BZ 214 Visual Programming Project Report

Group Number:02	
No:1030510264	**Name Surname:Emre ÖZGENÇ**
Software design, MVC pattern structure, Filtering data (WordFilter), Bug fixing, Git and Github configuration, Code optimization, Design, style files (css), and Directing team members	
No:1030510178	**Name Surname:Nurullah ATAŞ**
Reading data from the dictionary (TxtReader), Exception handling in controllers, Bug fixing, Adapting classes to incoming changes, Constant classes (UINames, Messages etc.), UML design	
No:1030510209	**Name Surname:Mehmet KEKEÇ**
Game creation and management systems (GameManager, GameCreatorManager), Main menu improvements, Testing game, Bug fixing, Adapting classes to incoming changes, Use-case and UML design	

Abstract

Developing a desktop application clone of New York Times Spelling Bee via JavaFX by implementing Model-View-Controller design pattern

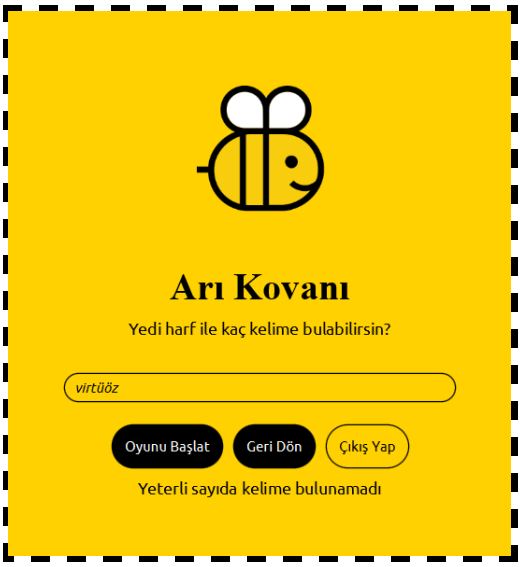
Software Design

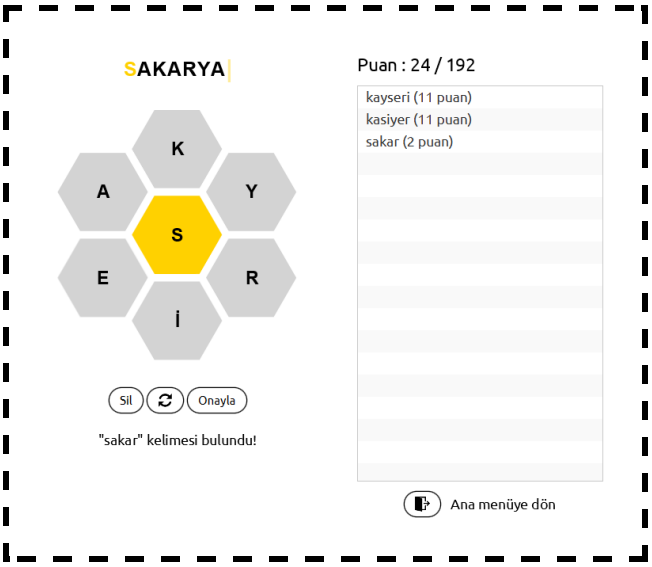
Technologies and Tools

- Git and Github (Version control / Working together)
- JavaFX - Java SE (not including FXML technology)
- IntelliJ IDEA
- Maven (Dependency management)

Design and User Interface

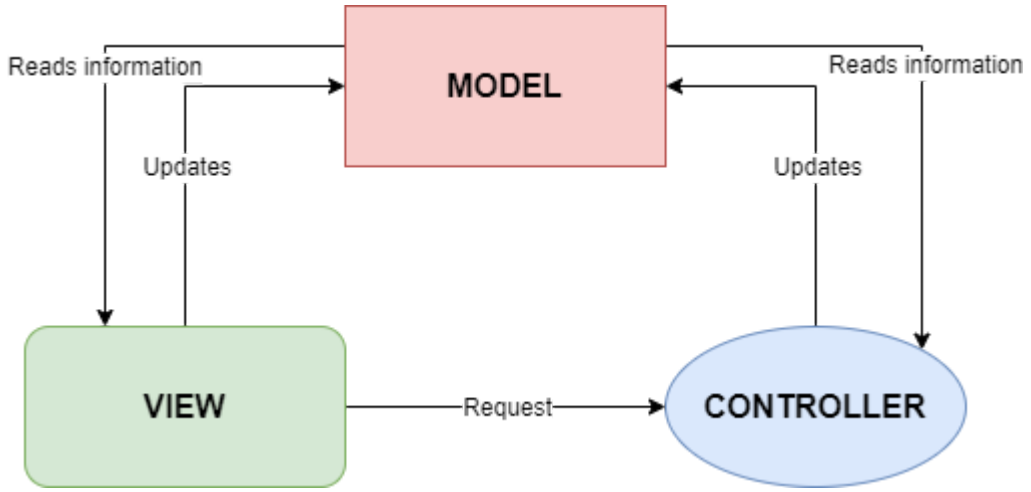
For the application design, we stayed loyal to the base game. We have built similar user interfaces, components and shapes. Also used the same color palette.





MVC Pattern

We did not prefer to use FXML and Scene Builder so we had to implement MVC manually. We found an answer from a thread on Stackoverflow.com. And used the answer as a reference to implement the pattern. (Thread link)



Application Workflow

Reading data -> Filtering data -> Game creation -> Game management (user actions)

Reading Data

Reading given dictionary process as a regular text file. The Reading phase also includes minor filters. (words whose length is less than 4 characters, "â" - "a" Turkish character problem) This work is done by the **TxtReader** class.

Filtering Data

Filtering the words by given instructions, finding appropriate pangrams, words, and letters that are used in the puzzle. This work is done by **WordFilter** class.

Game Creation

Creating the puzzle by using the data which comes after filtering. The Creation process includes finding the appropriate point range and word count. Also guarantees that the puzzle contains at least one pangram word. This work is done by **GameCreatorManager** class.

Game Management

Controlling the words that the user typed, the user's current point and storing the words that were found by the user. This work is done by **GameManager** class.

UML and use case diagrams can be found below

Conclusion

The application does everything described well. It reads the dictionary, filters the words, creates the game, and draws the game scene in under 500 milliseconds.

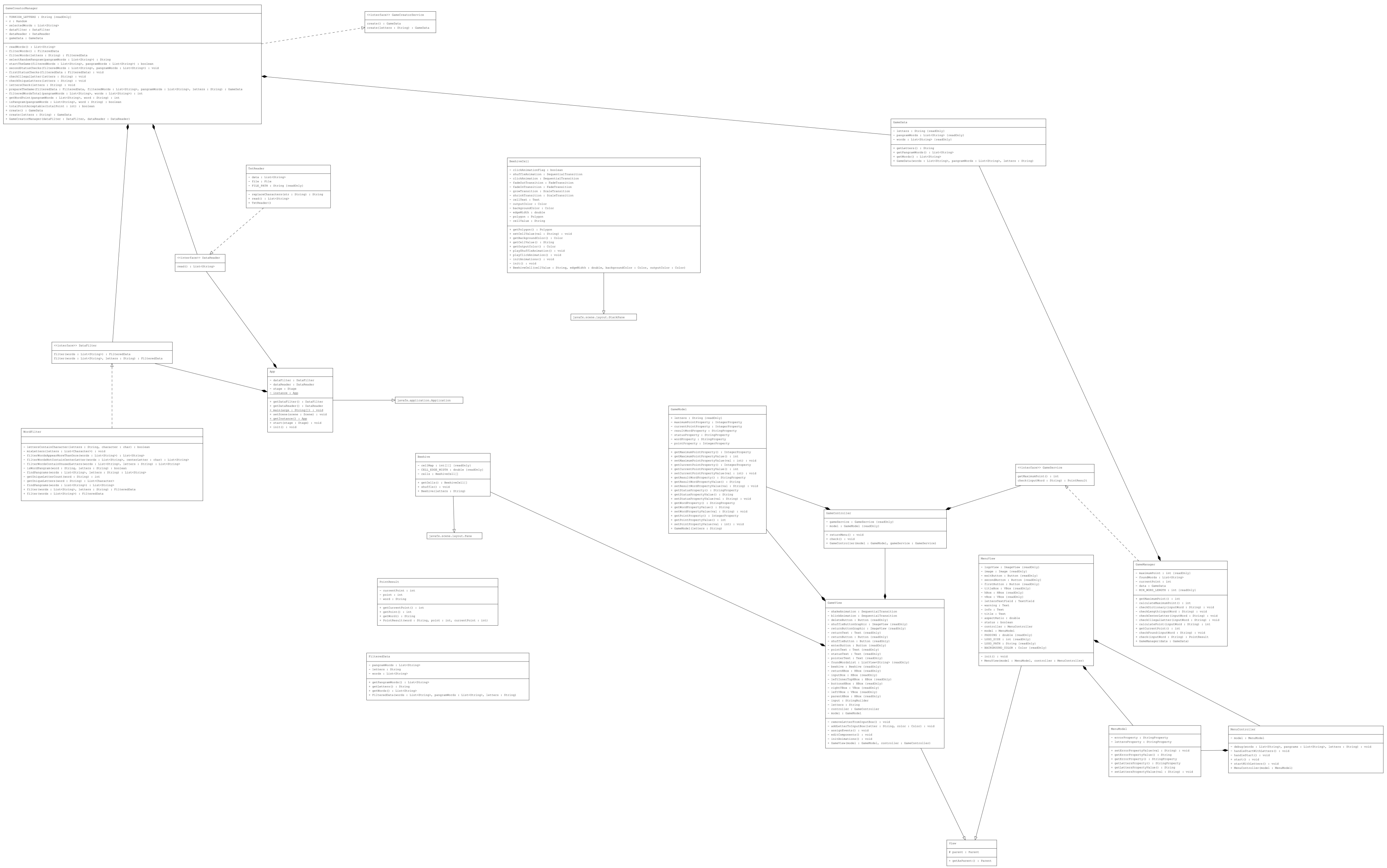
Warning: When you try to build a puzzle by using letters, the center letter of the puzzle will be the fourth letter. For example when you type “kayseri”, “s” will be the center letter. If you want to change center letter you can type “kasyeri” which will make the center letter “y”.

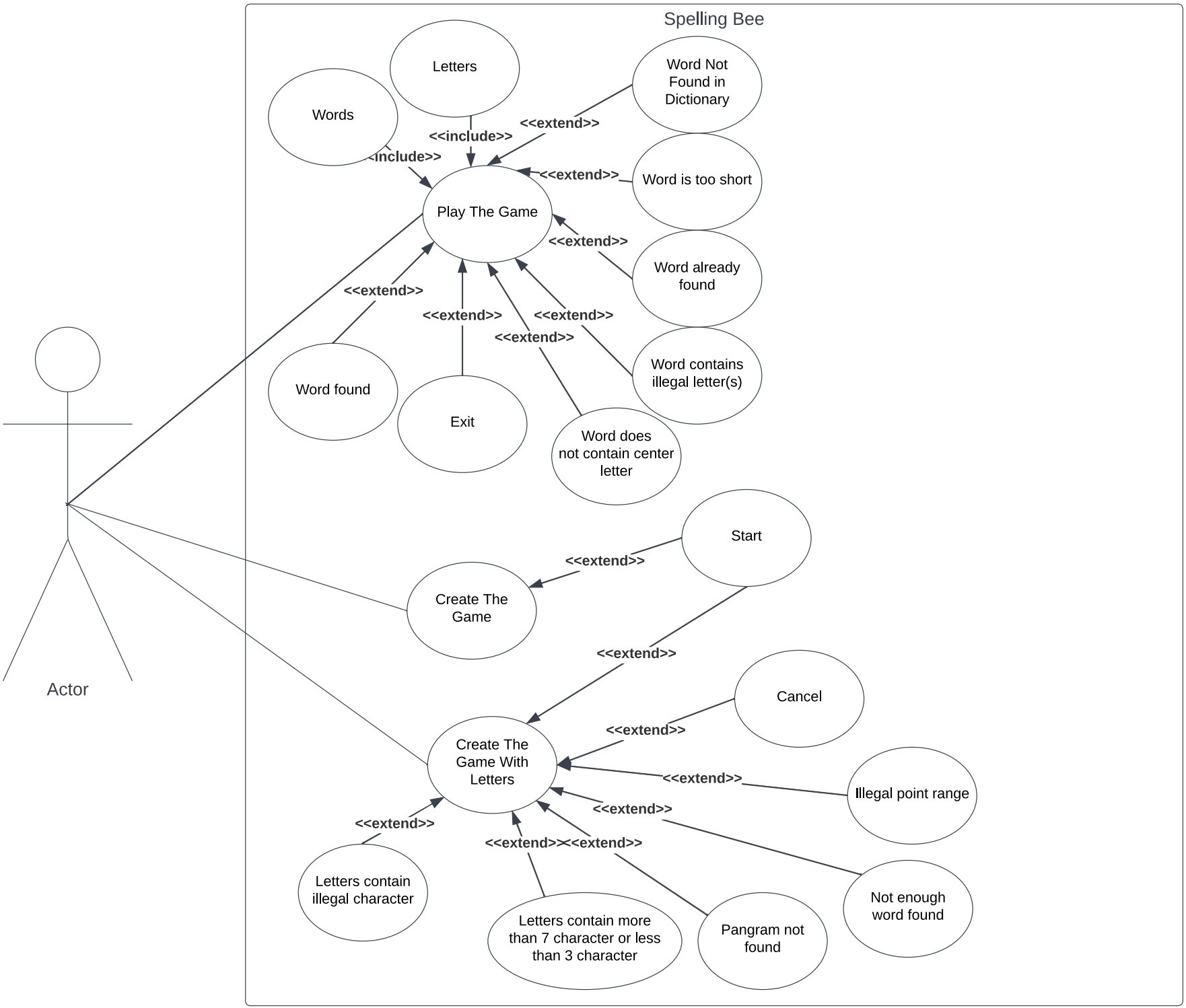
Things we have learned by developing this project

- Using Git version control system
- How to solve problems when working together
- The problems that might appear when working together
- How to use Github features (Pull Request, Issues, etc.)
- Task sharing
- How to work on the same file at the same time
- How to work remote
- Solving code conflicts
- JavaFX

Our opinions about the project

We think that the assignment is well thought out. We have learned lots of things about software design, event-driven programming, and algorithms. It was fun to develop that kind of application.


GameCreatorManager
- TURKISH_LETTERS : String {readOnly}
- r : Random
- selectedWords : List<String>
- dataFilter : DataFilter
- dataReader : DataReader
- gameData : GameData
- readWords() : List<String>
- filterWords() : FilteredData
- filterWords(letters : String) : FilteredData
- selectRandomPangram(pangramWords : List<String>) : String
- startTheGame(filteredWords : List<String>, pangramWords : List<String>) : boolean
- secondStatusChecks(filteredWords : List<String>, pangramWords : List<String>) : void
- firstStatusChecks(filteredData : FilteredData) : void
- checkIllegalLetter(letters : String) : void
- checkUniqueLetters(letters : String) : void
- lettersCheck(letters : String) : void
- prepareTheGame(filteredData : FilteredData, filteredWords : List<String>, pangramWords : List<String>, letters : String) : GameData
- filteredWordsTotal(pangramWords : List<String>, words : List<String>) : int
- getWordPoint(pangramWords : List<String>, word : String) : int
- isPangram(pangramWords : List<String>, word : String) : boolean
- totalPointAcceptable(totalPoint : int) : boolean
+ create() : GameData
+ create(letters : String) : GameData
+ GameCreatorManager(dataFilter : DataFilter, dataReader : DataReader)
<<interface>> GameCreatorService
create() : GameData
create(letters : String) : GameData
TxtReader
- data : List<String>
- file : File
- FILE_PATH : String {readOnly}
- replaceCharacters(str : String) : String
+ read() : List<String>
+ TxtReader()
<<interface>> DataReader
read() : List<String>
BeehiveCell
- clickAnimationFlag : boolean
- shuffleAnimation : SequentialTransition
- clickAnimation : SequentialTransition
- fadeOutTransition : FadeTransition
- fadeInTransition : FadeTransition
- growTransition : ScaleTransition
- shrinkTransition : ScaleTransition
- cellText : Text
- outputColor : Color
- backgroundColor : Color
- edgeWidth : double
- polygon : Polygon
- cellValue : String
+ getPolygon() : Polygon
+ setCellValue(val : String) : void
+ getBackgroundColor() : Color
+ getCellValue() : String
+ getOutputColor() : Color
+ playShuffleAnimation() : void
+ playClickAnimation() : void
- initAnimations() : void
- init() : void
+ BeehiveCell(cellValue : String, edgeWidth : double, backgroundColor : Color, outputColor : Color)
javafx.scene.layout.StackPane
GameData
- letters : String {readOnly}
- pangramWords : List<String> {readOnly}
- words : List<String> {readOnly}
+ getLetters() : String
+ getPangramWords() : List<String>
+ getWords() : List<String>
+ GameData(words : List<String>, pangramWords : List<String>, letters : String)
<<interface>> DataFilter
filter(words : List<String>) : FilteredData
filter(words : List<String>, letters : String) : FilteredData
App
- dataFilter : DataFilter
- dataReader : DataReader
- stage : Stage
- instance : App
+ getDataFilter() : DataFilter
+ getDataReader() : DataReader
+ main(args : String[]) : void
+ setScene(scene : Scene) : void
+ getInstance() : App
+ start(stage : Stage) : void
+ init() : void
javafx.application.Application
WordFilter
- lettersContainCharacter(letters : String, character : char) : boolean
- mixLetters(letters : List<Character>) : void
- filterWordsAppearMoreThanOnce(words : List<String>) : List<String>
- filterWordsNotContainCenterLetter(words : List<String>, centerLetter : char) : List<String>
- filterWordsContainUnusedLetters(words : List<String>, letters : String) : List<String>
- isWordPangram(word : String, letters : String) : boolean
- findPangrams(words : List<String>, letters : String) : List<String>
- getUniqueLetterCount(word : String) : int
- getUniqueLetters(word : String) : List<Character>
- findPangrams(words : List<String>) : List<String>
+ filter(words : List<String>, letters : String) : FilteredData
+ filter(words : List<String>) : FilteredData
Beehive
- cellMap : int[][] {readOnly}
- CELL_EDGE_WIDTH : double {readOnly}
- cells : BeehiveCell[]
+ getCells() : BeehiveCell[]
+ shuffle() : void
+ Beehive(letters : String)
javafx.scene.layout.Pane
PointResult
- currentPoint : int
- point : int
- word : String
+ getCurrentPoint() : int
+ getPoint() : int
+ getWord() : String
+ PointResult(word : String, point : int, currentPoint : int)
FilteredData
- pangramWords : List<String>
- letters : String
- words : List<String>
+ getPangramWords() : List<String>
+ getLetters() : String
+ getWords() : List<String>
+ FilteredData(words : List<String>, pangramWords : List<String>, letters : String)
GameModel
+ letters : String {readOnly}
- maximumPointProperty : IntegerProperty
- currentPointProperty : IntegerProperty
- resultWordProperty : StringProperty
- statusProperty : StringProperty
- wordProperty : StringProperty
- pointProperty : IntegerProperty
+ getMaximumPointProperty() : IntegerProperty
+ getMaximumPointPropertyValue() : int
+ setMaximumPointPropertyValue(val : int) : void
+ getCurrentPointProperty() : IntegerProperty
+ getCurrentPointPropertyValue() : int
+ setCurrentPointPropertyValue(val : int) : void
+ getResultWordProperty() : StringProperty
+ getResultWordPropertyValue() : String
+ setResultWordPropertyValue(val : String) : void
+ getStatusProperty() : StringProperty
+ getStatusPropertyValue() : String
+ setStatusPropertyValue(val : String) : void
+ getWordProperty() : StringProperty
+ getWordPropertyValue() : String
+ setWordPropertyValue(val : String) : void
+ getPointProperty() : IntegerProperty
+ getPointPropertyValue() : int
+ setPointPropertyValue(val : int) : void
+ GameModel(letters : String)
<<interface>> GameService
getMaximumPoint() : int
check(inputWord : String) : PointResult
GameController
- gameService : GameService {readOnly}
- model : GameModel {readOnly}
+ returnMenu() : void
+ check() : void
+ GameController(model : GameModel, gameService : GameService)
GameView
- shakeAnimation : SequentialTransition
- blinkAnimation : SequentialTransition
- deleteButton : Button {readOnly}
- shuffleButtonGraphic : ImageView {readOnly}
- returnButtonGraphic : ImageView {readOnly}
- returnText : Text {readOnly}
- returnButton : Button {readOnly}
- shuffleButton : Button {readOnly}
- enterButton : Button {readOnly}
- pointText : Text {readOnly}
- statusText : Text {readOnly}
- pointerText : Text {readOnly}
- foundWordsList : ListView<String> {readOnly}
- beehive : Beehive {readOnly}
- returnHBox : HBox {readOnly}
- inputBox : HBox {readOnly}
- leftInnerTopHBox : HBox {readOnly}
- buttonsHBox : HBox {readOnly}
- rightVBox : VBox {readOnly}
- leftVBox : VBox {readOnly}
- parentHBox : HBox {readOnly}
- input : StringBuilder
- letters : String
- controller : GameController
- model : GameModel
- removeLetterFromInputBox() : void
- addLetterToInputBox(letter : String, color : Color) : void
- assignEvents() : void
- editComponents() : void
- initAnimations() : void
+ GameView(model : GameModel, controller : GameController)
MenuView
- logoView : ImageView {readOnly}
- image : Image {readOnly}
- exitButton : Button {readOnly}
- secondButton : Button {readOnly}
- firstButton : Button {readOnly}
- titleBox : VBox {readOnly}
- hBox : HBox {readOnly}
- vBox : VBox {readOnly}
- lettersTextField : TextField
- warning : Text
- info : Text
- title : Text
- aspectRatio : double
- status : boolean
- controller : MenuController
- model : MenuModel
- PADDING : double {readOnly}
- LOGO_SIZE : int {readOnly}
- LOGO_PATH : String {readOnly}
- BACKGROUND_COLOR : Color {readOnly}
- init() : void
+ MenuView(model : MenuModel, controller : MenuController)
GameManager
- maximumPoint : int {readOnly}
- foundWords : List<String>
- currentPoint : int
- data : GameData
- MIN_WORD_LENGTH : int {readOnly}
+ getMaximumPoint() : int
- calculateMaximumPoint() : int
- checkDictionary(inputWord : String) : void
- checkLength(inputWord : String) : void
- checkCenterLetter(inputWord : String) : void
- checkIllegalLetter(inputWord : String) : void
- calculatePoint(inputWord : String) : int
+ getCurrentPoint() : int
- checkFound(inputWord : String) : void
+ check(inputWord : String) : PointResult
+ GameManager(data : GameData)
MenuModel
- errorProperty : StringProperty
- lettersProperty : StringProperty
+ setErrorPropertyValue(val : String) : void
+ getErrorPropertyValue() : String
+ getErrorProperty() : StringProperty
+ getLettersProperty() : StringProperty
+ getLettersPropertyValue() : String
+ setLettersPropertyValue(val : String) : void
MenuController
- model : MenuModel
- debug(words : List<String>, pangrams : List<String>, letters : String) : void
- handleStartWithLetters() : void
- handleStart() : void
+ start() : void
+ startWithLetters() : void
+ MenuController(model : MenuModel)
View
parent : Parent
+ getAsParent() : Parent


Spelling Bee
Letters
Words
Word Not Found in Dictionary
<<include>>
<<include>>
<<extend>>
Word is too short
Play The Game
<<extend>>
Word already found
<<extend>>
<<extend>>
<<extend>>
<<extend>>
Word found
Exit
Word does not contain center letter
Word contains illegal letter(s)
Actor
Start
<<extend>>
Create The Game
<<extend>>
Cancel
<<extend>>
Create The Game With Letters
<<extend>>
Illegal point range
<<extend>>
<<extend>>
<<extend>>
Letters contain illegal character
Letters contain more than 7 character or less than 3 character
Pangram not found
Not enough word found