**BUZZWORD**

**Software Design Description**

**Author:**  Hridesh Kapur

**Abstract:** This document describes the software design for the game Buzzword.

**Based on IEEE Std 1016TM-2009 document format**

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

# 1 Introduction

This is the Software Design Description (SDD) for the Buzzword application. Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

## 1.1 Purpose

This document is to serve as the blueprint for the construction of the Buzzword application. This design will use UML class diagrams to provide complete detail regarding all packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence diagrams will be used to specify object interactions post-initialization of the application, meaning in response to user interactions or timed events.

## 1.2 Scope

Buzzword is the first of a many more word-games to come. Therefore, it is crucial to make design 3 decisions that will lead to reusable components. This paves the way for framework(s) to be designed and constructed such that they can be employed for similar other word-games in future. Likewise, this document describes a Java Framework for Language Applications and Games (JFLAG) that provides the foundational building blocks for the development of such games. JFLAG, however, is specific to word-games, and is not meant to be used as a general purpose framework for the development of all kinds of educational games.

## 1.3 Definitions, acronyms, and abbreviations

**Class Diagram –** A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

**IEEE –** Institute of Electrical and Electronics Engineers, the “world’s largest professional association for the advancement of technology”.

**Framework** – In an object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

**Java** – A high-level programming language that uses a virtual machine layer between the Java application and the hardware to provide program portability.

**Sequence Diagram** – A UML document format that specifies how object methods interact with one another.

**UML** – Unified Modeling Language, a standard set of document formats for designing software graphically.

Educational games - Educational games are games that are designed to help people to learn about certain subjects, expand concepts, reinforce development, understand an historical event or culture, or assist them in learning a skill as they play.

Word games - Word games are games that involve making, guessing, or selecting words.

## 

## 1.4 References

**IEEE Std 830TM-1998 (R2009) –** IEEE Standard for Information Technology – Systems Design – Software Design Descriptions

## 

## 1.5 Overview

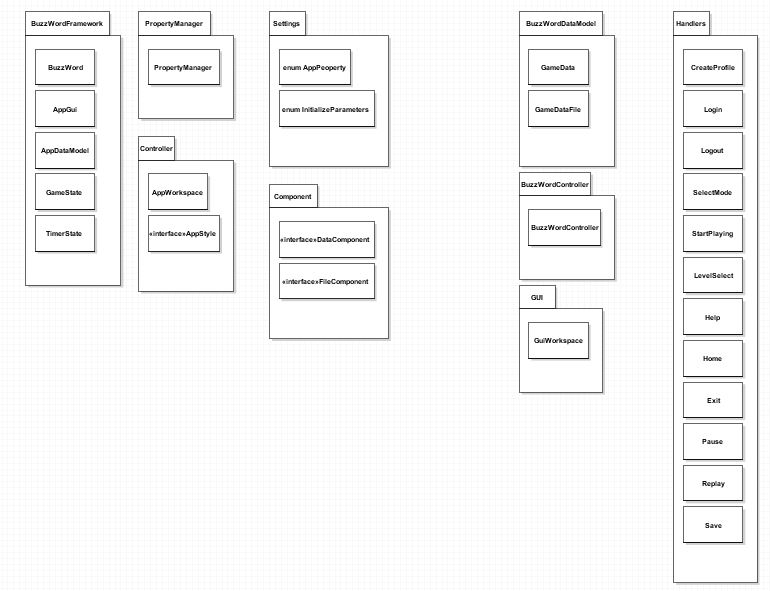
This Software Design Description document provides a working design for the Buzzword software application as described in the Buzzword Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 will provide the Method-Level System Viewpoint, describing how methods will interact with one another. Section 5 provides deployment information like file structures and formats to use. Section 6 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the VioletUML editor.

# 2 Package-Level Design Viewpoint

As mentioned, this design will encompass both the BuzzWord game application and the BuzzWordController to be used in its construction. In building both we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

## 2.1 BuzzWord and BuzWordController Overview

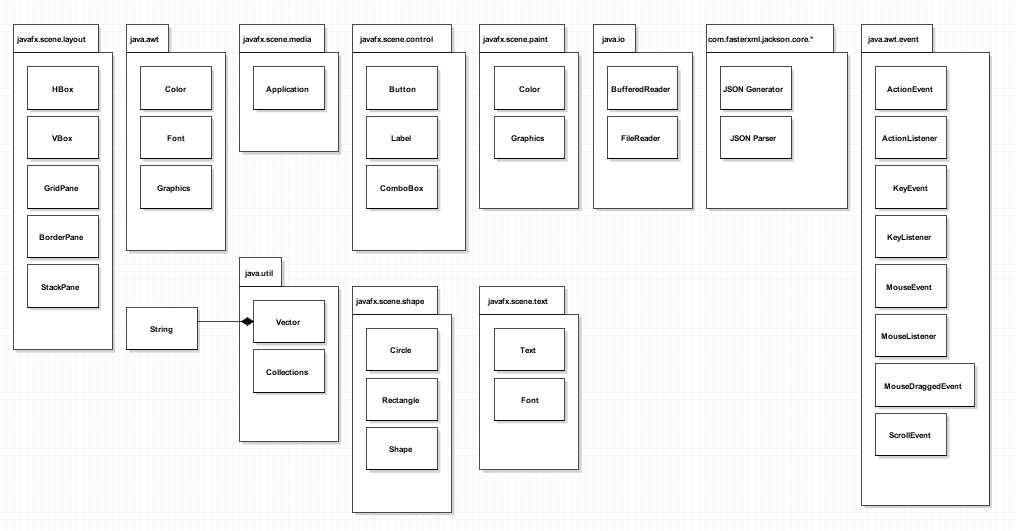
The Buzzword and BuzzwordController framework will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages.



**Figure 2.1: Design Packages Overview**

## 2.2 Java API Usage

Both the framework and the Controller application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.



**Figure 2.2: Java API Classes and Packages To Be Used**

## 2.3 Java API Usage Descriptions

Tables 2.1-2.10 below summarize how each of these classes will be used.

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Color** | For setting the rendering colors for text and the progress bar |
| **Font** | For setting the fonts for rendered text |
| **Graphics** | For rendering text, images, and shapes to the canvas |
| **Image** | For storing image data |
| **Insets** | For changing component margins |
| **MediaTracker** | For ensuring synchronous image loading |
| **Toolkit** | For loading images |

**Table 2.1: Uses for classes in the Java API’s java.awt package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **ActionEvent** | For getting information about an action event like which button was pressed. |
| **ActionListener** | For responding to an action event, like a button press. We will provide our own custom implementation of this interface. |
| **KeyEvent** | For getting information about a key event, like which key was pressed. |
| **KeyListener** | For responding to a key event, like a key press. We will provide our own custom implementation of this interface. |
| **MouseEvent** | For getting information about a mouse event, like where was the mouse pressed? |
| **MouseListener** | For responding to a mouse event, like a mouse button press. We will provide our own custom implementation of this interface. |
| **MouseDraggedEvent** | For responding to a mouse dragged event, like a mouse drag. |
| **ScrollEvent** | For responding to a scroll event, like a mouse scroll. |

**Table 2.2: Uses for classes in the Java API’s java.awt.event package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Application** | The entry point for JavaFX applications is the Application class. |

**Table 2.3: Uses for classes in the Java API’s javafx.scene.media package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **BufferedReader** | For reading text files, we’ll use this for loading some game data at startup. |
| **FileReader** | For reading files. |

**Table 2.4: Uses for classes in the Java API’s java.io package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Collection** | For storing groups of data, Values in a Map, i.e. TreeMap or  HashMap are stored in Collections. We’ll need to iterate through Collections for rendering. |
| **HashMap** | For storing (name,value) key pairs, we’ll use it for storing our Images, accessible using their ID names. |
| **Vector** | For storing data like the Strings for rendering debugging text. |

**Table 2.5: Uses for classes in the Java API’s java.util package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **HBox** | Provides a horizontal boxfor our GUI. |
| **VBox** | Provides vertical box for our GUI. |
| **GridPane** | Provides a grid pane for the game to be rendered onto. |
| **BorderPane** | Provides a border pane for the game to be rendered onto. |
| **StackPane** | Provides a stack pane for the game to be rendered onto. |

**Table 2.6: Uses for classes in the Java API’s javafx.scene.layout package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Button** | The button control contains text and/or a graphic. |
| **Label** | Used to display text within a specific space. |
| **ComboBox** | Used to make the drop-down list for the game modes. |

**Table 2.7: Uses for classes in the Java API’s javafx.scene.control package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **JSON Generator** | Writes JSON data to an output source in a streaming way. |
| **JSON Parser** | Provides forward, read-only access to JSON data in a streaming way. |

**Table 2.8: Uses for classes in the Java API’s com.fasterxml.jackson.core.\* package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Circle** | The Circle class creates a new circle with the specified radius and center location measured in pixels |
| **Rectangle** | The Rectangle class defines a rectangle with the specified size and location. |
| **Shape** | The Shape class provides definitions of common properties for objects that represent some form of geometric shape. |

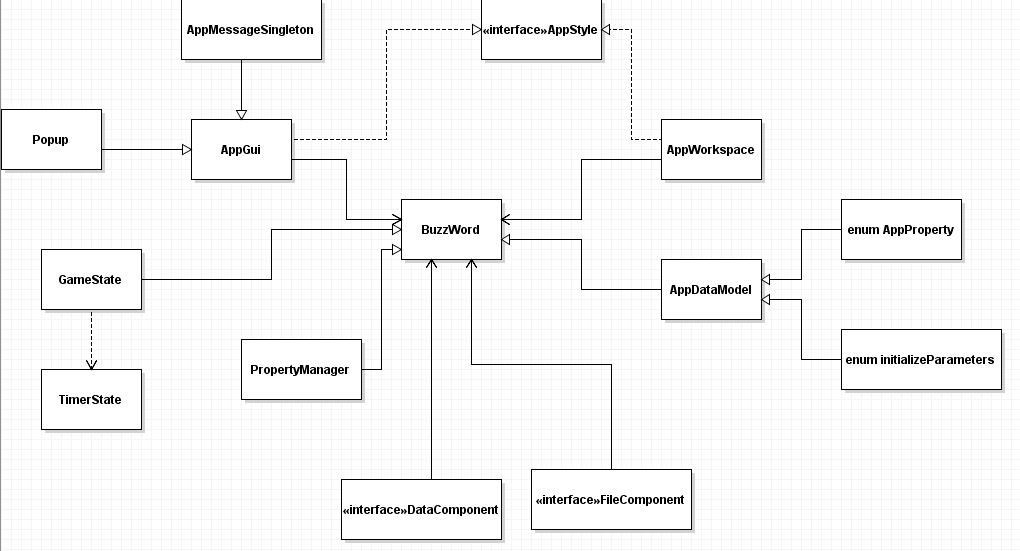
**Table 2.9: Uses for classes in the Java API’s javafx.scene.shape package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Text** | The Text class defines a node that displays a text. |
| **Font** | The Font class represents fonts, which are used to render text on screen. |

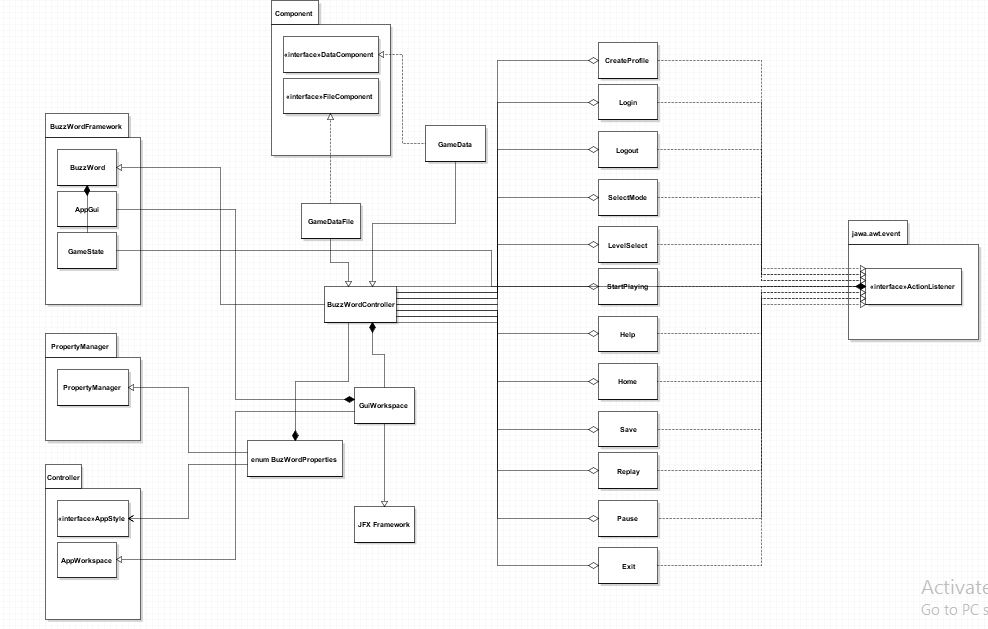
**Table 2.10: Uses for classes in the Java API’s javafx.scene.text package**

# 3 Class-Level Design Viewpoint

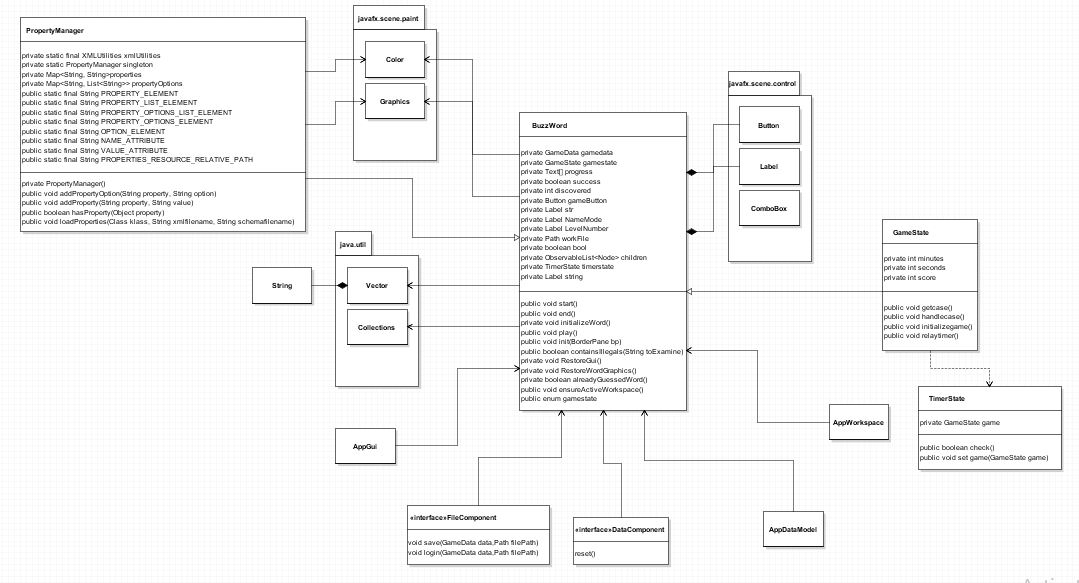
As mentioned, this design will encompass both the Buzzword game application and the BuzzwordController. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones.



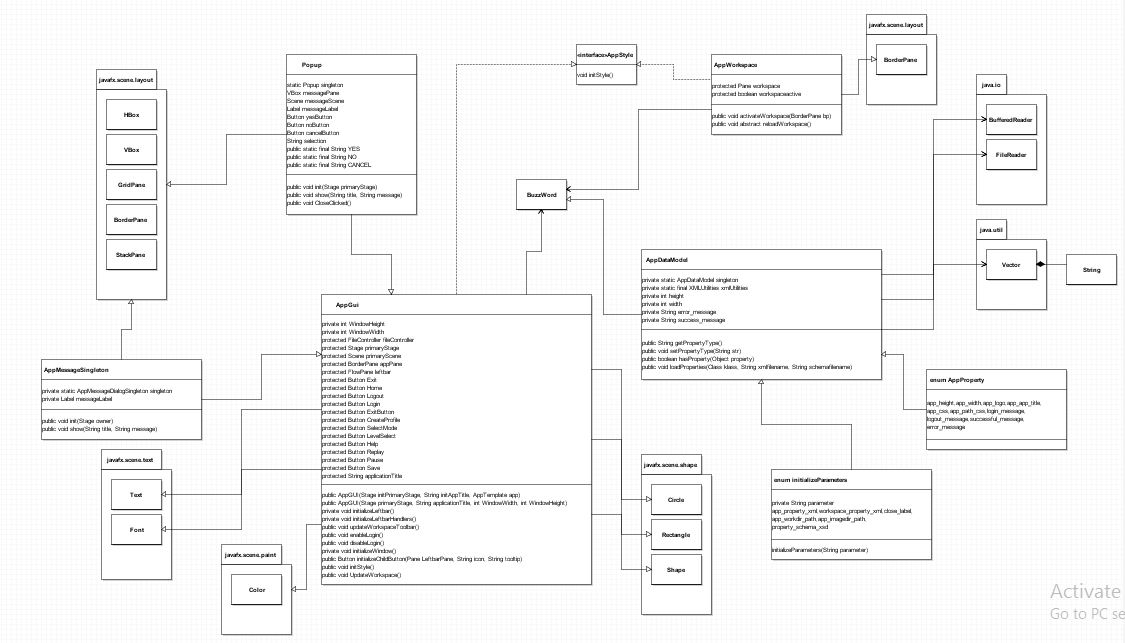
**Figure 3.1: BuzzWord FrameWork Overview**



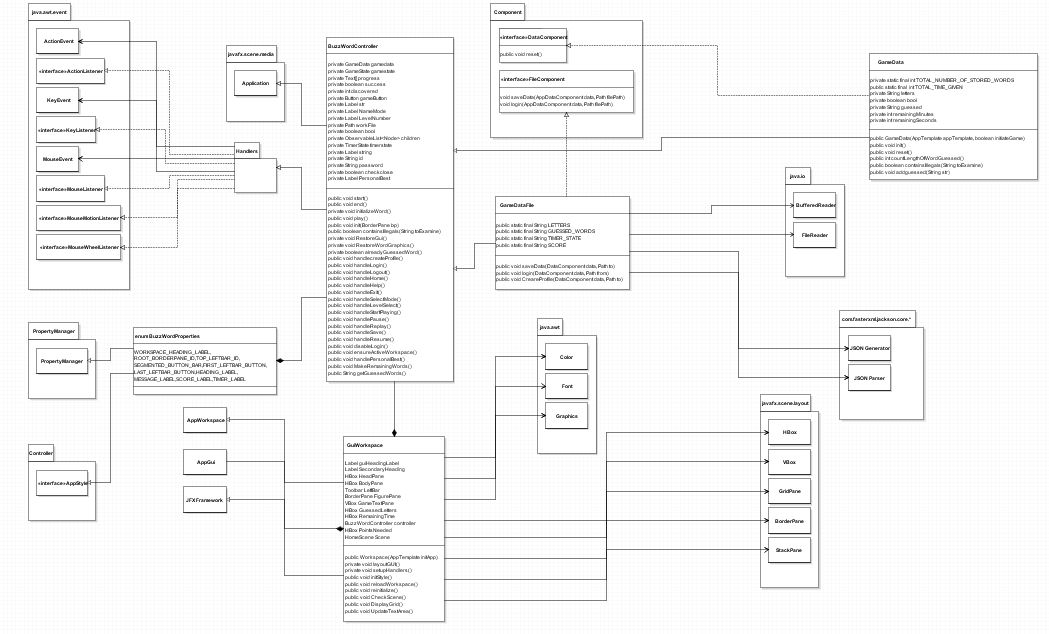
**Figure 3.2: BuzzWord Controller Overview UML Class Diagram**



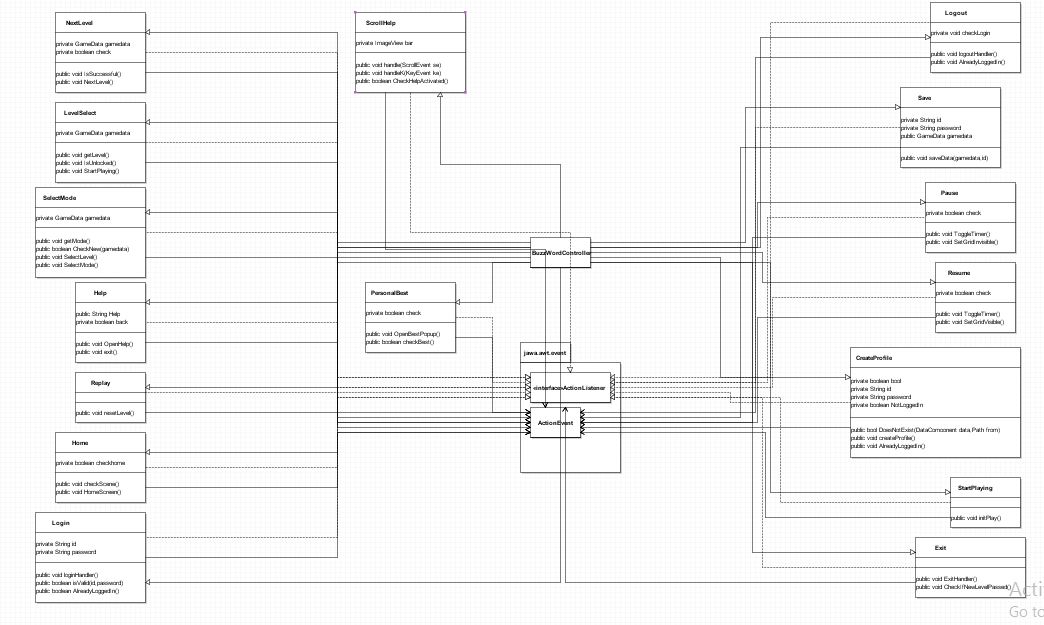
**Figure 3.3: Detailed BuzzWord UML Class Diagram**



**Figure 3.4: Detailed BuzzWord UML Class Diagram 2nd Part**



**Figure 3.5: Detailed BuzzWordController UML Class Diagram**

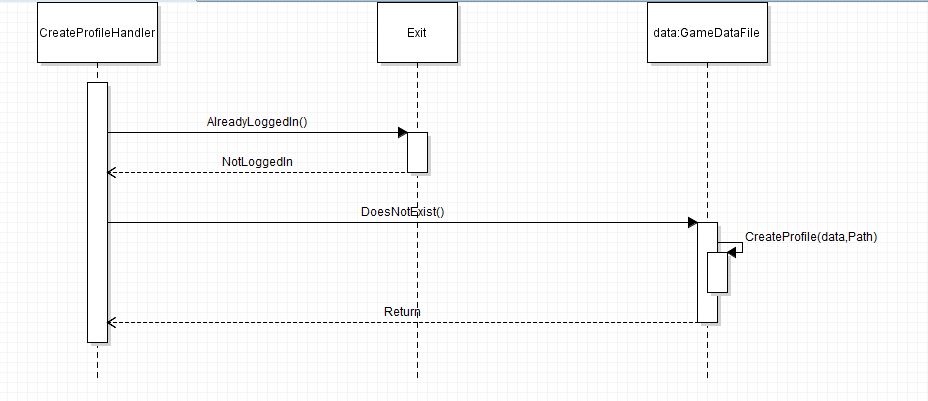


**Figure 3.6: Detailed Handlers UML Class Diagram**

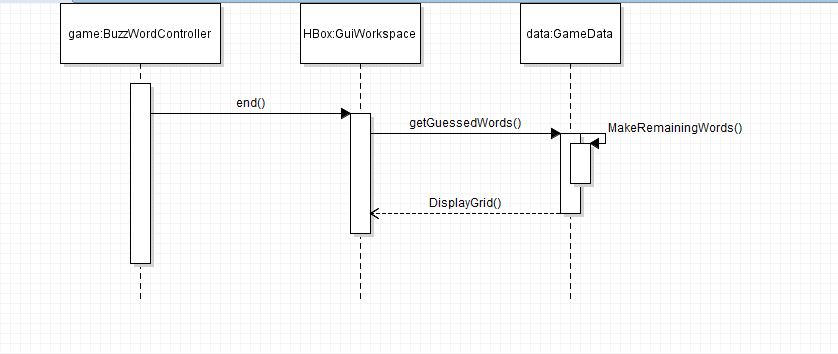
**Diagrams**

# 4 Method-Level Design Viewpoint

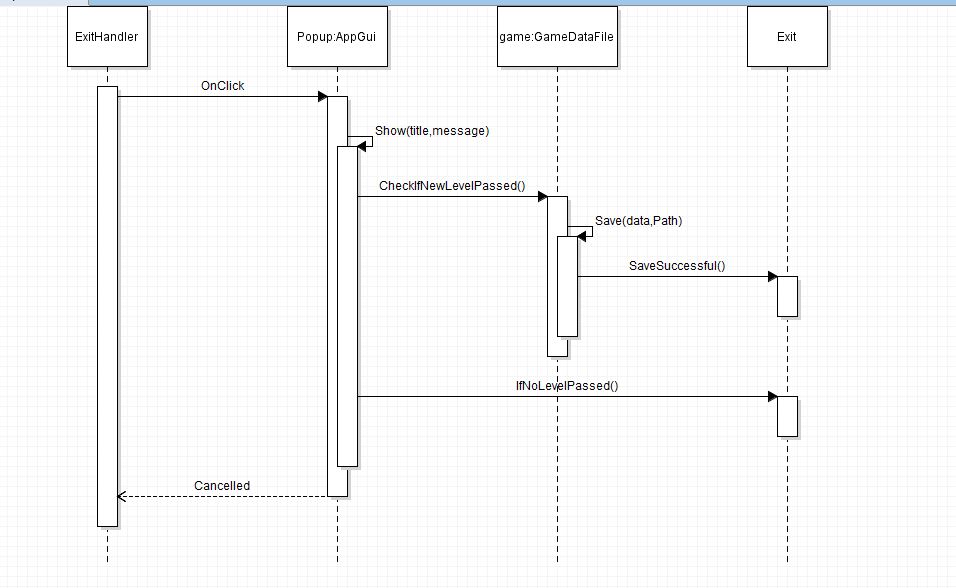
Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.



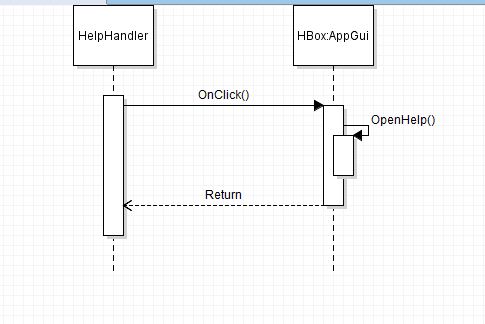
**Figure 4.1: CreateProfile UML Sequence Diagrams**



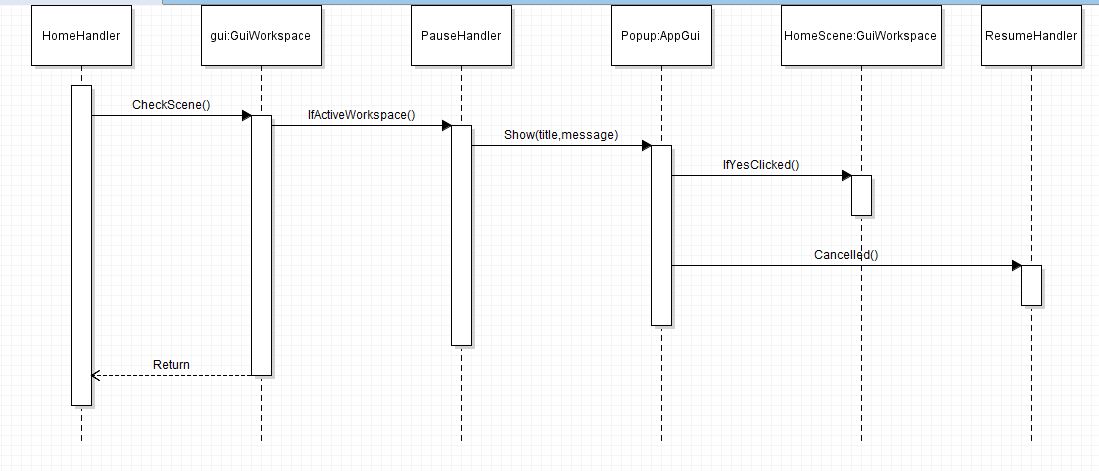
**Figure 4.2: GameEnds UML Sequence Diagrams**



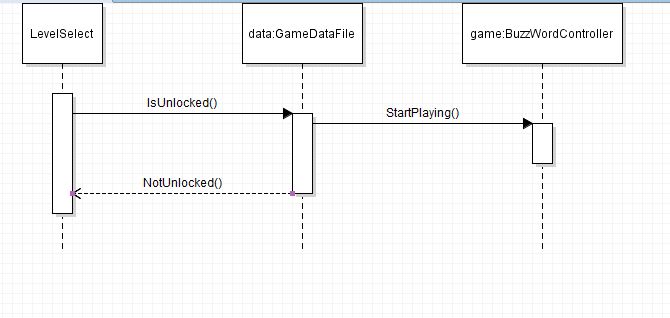
**Figure 4.3: Exit UML Sequence Diagrams**



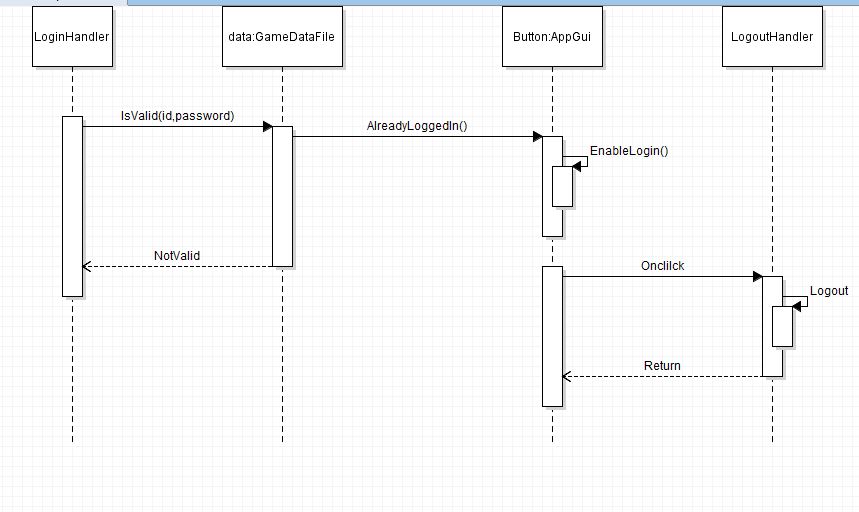
**Figure 4.4: Help UML Sequence Diagrams**



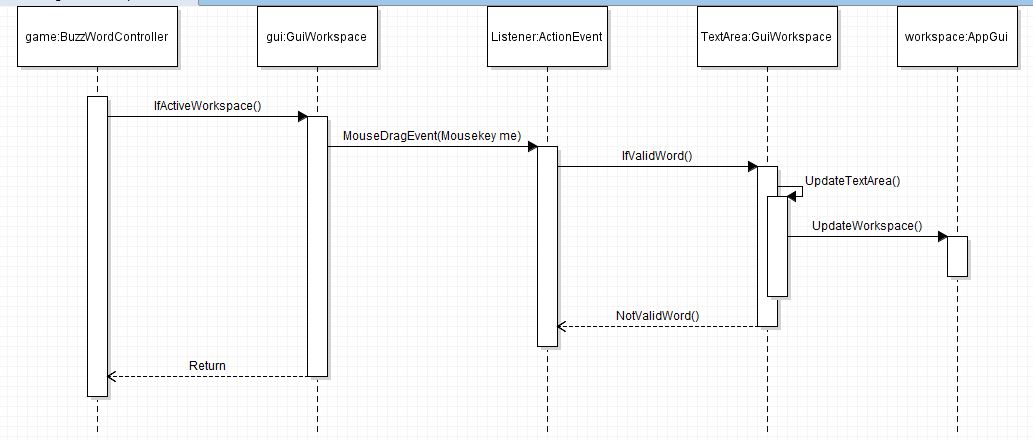
**Figure 4.5: Home UML Sequence Diagrams**

****

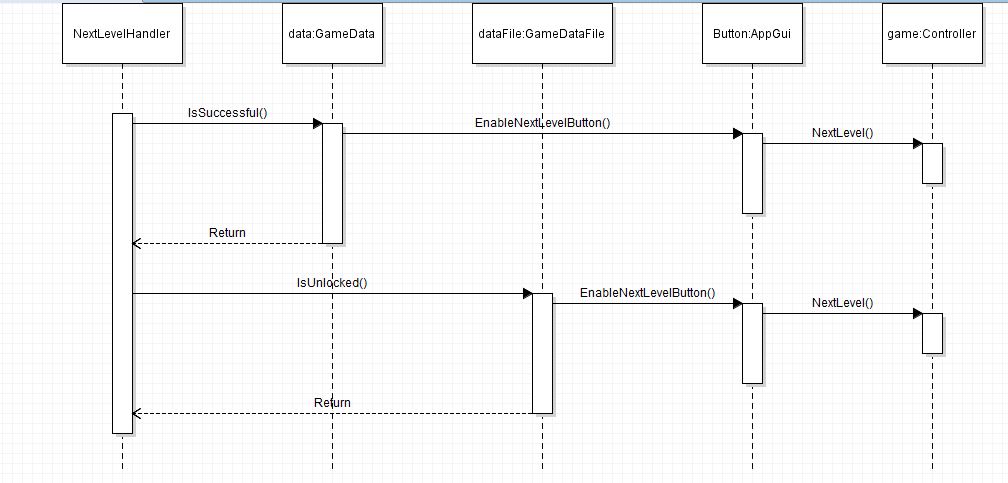
**Figure 4.6: LevelSelect UML Sequence Diagrams**

****

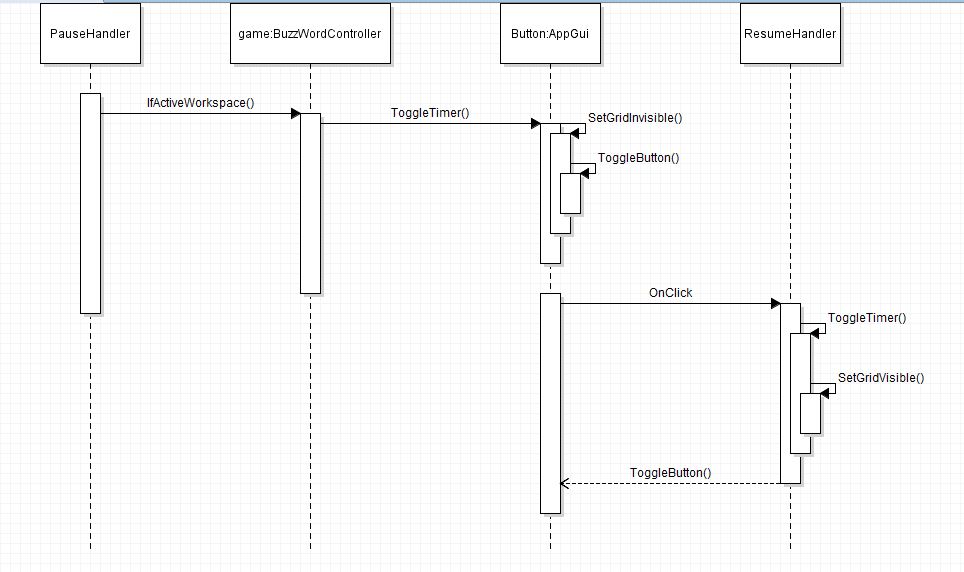
**Figure 4.7: Login/Logout UML Sequence Diagrams**



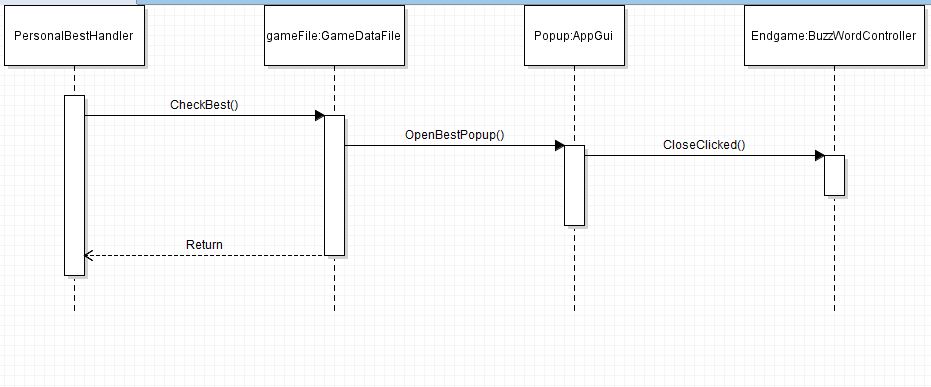
**Figure 4.8: MouseDragged UML Sequence Diagrams**



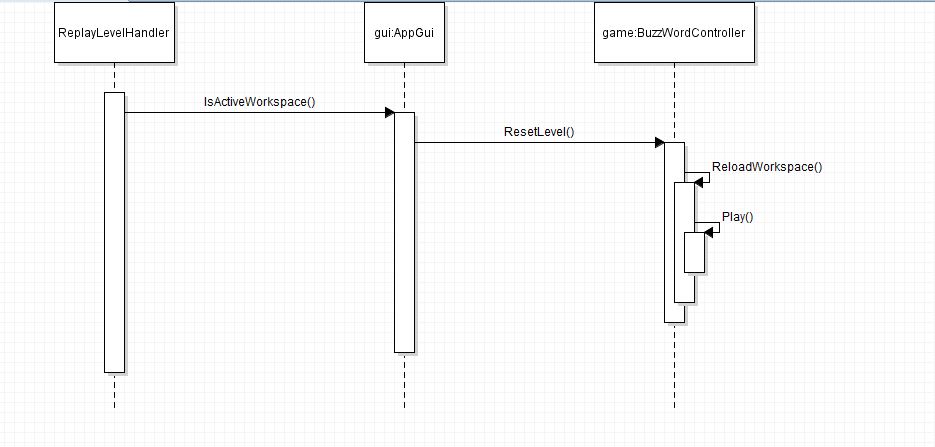
**Figure 4.9: NextLevel UML Sequence Diagrams**



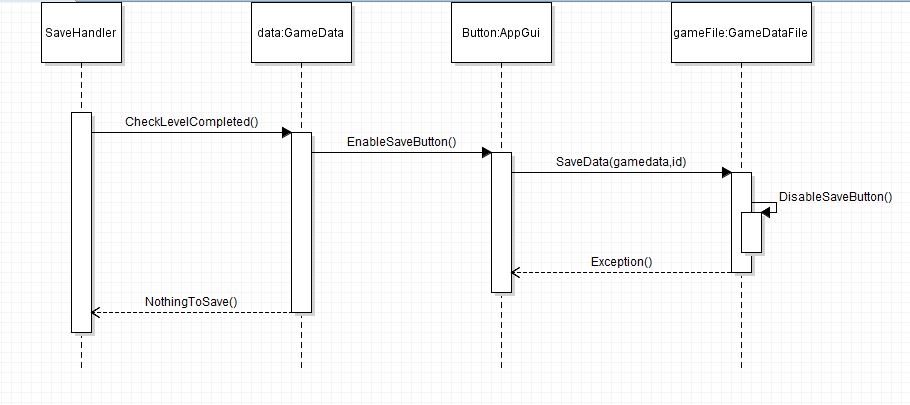
**Figure 4.10: Pause/Resume UML Sequence Diagrams**



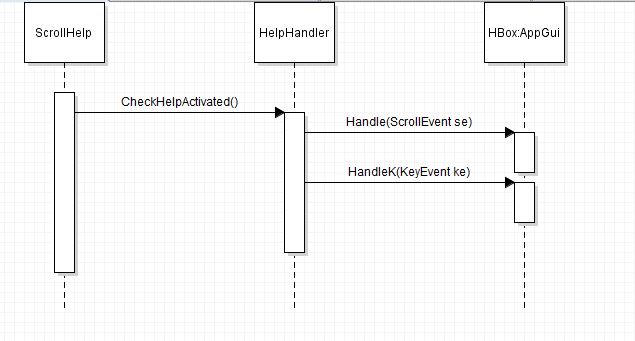
**Figure 4.11: PersonalBest UML Sequence Diagrams**

****

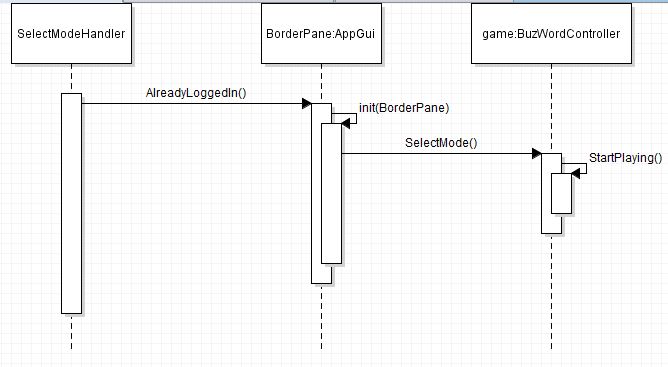
**Figure 4.12: ReplayLevel UML Sequence Diagrams**

****

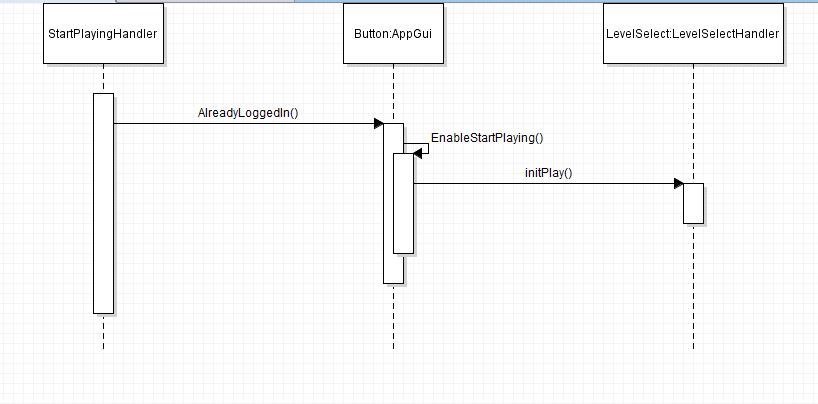
**Figure 4.13: Save UML Sequence Diagrams**



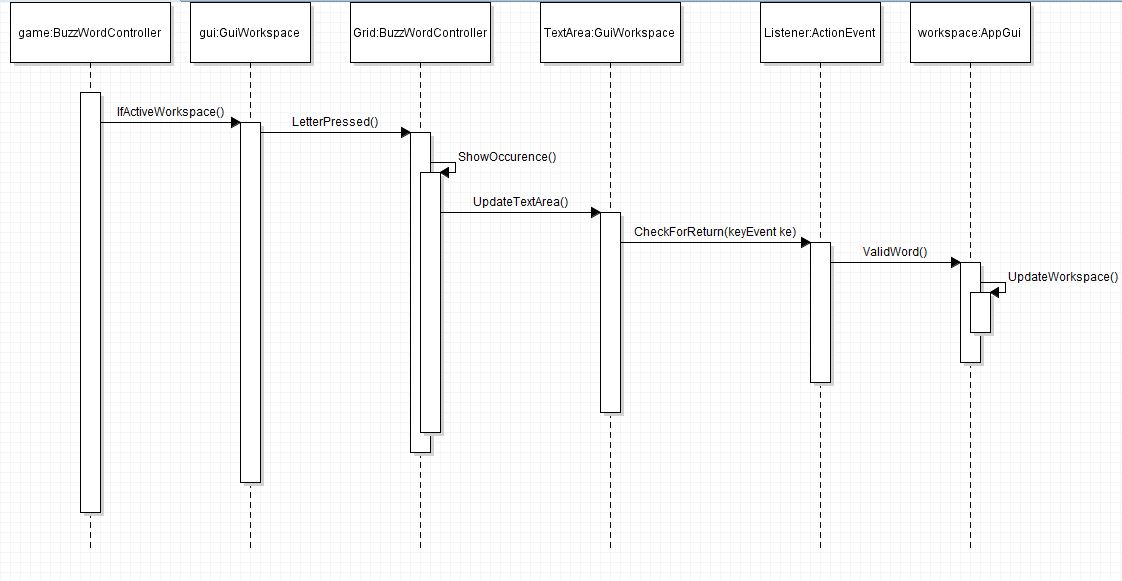
**Figure 4.14: ScrollHelp UML Sequence Diagrams**



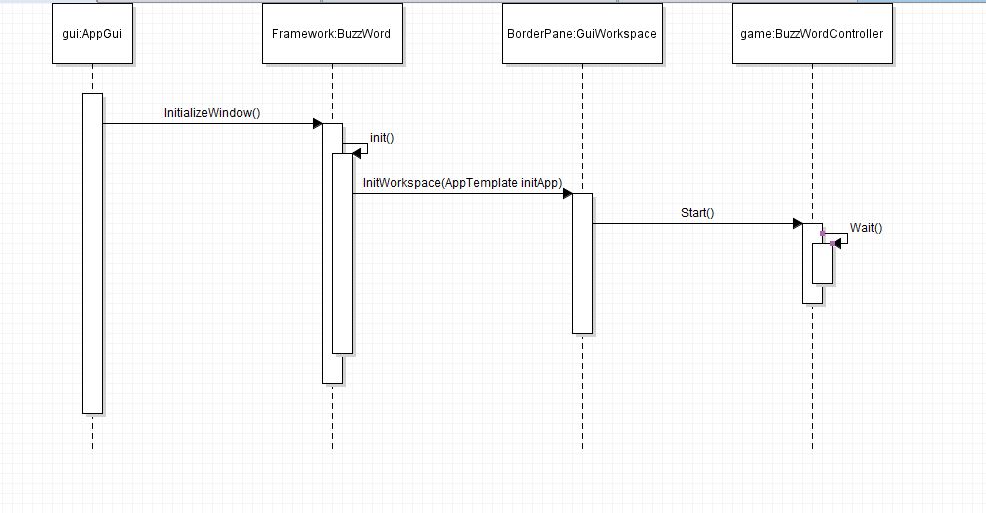
**Figure 4.15: SelectMode UML Sequence Diagrams**



**Figure 4.16: StartPlaying UML Sequence Diagrams**



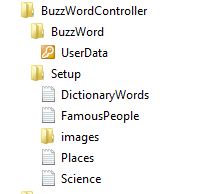
**Figure 4.17: WordTyped UML Sequence Diagrams**



**Figure 4.18: Launch UML Sequence Diagrams**

# 5. File Structure and Formats

Note that the BuzzWord Framework will be provided inside BuzzWord.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the BuzzWord application and will be included in the deployment of a single, executable JAR file titled BuzzWordController.jar. Note that all necessary data and art files must accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use. Note that all necessary images should of course go in the image directory.



**Figure 5.1: BuzzWord File Structure**

The DictionaryWords.txt provides all the words required to be imported to by the project for the Dictionary levels. TheFamousPeople.txt provides all the words required to be imported to by the project for the FamousPeople levels. The Places.txt provides all the words required to be imported to by the project for the Places levels. The Science.txt provides all the words required to be imported to by the project for the Science levels. The images folder contains any images used in the project. The UserData is a protected file which has access to all login ids and passwords.

## 6.1 Table of contents

[1. Introduction 2](#_Toc19584)

[1. Purpose 2](#_Toc19585)

[2. Scope 2](#_Toc19586)

[3. Definitions, acronyms, and abbreviations 2](#_Toc19587)

[4. References 3](#_Toc19588)

[5. Overview 3](#_Toc19589)

[2. Package-Level Design Viewpoint 4](#_Toc19590)

[1. BuzzWord and BuzWordController Overview 4](#_Toc19591)

[2. Java API Usage 5](#_Toc19592)

[3. Java API Usage Descriptions 5](#_Toc19593)

[3. Class-Level Design Viewpoint 8](#_Toc19594)

[4. Method-Level Design Viewpoint 14](#_Toc19595)

[5. File Structure and Formats 17](#_Toc19596)

[6. Supporting Information 18](#_Toc19597)

[1. Table of contents 18](#_Toc19598)

[2. Appendixes 18](#_Toc19599)

## 6.2 Appendixes

N/A