In this last and final version of WordNerd, you will add two new functionalities: track the player's scores and search for the words played so far. A brief overview of the functionality is described below. for more details, see the video clippings provided on Canvas.

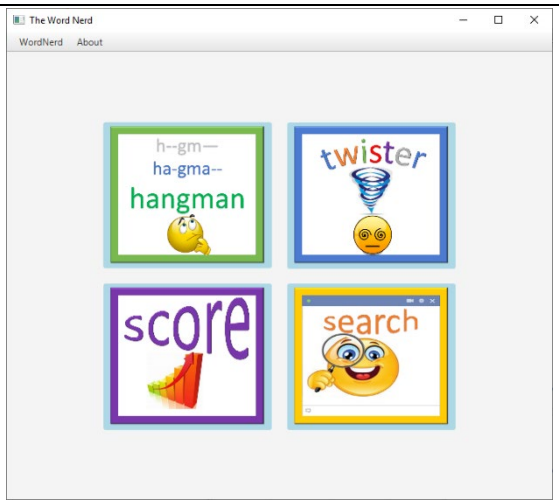


Figure 1: Opening scene

The **ScoreBoard** (Figure 2) shows line charts for scores earned in the two games. The scores are computed as soon as a round is completed (won or lost) and stored in scores.csv file (Figure 3). The scores are computed as:

- Hangman score = number of hits / number of misses
- Twister score = total number of correct words submitted / total number of solution words

Figure 2: ScoreBoard View (The linecharts dynamically adjust their x and y-axis scales as more data is read from scores.csv)

```
0,retirement,11,4.000000
0,lakewood,23,0.111111
0,outkast,21,0.750000
0,valencia,27,0.500000
1,scots,120,0.550000
1,morocco,120,0.529412
1,tracker,120,0.219178
```

Figure 3: scores.csv

Each row in the csv file has score for one round of any of the two games. Each row has fields: gameid, puzzleword, time in seconds, score. gameid 0 corresponds to HANGMAN_GAME_ID and 1 corresponds to TWISTER_GAME_ID

The **Search** functionality shows a table view of the scores data. (Fig.4). The gameComboBox allows the data in tableView to be filtered by the game. By default, it shows data from all games. The searchTextField can be used to further narrow down the puzzle words by typing in the letters that the words must have.

The Exit button in both the views (Score and Search) take control back to the opening scene (Fig.1).

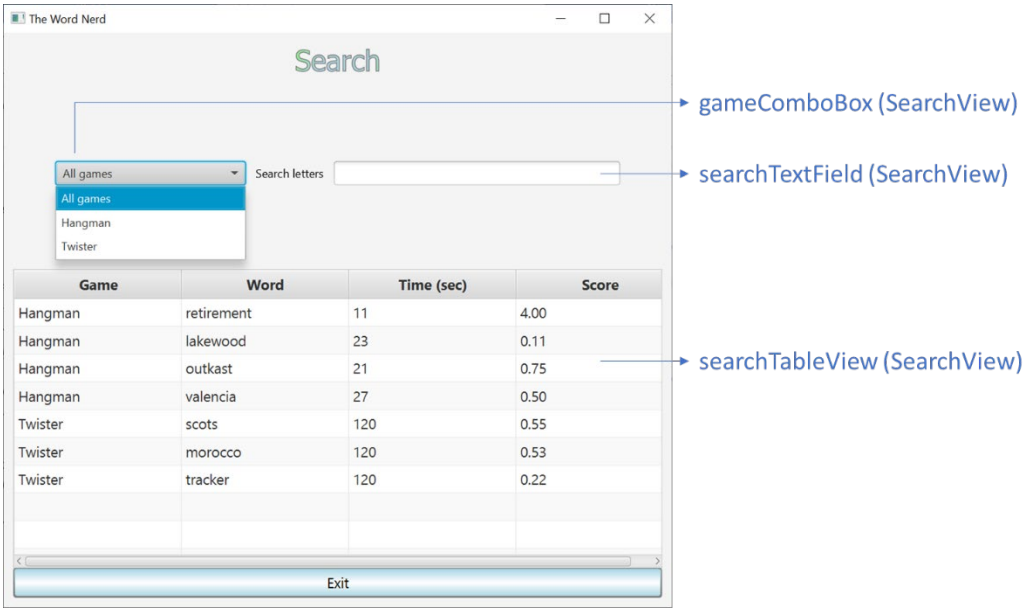
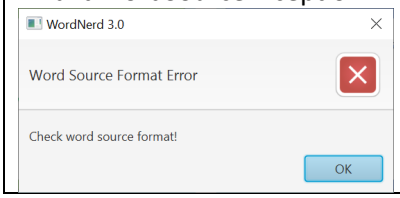


Figure 4: SearchView

To develop these new functionalities, some key design changes are made. WordNerdModel has read and write methods for scores.csv, and there are several new classes as listed below

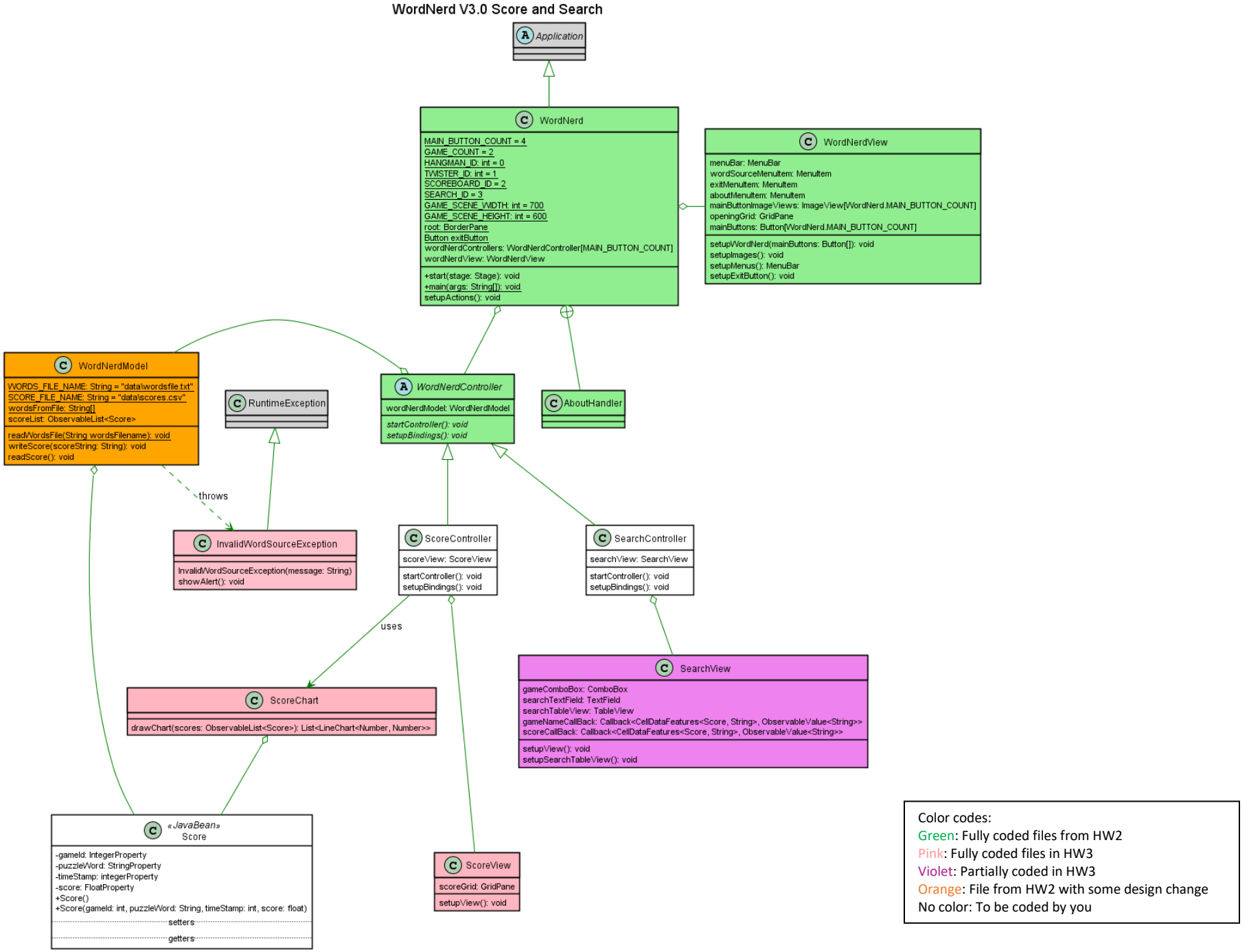
New classes	Description
Score	Java bean to keep score. Its four member variables represent the four values stored in scores.csv, i.e. gameId, puzzleWord, timestamp, and score.
ScoreView	**Fully coded**
ScoreChart	**Fully coded**. It has one method drawChart() that takes one observable list of Scores and returns a list of two LineCharts - one for hangman and one for twister. These two charts need to be plugged into ScoreView in one of the methods of ScoreContoller
ScoreController	Extends WordNerdController and Implements its methods as needed
SearchView	**Partially coded**. You need to setup the searchTableView and write Callbacks as needed. I wrote two callbacks - gameNameCallback - to get Hangman or Twister for 0 and 1, and - scoreCallback - to format the score value to display as shown in Fig. 4.
Search Controller	Extends WordNerdController and Implements its methods as needed
InvalidWordSourceException 	**Fully coded**. This exception is thrown when either the word source file is empty or has some words with non-alphabet characters (e.g. 5, $, or *). When the user tries to open a file with any of these two problems, WordNerd is supposed to display an Alert (see image on the left) and go back to the word source that it was previously attached to. The exception GUI is already coded for you in the class. You have to figure out how to use it appropriately.

In this version, you can make changes to any of the Java files, including the ones fully coded by me. And that means you must submit ALL java files except the test-case files because I won't know what you changed. You have to implement all the functionality, and you are free to changes any class. But make sure that all test-cases from HW2 continue to pass!

Instructions:

1. **Folder structure**:
   a. The package structure will be same as in HW2, except that it will be in a new package named hw3.
   b. Download java files and store them in hw3 package. Copy/import other Java, data, and image files from hw2 into this package.
   c. Complete the code as needed. Write your name and Andrew ID in ALL java files except test-files.
2. **Rubric**: You will test your program in two ways: Console output and test-cases. These will get you 80% of the points. Other criteria applied to evaluate your program are:
   a. Documentation (5%): Your code should be well-commented, i.e. neither too many comments, nor too few. Name your variables in a self-explanatory way. Write your name and Andrew id at the top in the comments in each class. Indent your code properly.
   b. Code quality (5%): coding conventions, no unused variables/libraries, etc. Use your judgment to assess these criteria.
   c. Code robustness (5%): Your program should not throw any errors while processing. You can safely assume that the user will not enter any garbage input.
   d. Submission instructions (5%): Zip ALL java except test files into AndrewId-hw3.zip. Do not submit any other folders, class files, text files, and rest of your kitchen sink! Only last submission will be graded.

NO LATE SUBMISSIONS PLEASE! If you are unable to submit on time, you lose all the points. Please avoid last minute submission as Canvas may decide to quit on you! Learn to trust technology only to the extent you should! Do not take that risk! **I will not accept late submission. Good luck!!**


WordNerd V3.0 Score and Search
Application
WordNerd
MAIN_BUTTON_COUNT = 4
GAME_COUNT = 2
HANGMAN_ID: int = 0
TWISTER_ID: int = 1
SCOREBOARD_ID = 2
SEARCH_ID = 3
GAME_SCENE_WIDTH: int = 700
GAME_SCENE_HEIGHT: int = 600
root: BorderPane
Button exitButton
wordNerdControllers: WordNerdController[MAIN_BUTTON_COUNT]
wordNerdView: WordNerdView
+start(stage: Stage): void
+main(args: String[]): void
setupActions(): void
WordNerdView
menuBar: MenuBar
wordSourceMenuItem: MenuItem
exitMenuItem: MenuItem
aboutMenuItem: MenuItem
mainButtonImageViews: ImageView[WordNerd.MAIN_BUTTON_COUNT]
openingGrid: GridPane
mainButtons: Button[WordNerd.MAIN_BUTTON_COUNT]
setupWordNerd(mainButtons: Button[]): void
setupImages(): void
setupMenus(): MenuBar
setupExitButton(): void
WordNerdModel
WORDS_FILE_NAME: String = "data\wordsfile.txt"
SCORE_FILE_NAME: String = "data\scores.csv"
wordsFromFile: String[]
scoreList: ObservableList<Score>
readWordsFile(String wordsFilename): void
writeScore(scoreString: String): void
readScore(): void
RuntimeException
WordNerdController
wordNerdModel: WordNerdModel
startController(): void
setupBindings(): void
AboutHandler
throws
InvalidWordSourceException
InvalidWordSourceException(message: String)
showAlert(): void
ScoreController
scoreView: ScoreView
startController(): void
setupBindings(): void
SearchController
searchView: SearchView
startController(): void
setupBindings(): void
uses
ScoreChart
drawChart(scores: ObservableList<Score>): List<LineChart<Number, Number>>
SearchView
gameComboBox: ComboBox
searchTextField: TextField
searchTableView: TableView
gameNameCallBack: Callback<CellDataFeatures<Score, String>, ObservableValue<String>>
scoreCallBack: Callback<CellDataFeatures<Score, String>, ObservableValue<String>>
setupView(): void
setupSearchTableView(): void
«JavaBean»
Score
-gameId: IntegerProperty
-puzzleWord: StringProperty
-timeStamp: integerProperty
-score: FloatProperty
+Score()
+Score(gameId: int, puzzleWord: String, timeStamp: int, score: float)
setters
getters
ScoreView
scoreGrid: GridPane
setupView(): void
Color codes:
Green: Fully coded files from HW2
Pink: Fully coded files in HW3
Violet: Partially coded in HW3
Orange: File from HW2 with some design change
No color: To be coded by you