Problem statement: In HW2 you will deliver WordNerd 2.0 that has a GUI and two games - Hangman and Twister. Fig1. shows the opening GUI with two buttons for two games. The functionality for other two buttons - Score and Search - will be implemented in HW3.

Solution design: The screenshots in Fig.1, 3, and 4 show various GUI components fully designed for you. You need to attach handlers and underlying logic using the concepts of inheritance, Arrays and Lists, Java FX GUI event handlers, JavaFX properties, beans, and bindings (special videos provided to learn this). Refer to next page for more details about the functionality and design of the games.

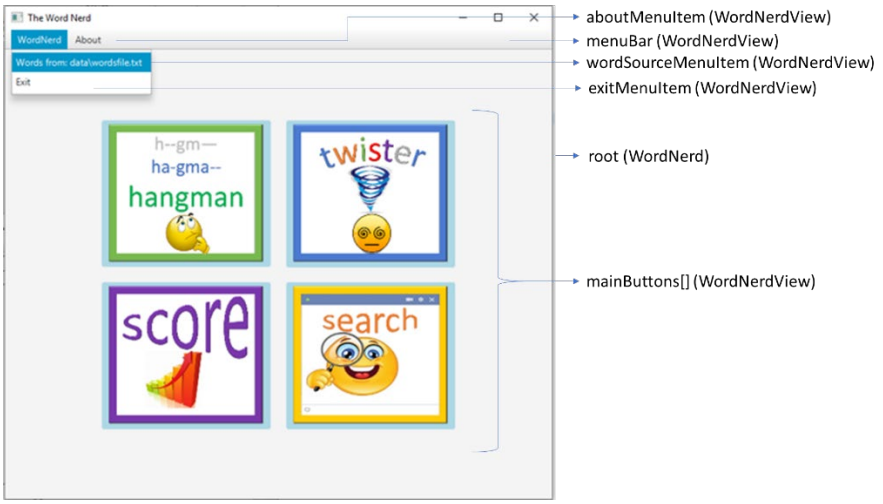


Figure 1: Opening scene

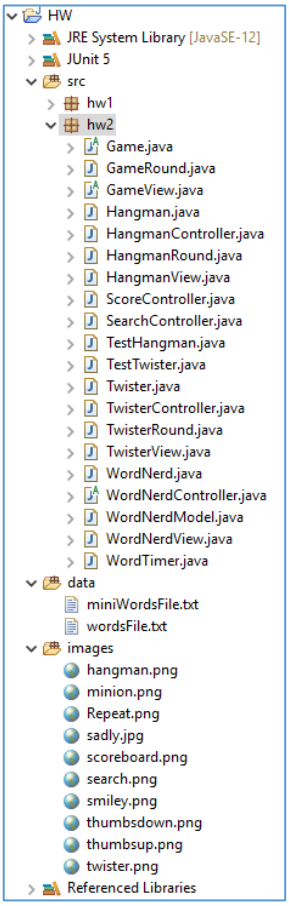


Figure 2: Folder structure

Instructions:

1. **Folder structure**:
   a. Create a package named hw2, and two source folders: data and images in HW project. Note the special icon for source folders.
   b. Download java, data, and image files from Canvas and store them in appropriate folders as shown in Figure 2.
   c. Create the missing class files and complete the code as needed. Write your name and Andrew ID in ALL your java 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 your java files except the given fully-coded files and test-files into AndrewId-hw2.zip. Do not submit any other folders, class files, text files, and rest of your kitchen sink! Max two submissions allowed. 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!!**

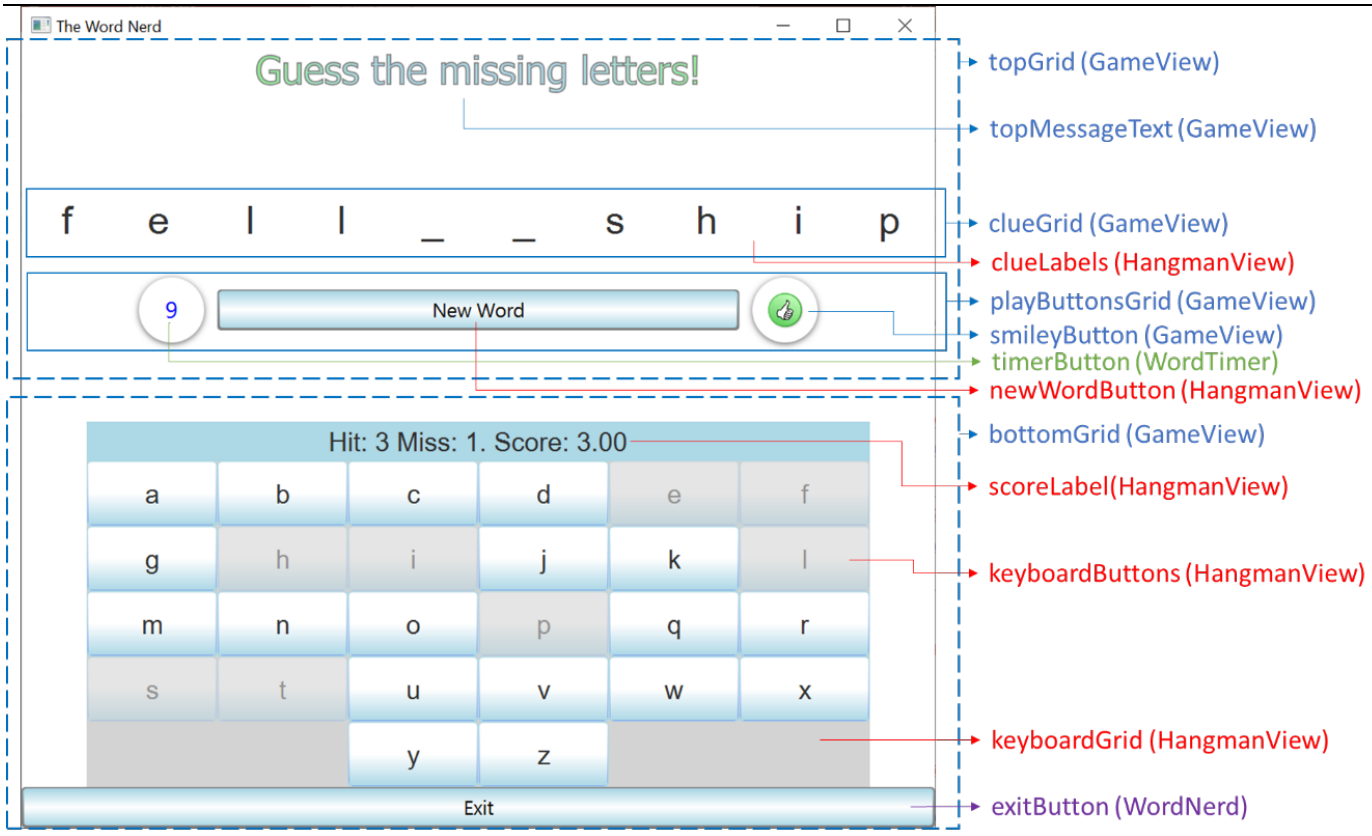


Figure 3: Hangman View (names in parentheses indicate the class names where these components are declared)

Hangman rules:

1. The game displays a clue with number of alphabets between MIN_WORD_LENGTH and MAX_WORD_LENGTH in clueLabels. The player has to guess the letters for missing alphabets in HANGMAN_TRIALS. (Note that the clue has 'underscores' whereas HW1 had 'hyphens.')
2. The player starts Hangman by clicking at Hangman button in opening scene (Fig.1) which starts a new round.
3. Hangman view displays a new clue on top. The wordTimer starts from HANGMAN_GAME_TIME counting down to 0.
4. The player can make guesses using keyboardButtons. To start with, keyboardButtons has those buttons disabled that have letters that are part of the clue. As the player clicks on more buttons, they get disabled to indicate that they have been clicked, irrespective of whether they are correct guesses or not.
5. Every right guess shows the image at THUMBS_UP_INDEX in smileyButton and every incorrect guess shows the image at THUMB_DOWN_INDEX. If time runs out, it shows image at SADLY_INDEX, and if user is able to complete the word, it shows the image at SMILEY_INDEX.
6. If the player clicks on New Word button, the game starts a new round.
7. If the timer runs out before the player could guess the word, the entire keyboardGrid gets disabled to indicate that the round is over.
8. If the player clicks on Exit button, the game goes back to opening scene (Fig. 1).

Please watch the video clippings on Canvas to understand more of the expected functionality.

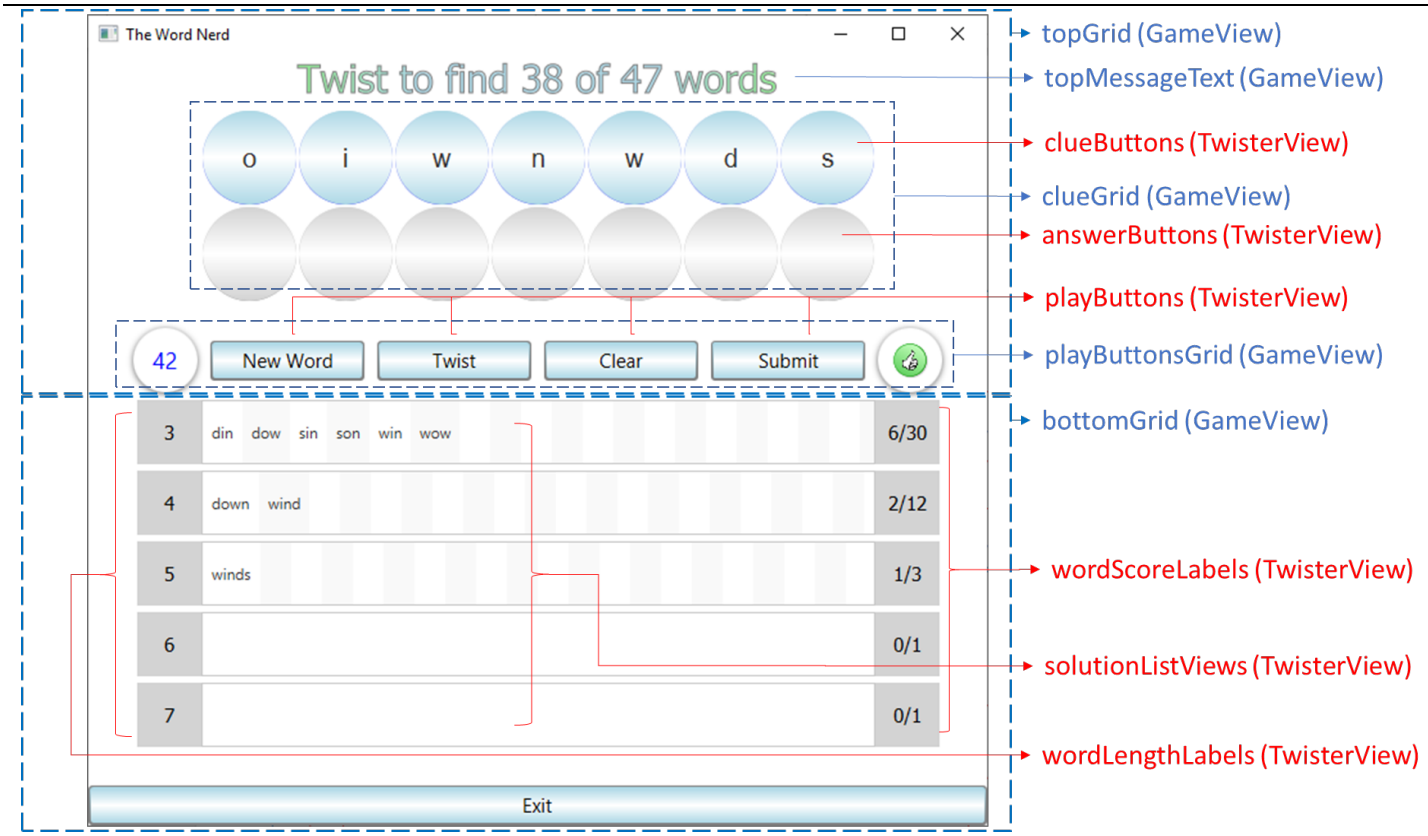


Figure 4: Twister View

Twister rules:

1. The challenge in Twister game is to wring out a given number of words from a clue. This number is indicated in topMessageText.
2. The words must be between TWISTER_MIN_WORD_LENGTH and the clue length. So if the clue is 'abcde' then the length of solution words must be between the value of TWISTER_MIN_WORD_LENGTH, which is 3, and 5.
3. The wordLengthLabels indicate the length of words to be found, and wordScoreLabels show the number of possible words of each length based on the words in the source WORDS_FILE_NAME. For example, the example above shows that there are 30 words of length 3, out of which six words have been found by the player. The words found by the player are displayed in solutionListViews in sorted order, as shown.
4. The round starts with a clue displayed in clueButtons and a topMessageText displaying the number of words to be found. It also starts the wordTimer from TWISTER_GAME_TIME, counting down to 0.
5. The letters in clueButtons are to be moved to answerButtons to form a word. The player can click any of the clueButtons. This moves the alphabet from the clicked clueButton to the first free answerButton from the left. The player can keep moving alphabets from clueButtons to answerButtons to form a word.
6. If the player clicks on any of the answerButtons that has an alphabet, then the alphabet moves back up to the first free space available in clueButtons from the left
7. If the player clicks on Twist button, then the game shuffles the letters in clueButtons. This is to help the player look at the letters in different ways.
8. If the player clicks on Clear button, then the game moves the alphabets in answerButtons, if any, to empty slots in clueButtons.
9. When the player clicks on Submit button,

a. the game takes the word from answerButtons and checks if it is of TWISTER_MIN_WORD_LENGTH. If not, the smileyButton displays the image at THUMBS_DOWN_INDEX.
b. If the submitted word's length is at least TWISTER_MIN_WORD_LENGTH, it checks if it is one of the solutionWords (loaded from words file) that has not been entered before. If yes, then the smileyButton displays the image at THUMBS_UP_INDEX. The game adds the word to one of the solutionlistViews that has words of submitted word length. The submitted word is displayed in the position of its sorted order. The game also updates the wordScoreLabel on the right of the listView as well as the scoreString at the top in topMessageText.
c. If the submittedWord is correct but was entered before, then smileyButton displays image at REPEAT_INDEX
d. If it is not a correct word (i.e. not in words file), then the smileyButton displays image at THUMBS_DOWN_INDEX
e. If the player completes all words in solutionWordList, then smileyButton displays image at SMILEY_INDEX.

10. If the timer runs out, then the smileyButton displays image at SADLY_INDEX
11. If the player clicks on New word, the game starts a new round
12. If the player clicks on Exit button, the game goes back to opening screen.

For more details on game scenarios, please watch the video clippings on Canvas.

Class diagram: The following class diagrams show the classes used in WordNerd 2.0. For the diagram to fit into one page, Fig. 5 shows classes for Hangman game only. The classes for Twister game are shown in Figure 6. The green classes are fully coded and yellow are partially coded. A brief description of these classes, variables, and methods is given in Table 1.

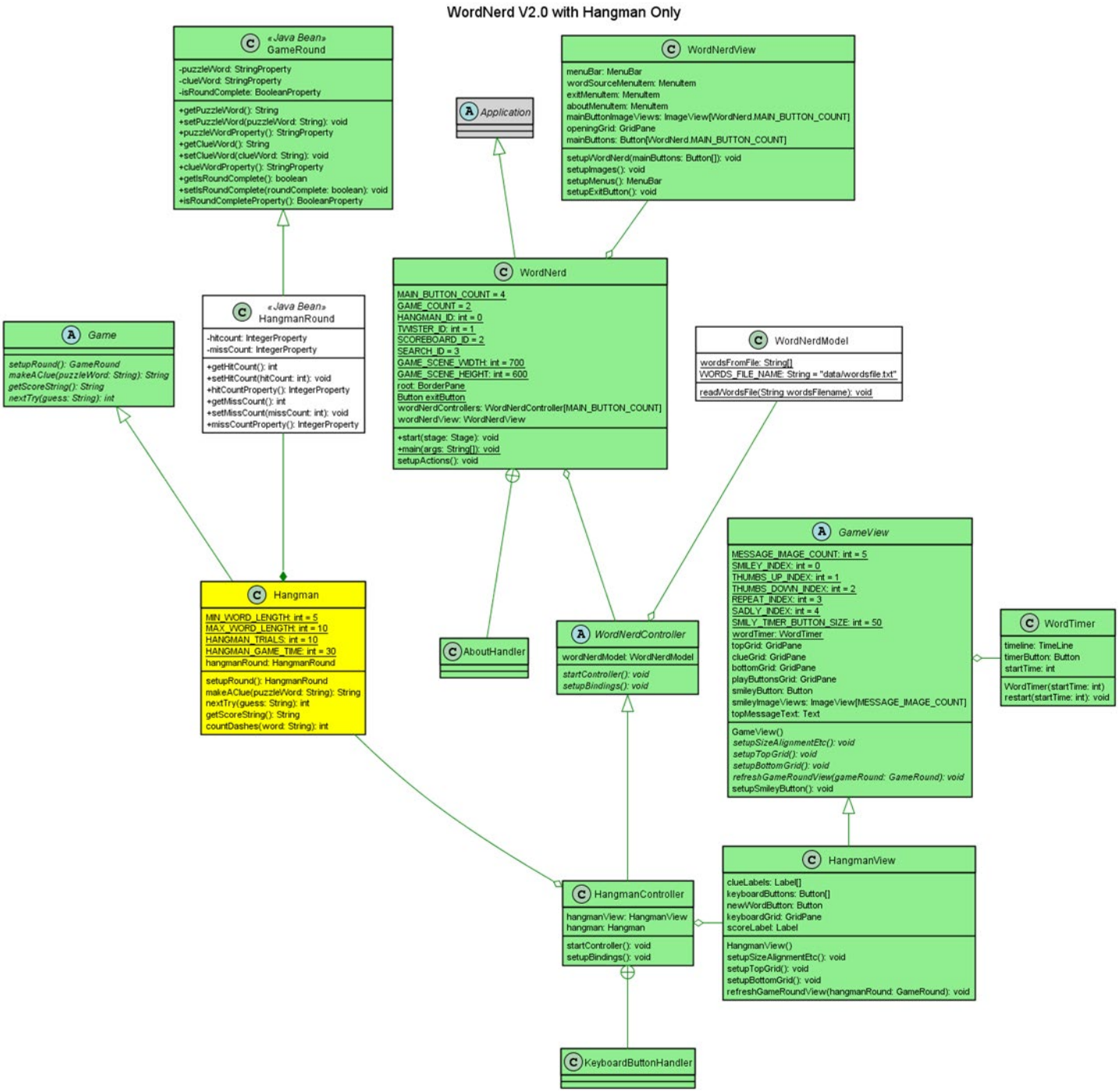


Figure 5: Class diagram for WordNerd with Hangman

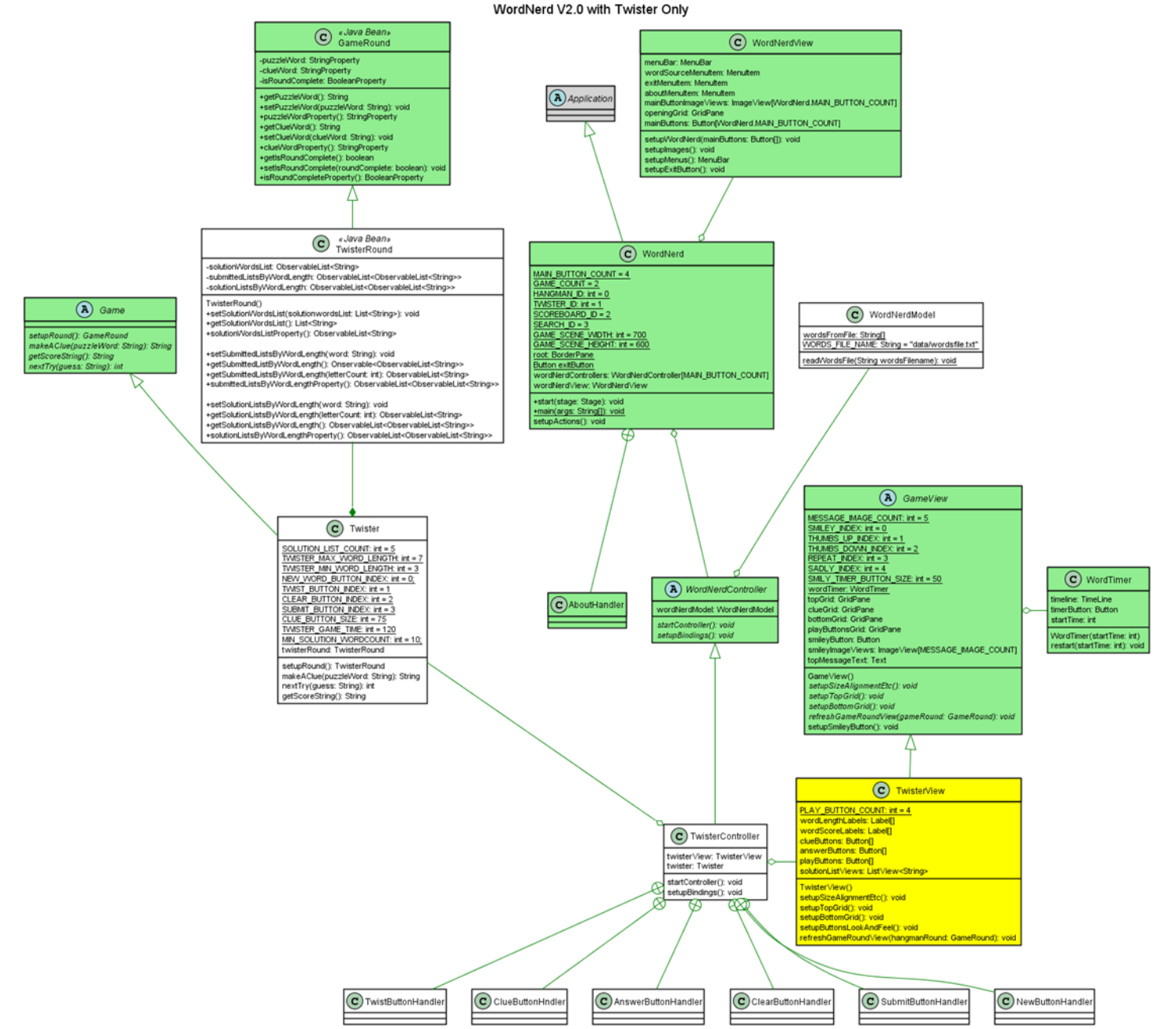
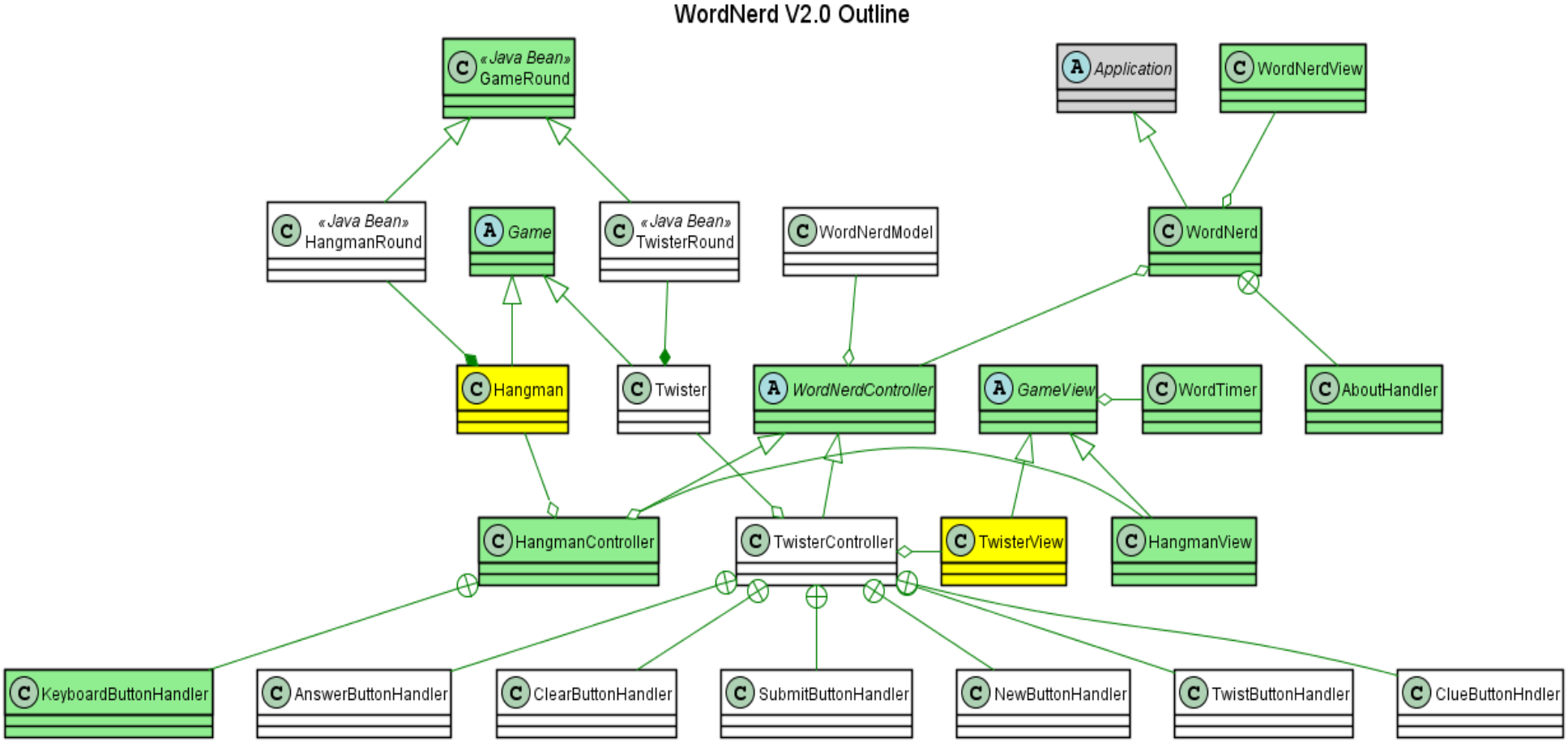


Figure 6: Class diagram for WordNerd with Twister.

Table 1: Class descriptions

#	Class	Description
1	**GameRound**	Same as in HW1 but now it is defined as a Java bean.
2	**HangmanRound**	Same as in HW1 but now it is defined as a Java bean
3	**Game**	Almost same as HW1 except that file handling operations are moved to WordNerdModel class which is new in HW2. This is to follow the MVC design pattern.
4	**WordNerdModel**	Will handle file handling for reading words-source files, and reading/writing score file in HW3. Note that WORDS_FILE_NAME has value : "data\\wordsfile.txt" as data files will now be in data source folder.
4	**Hangman**	Most of the functionality in this class remains same as in HW1 except for the changes required due to GUI. Please refer to the code comments for specifics.
5	**GameView**	Fully coded class with some abstract methods to be implemented by Hangman and Twister
6	**HangmanView**	Fully coded class that extends GameView and lays out Hangman-specific GUI elements as shown in Figure 2.
7	**WordNerdController**	Fully coded abstract class to be implemented by four controllers for four main buttons in the opening scene (Figure 1)
8	**HangmanController**	Fully coded class that extends GameController and implements its methods.
9	**KeyBoardHandler** (Inner class in HangmanController)	Fully coded Event handler in Hangman to handle each button-click on keyboardButtons. Refer to the code comments for more details.
10	**WordNerd**	Gets the game started
11	**WordNerdView**	Lays out the opening scene's GUI items
12	**GameView**	Fully coded abstract class that lays out the GUI elements common to Hangman and Twister.
13	**HangmanView**	Fully coded class that extends GameView and lays out Hangman-specific GUI elements as shown in Fig.3.
14	**WordTimer**	Fully coded class that has a timerButton with a timer that can be started and restarted to run for a given 'startTime'. Used both by Hangman and Twister for each round of play.
15	**TwisterView**	Partially coded class that extends GameView and lays out Twister-specific GUI elements as shown in Figure 4. There is only one method - refreshGameRoundView() - to be coded by you. Refer to HangmanView class for some hints. However, this one will be little more complex because you need to add new clueButtons, answerButtons, and solutionListViews for words of different lengths depending on the puzzleWord's solutions.
16	**Twister**	Extends Game class. Some specifics: • setupRound() returns a new puzzleWord, randomly drawn from wordsFromFile whose length is between TWISTER_MIN_WORD_LENGTH and TWISTER_MAX_WORD_LENGTH. It also must have at least MIN_SOLUTION_WORDCOUNT words twisted out of it. Once the puzzleWord is found, this method creates a new instance of TwisterRound, initializes its properties based on the puzzleWord found, and returns the instance • makeAClue() takes the puzzleword as currently displayed in cluebuttons and shuffles its letters to make a clue • getScoreString() returns the scoreString to be displayed in the topMessageText • nextTry()takes the guess String passed to it, checks the twisterRound's various lists to determine which of the smiley images should be displayed in

		smileyButton as per the game rules, and returns the index of that image. Note that the guess must be at least of TWISTER_MIN_WORD_LENGTH
17	**TwisterRound**	A Java bean extending GameRound. Its properties and methods are explained below:
	solutionWordsLists: Has all possible words that can be twisted out of the puzzle word. Getters and setters for solutionWordLists • **setSolutionWordsList**(List<String> solutionWordsList): setter for solutionWordsList • **getSolutionWordsList**(): getter for solutionWordsList • **solutionWordsListProperty**(): getter for solutionWordsList property	
	solutionListsByWordLength: a list of lists. Each list in it carries words of a certain size that can be twisted out of the puzzleWord. The number of lists in it equals TWISTER_MAX_WORD_LENGTH - TWSITER_MIN_WORD_LENGTH + 1. For example, if the two constants mentioned above are 7 and 3 respectively, then solutionWordLists has (7-3+1) = 5 lists, for words of length 3, 4, 5, 6, and 7. The setupRound() in Twister populates these lists with words. Getters and setters for solutionListsByWordLength: • **setSolutionListsByWordLength**(String word): adds the word to the list that contains words of its length • **getSolutionListsByWordLength**(): returns the list • **getSolutionListsByWordLength**(int length): overloaded method that returns the list that contains words of given length • **solutionListsByWordLengthProperty**(): returns the property	
	submittedListsByWordLength: a list of lists similar to solutionListsByWordLength described above, except that it contains words submitted by the player. • **setSubmittedListsByWordLength**(String word): adds the word to the list that contains words its length • **getSubmittedListsByWordLength**(): returns the list of lists • **getSubmittedListsByWordLength**(int length): overloaded method that returns the list with words of given length • **submittedListsByWordLengthProperty**(): returns the property	
18	**TwisterController**	Extends GameController class.
19	**TwistButtonHandler** (inner class in TwisterController)	EventHandler attached to Twist button. Twists the word currently displayed in clueButtons
20	**ClearButtonHandler** (inner class in TwisterController)	EventHandler attached to Clear play button. Clears the letters in answerButtons, if any, and moves them to empty slots in clueButtons
21	**SubmitButtonHandler** (inner class in TwisterController)	EventHandler attached to Submit button. Refer to Twister rules to for what this handler is supposed to do.
22	**ClueButtonHandler** (inner class in TwisterController)	EventHandler attached to all clueButtons. Takes the alphabet in clicked clueButton, finds the next empty slot starting from left in answerButtons, and puts the alphabet in it. Clears the clicked clueButton
23	**AnswerButtonHandler** (inner class in TwisterController)	EventHandler attached to all answerButtons. Takes the alphabet in clicked answerButton, finds the next empty slot starting from left in clueButtons, and puts the alphabet in it. Clears the clicked answerButton
24	**NewButtonHandler**	EventHandler attached to New button. Starts a new round of game.


WordNerd V2.0 Outline
«Java Bean»
GameRound
Application
WordNerdView
«Java Bean»
HangmanRound
Game
«Java Bean»
TwisterRound
WordNerdModel
WordNerd
Hangman
Twister
WordNerdController
GameView
WordTimer
AboutHandler
HangmanController
TwisterController
TwisterView
HangmanView
KeyboardButtonHandler
AnswerButtonHandler
ClearButtonHandler
SubmitButtonHandler
NewButtonHandler
TwistButtonHandler
ClueButtonHndler