple helper method	Tested out the method, used 1 character string and multi-character string And it worked
isHorizontal(Tile[][] oldBoard, Tile[][] newBoard)	Helper method for many other methods; since the words placed in Bhatnagarabble have to be either horizontal or vertical, this returns true if the new word is horiz; false if not	Precondition is that the word is either horizontal or vertical, so there isn't much room for error;	Tested both horizontal and vertical words and this method worked as planned

GUI: Testing of the GUI was very straightforward since it involved directly interacting with what was displayed on the screen.

Method	Possible Errors	How Errors Were Avoided
onCreate()	Board is not drawn correctly	Board errors were easy to spot, since the board would clearly look wrong. Board errors were also caused by carelessness more than anything else, and mindful programming resolved all board problems.

onCreate()	Player's hand is not displayed correctly	A seed was fed into the random number generator, and the displayed graphics were compared with the printed output of the player's hand to ensure that the two were the same.
onCreate()	Different screen resolutions and sizes cause things to be displayed in different places, sometimes even going out of the screen	Instead of hard coding pixel values, all dimensions of View objects were calculated as values proportional to the screen dimensions.
MyEndClickListener()	Turn and score aren't displayed correctly	Initially believed to be a problem with the API; determined to actually be a problem with the GUI's translator methods that were passing the wrong values to the API.
MyEndClickListener()	Upon ending turn with an invalid move, letter blocks are frozen on board, disallowing the player from fixing his or her mistakes	The playWord() method in the API was modified slightly to return a boolean (false if the move is invalid and true if valid); click listeners of the letters were then turned off only if playWord() returned true.
MyDragListener2()	Blocks can be dragged on top of each other	A simple check was made to make sure that blocks couldn't be drawn into spaces that already contained a block.

II. DESIGN SPECIFICATIONS

SCRABBLE API:

Board Class

The board class represents a standard 15x15 Scrabble board. There are instance variables for our dictionary (stored in an array), our letter bag (stored as an ArrayList of "Letter" objects), our board (a 2-D array of tiles), and integers to record the amount of skips and amount of turns (to determine when the game is over). We also have an array of "Player" objects.

Tile Abstract Class

This abstract class contains the characteristics and behaviors of all tiles. The more specific tiles inherit from this class. In this class, we have two instance variables: Letter placeHold (representing what letter is on the tile) and boolean isEmpty (representing if the tile is filled or not). In this class, we have implemented the constructor with no parameters (to initialize placeHold to an empty value and set isEmpty to true), the isEmpty method (to check if the tile is empty), setLetter (to put a letter on the tile), getLetter (to return the letter on the tile), and getTileScore (to return the score of the Tile).

There are 5 abstract methods: isWordMult, getMultiplier, isMult, getFactor, and toString. Each class that inherits the Tile class will implement these methods differently because they will be different types of Tiles.

MultiplierTile Class

This class represents a specific tile, the multiplier tile. If someone makes a word that includes one of these tiles, the point value of the word will go up (either triple, double, quadruple) by the multiplier value. It inherits the Tile class since a MultiplierTile has an "is-a" relationship with the tile.

GenericTile Class

This class represents a specific tile, the generic tile. This type of tile is a tile with no special multiplier. It inherits the Tile class since a GenericTile has an "is-a" relationship with the Tile.

Letter Class

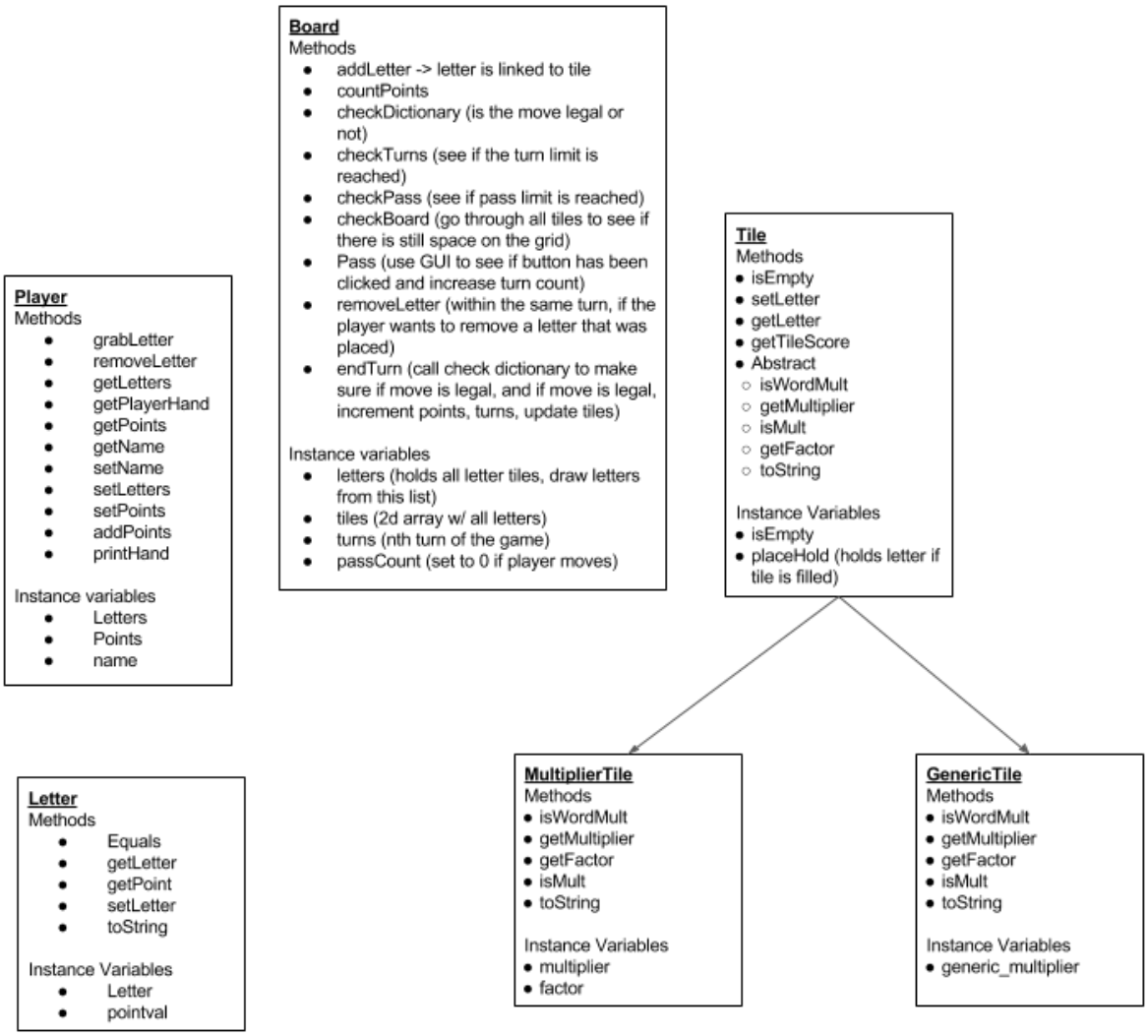
This class represents a specific letter. This Letter can be placed on the tile board.

Player Class

This class represents a player of the game. This player can grab letters and remove letters.

HIERARCHY DRAWINGS:

Method Chart/Diagram



GUI (on Android Studio):

MainActivity Class

This class contains the onCreate() method that runs upon application start. It draws the board and instantiates the Board class that will be the game. It further communicates with the API to initialize the player's letters. Board tiles are LinearLayouts while letter blocks are ImageViews. The parent ViewGroup is RelativeLayout to optimize performance on a variety of screen sizes. Additionally, TextViews to display score and turn count and Buttons to end or pass a turn are drawn.
The MainActivity class also has several global variables, most notably the playerHand View array that stores the letters currently in the player's hand and the blockBoard View array that stores the letters currently on the board. This information is translated into Letter and Tile arrays and then passed to the API to play the game.

Listeners Class

This class contains all of the click and drag listeners that are needed to make the letters and buttons functional:

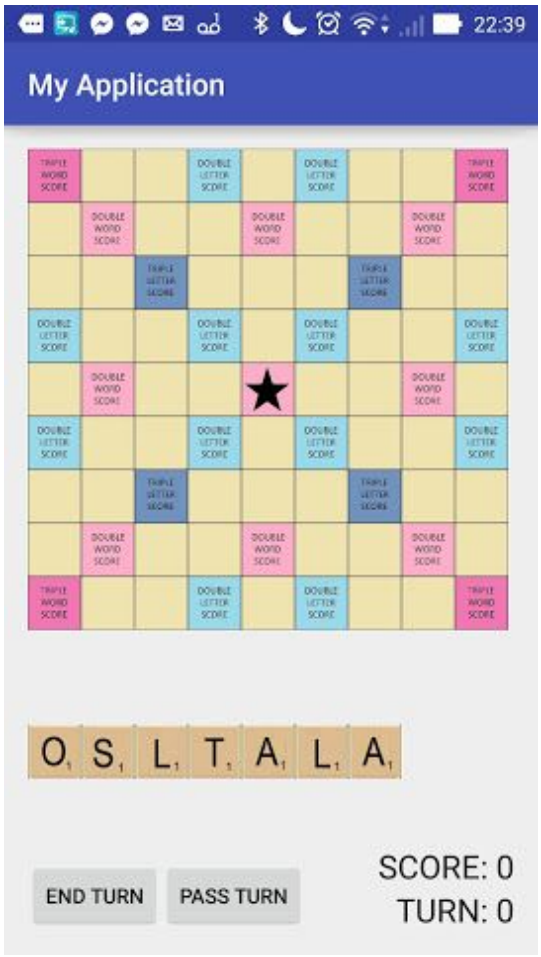
- MyEndClickListener class: Click listener for the "End Turn" button. Most of the game occurs here. Upon click, the listener calls the wordPlay() method from the Board class and changes the TextViews to display the correct score and turn count. It also calls the endGame() method to see if conditions to end the game have been met: if so, a dialog appears that displays the player's final score and allows the player to restart the game. Otherwise, the click listeners of the letters on the board are turned off, and the player's hand is updated to reflect newly drawn letters.
- MyPassClickListener class: Click listener for the "Pass Turn" button. The listener calls the skipTurn() method and updates the turn TextView accordingly. The same endGame() check is also done.
- MyLongClickListener class: Long click listener for the letters. The listener, upon long click input from the user, sets the ImageView to invisible and draws a shadow that the user can then drag around.
- MyDragListener class: Drag listener for the 7 tiles that hold the player's hand. When a letter enters within the range of the LinearLayout, the LinearLayout changes its background color to red. Upon a letter's exit, the background changes back to the default tile image. When a letter is dropped, the ImageView is set back to visible and redrawn over the LinearLayout. This listener only allows

the letter that originally occupied its space to return to that space. Letters dragged onto the board also cannot be dragged back into the player's hand.

- MyDragListener2 class: Drag listener for the tiles on the board. When a letter enters within the range of the LinearLayout, the LinearLayout changes its background color to red. Upon a letter's exit, the background changes back to the according tile image. When a letter is dropped, the ImageView is set back to visible and redrawn over the LinearLayout. This listener does not allow letters to be dropped into spaces that currently contain letters. It is also responsible for updating the playerHand View array and blockBoard View array.

Translator class

This class contains several static methods that facilitate communication between the GUI and API. It converts View arrays from the GUI into Letter or Tile arrays that the API can understand, and it converts Letter arrays from the API into Drawable ids or View ids that help the GUI with identification and finding the correct backgrounds.



III. USER MANUAL

Working with the interface:

The user should note that the letters respond to dragging after a *long click*, i.e. holding your finger down onto the letter block until you feel a vibration. The user should not release his or her finger until the letter is in the range of the tile that the user wishes to drop the letter into (if in range, the tile will be red.)

Once the user is satisfied with his or her actions, the user should press the “End Turn” button to end the turn. If the user’s move is not valid, nothing will happen (the score and turn counter will not increment.) If the user’s move is valid, the user should see the score and turn counter change accordingly. See the game rules for what makes a move valid or not valid and the way that points are scored.

GAME RULES:

Game specifics:

- Premium squares, which are indicated by different colored tiles, multiply the number of points awarded. There are *four* dark pink triple-word squares, *nine* double-word squares, where the center square is marked with a star, *four* dark blue triple-letter squares, and *twelve* light blue double-letter squares.
- There are 100 letter tiles, 98 of which are marked with a letter and a point value ranging from 1-10, which is based on how frequently a letter is used in English writing (ie. A is worth 1 point whereas Q is worth 10 points).

Making Plays:

- The first played word must be at least two letters long and cover the center square H8.
- Moves made after the first play involve using one or more letter tiles to make a word on the board.
- As long as all the letters placed on the board in one play lie in one row or column and are connected by a main word, any run of tiles on two or more consecutive squares along a row on column are legal combos in a play.

Scoring:

- Tile value: the point value of each tile is indicated in the corner of the tile. Blank tiles are worth zero points.
- Each new word formed in a play is scored separately, and then the scores are added up.
- If hook words are played, the scores for each word are added separately.
- The word containing every played letter is denoted as the “main word”. The point values of each letter tile is added up, and the tiles that are placed on the double-word squares are double while the tiles placed on the triple-word squares are tripled in value. However, premium squares only apply when newly placed tiles cover them. Any subsequent plays do not count premium squares.
- If a play is made in which the main word covers two double-word squares, the value of the word is doubled, then redoubled. If the main word covers two two-word squares, the value of that word is tripled, then tripled.
- The game ends if the player passes six times consecutively, or if there are no more letters to be drawn

IV. POST PROJECT REFLECTIONS

Looking back, it would have made more sense to do a simpler project given the time constraints and our limited to no knowledge of Java, GUI, and Java Swing.

Ways we could have improved the project workflow was to schedule and make room for Google Hangout meetings every 2-3 days to check up on the code and learning progress. This would help negate a lot of time we spent trying to contact each other when problems arose. Unfortunately, a large portion of the project working period occurred over the course of AP testing, so not everyone was available to work on the code daily.

It would have also been beneficial if all members were well versed in GUI -- we would often run into problems where we would write code, but the GUI would not be compatible with the code. As Michelle puts it, the GUI is “incredibly stupid and limited”, and we spent a large majority of our time working on the code so that it would be compatible with the GUI.

Developing a GUI proved to be a challenging experience. We began with almost zero experience in Android Studio and had to learn the basics of GUI development before we could even start work on the project. However, under the pressure of time constraints, developing a GUI from scratch also proved to be a great learning experience, and we are now much more familiar with Android Studio.

Although we had to give up weekends and time that we needed to spend studying for finals and working on other final projects, it was a rewarding and worthwhile experience.