Zombiquarium™
Software Design Description

Author: Richard McKenna
Debugging Enterprises™
November, 2011
Version 1.0

Abstract: This document describes the software design for Zombiquarium, a casual mini-game in development as part of Plants vs. Zombies.

Based on IEEE Std 1016™-2009 document format

1 Introduction

This is the Software Design Description (SDD) for the Zombiquarium™ mini-game 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 Zombiquarium 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

Zombiquarium will be one mini-game among many to be included in the Plants vs. Zombies application. Tools for its construction should be developed with this in mind such that additional mini-games may avoid duplication of work. As such, a framework called the MiniGame Framework, will be designed and constructed along with the Zombiquarium game such that it may be used to build additional mini-games. So, this design contains design descriptions for the development of both the framework and game. Note that Java is the target language for this software design.

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.

Mini-Game – A standalone game that is a subset of a larger game application, typically sharing the primary game theme with that parent game application.

Mini Game Framework – The software framework to be developed in tandem with the Zombiquarium game such that additional mini-games can easily be constructed. Note that in the Zombiquarium SRS this was sometimes called the "Mini Zombie Game Framework", but has been renamed the "Mini Game Framework", since it's not Zombie-specific.

Plants vs. Zombies – The PopCap Games game that is the parent application of our Zombiquarium mini-game. Note that Zombiquarium is to be distributed as part of that program.

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

Sprite – a renderable, and sometimes movable or clickable image in the game. Each Sun, Zombie, and Brain will be its own Sprite, as will GUI controls.

SpriteType – a type of Sprite, meaning all the artwork and states corresponding to a category of sprite. We do this because all the suns share artwork, so we will load all their artwork into a common Sprite Type, but each Sprite has its own position and velocity, so each will be its own Sprite that knows what Sprite Type it belongs to.

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

Zombie – An undead creature, meaning something that has died and then come back to life. These beings are typically slow moving and love to eat brains.

Zombiquarium – The title of the mini-game described by this document. Again, note that this game will be distributed as part of the Plants vs. Zombies application.

1.4 References

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

Zombiquarium™ SRS – Debugging Enterprises' Software Requirements Specification for the Zombiquarium mini-game application.

1.5 Overview

This Software Design Description document provides a working design for the Zombiquarium software application as described in the Zombiquarium 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 Zombiquarium game application and the Mini-Game Framework 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 Zombiquarium and Mini Game overview

The Zombiquarium and MiniGame 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.

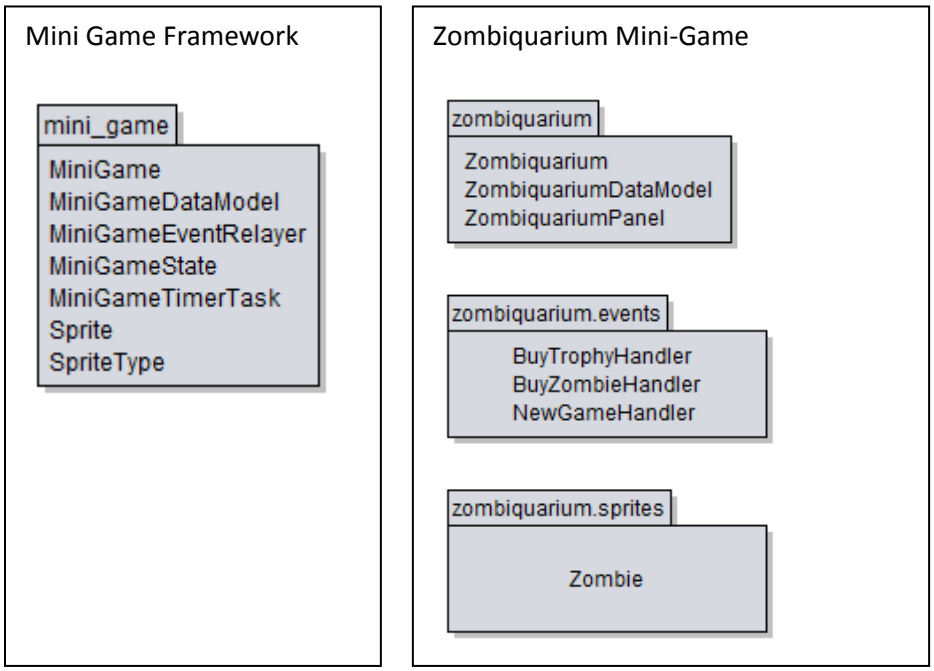

Figure 2.1: Design Packages Overview

2.2 Java API Usage

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

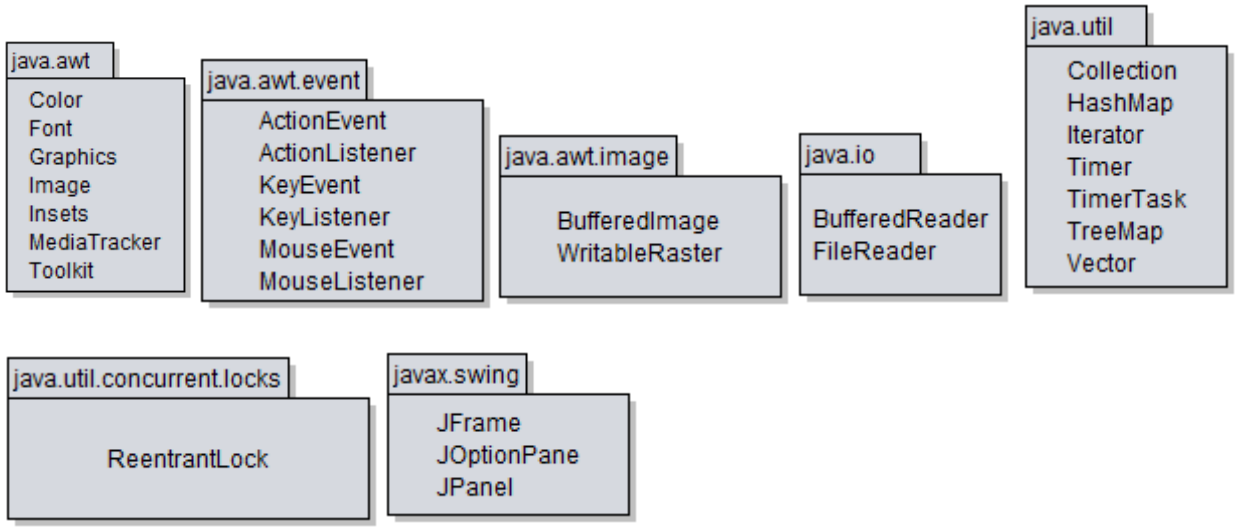

Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions

Tables 2.1-2.7 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.

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

Class/Interface	Use
BufferedImage	For storing image data where pixel information can be accessed and changed.
WritableRaster	For changing pixel data in a BufferedImage. We'll use this to add transparency to pixels loaded with the color key.

Table 2.3: Uses for classes in the Java API's java.awt.image 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.
Iterator	For iterating through a data structure during operations like rendering.
Timer	Will execute our custom task at fixed intervals.
TimerTask	Our custom task to be executed at a fixed framerate. We will extend this class and provide the task implementation in the run method.
TreeMap	For storing (name,value) key pairs, we'll use it for storing SpriteTypes and GUI components, 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.io package

Class/Interface	Use
ReentrantLock	For ensuring only one thread has access to program data.

Table 2.6: Uses for classes in the Java API's java.util.concurrent package

Class/Interface	Use
JFrame	Provides the window for our GUI.
JOptionPane	Provides a popup dialog for error feedback.
JPanel	Provides a canvas for the game to be rendered onto.

Table 2.7: Uses for classes in the Java API's javax.swing package

3 Class-Level Design Viewpoint

As mentioned, this design will encompass both the Zombiquarium game application and the Mini-Game Framework. 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.

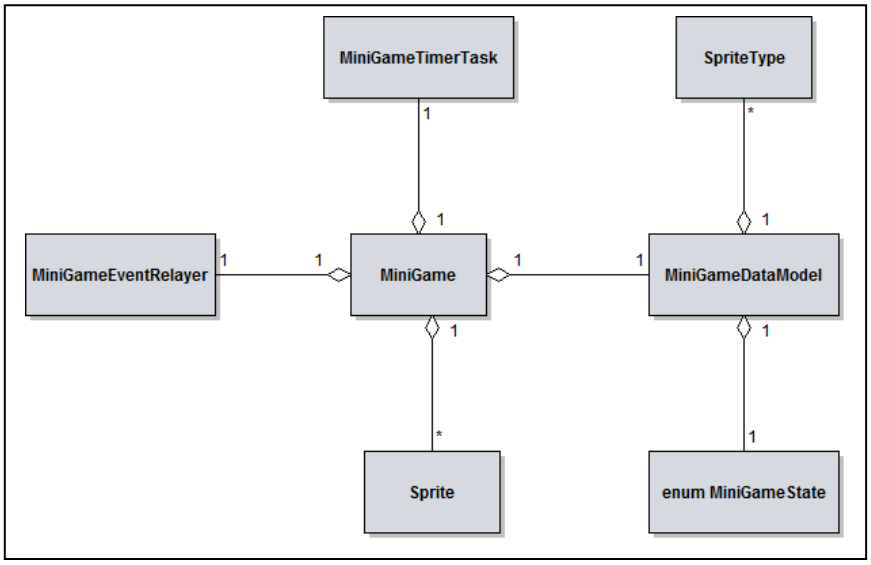

Figure 3.1: Mini-Game Framework Overview UML Class Diagram

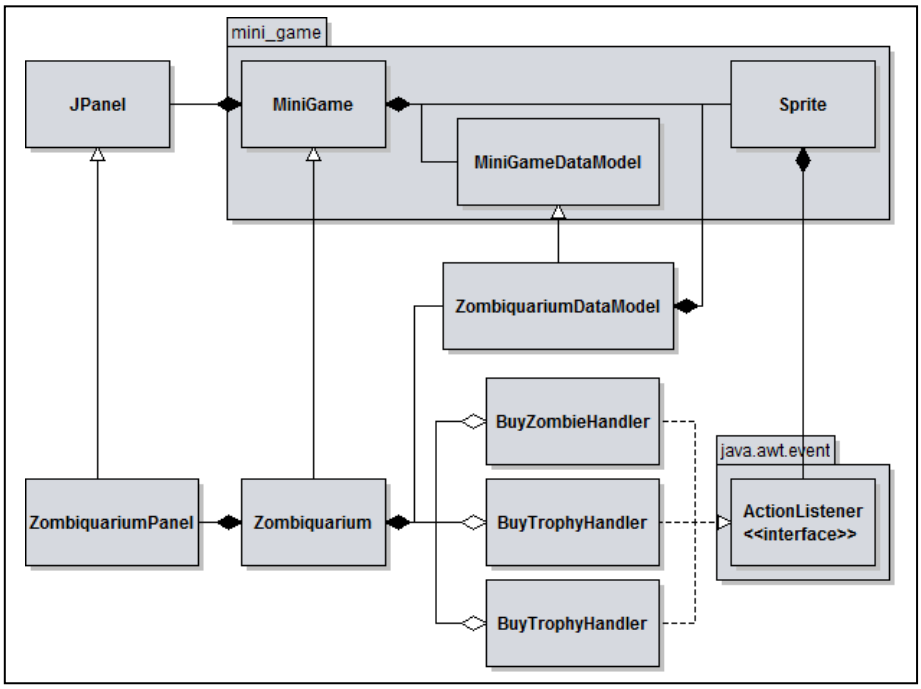

Figure 3.2: Zombiquarium Overview UML Class Diagram

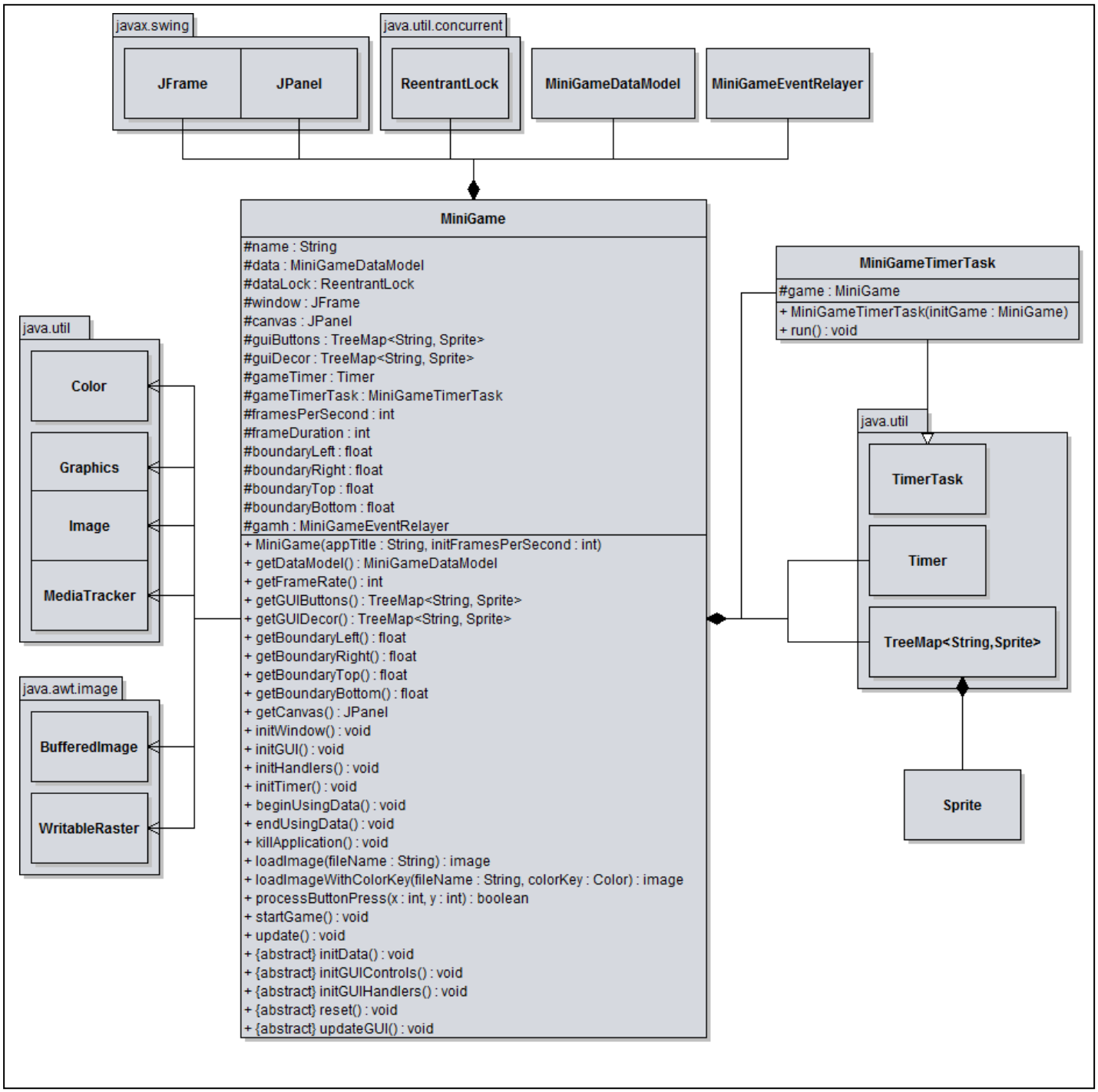

Figure 3.3: Detailed MiniGame and MiniGameTimerTask UML Class Diagram

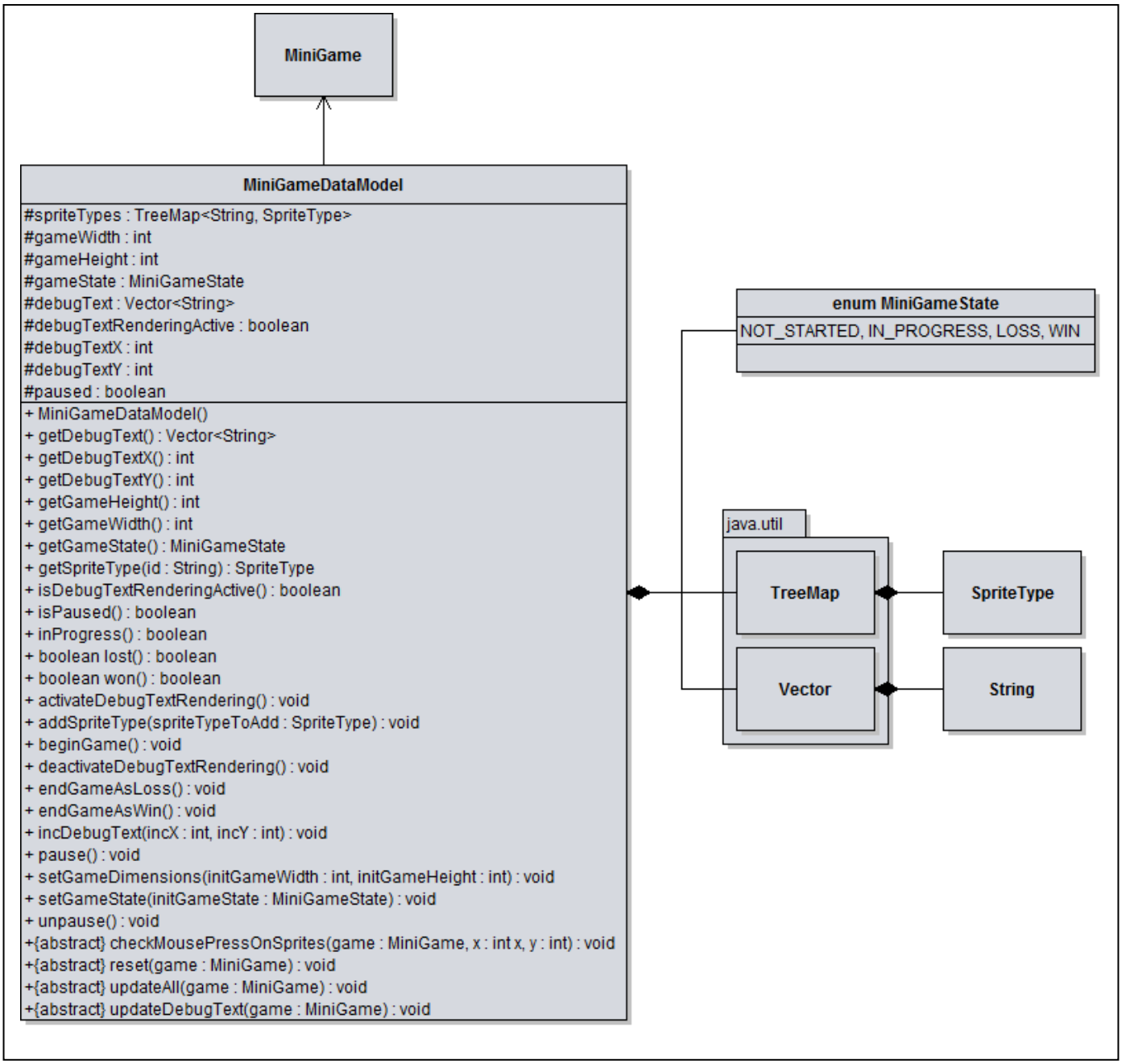

Figure 3.4: Detailed MiniDataModel UML Class Diagram

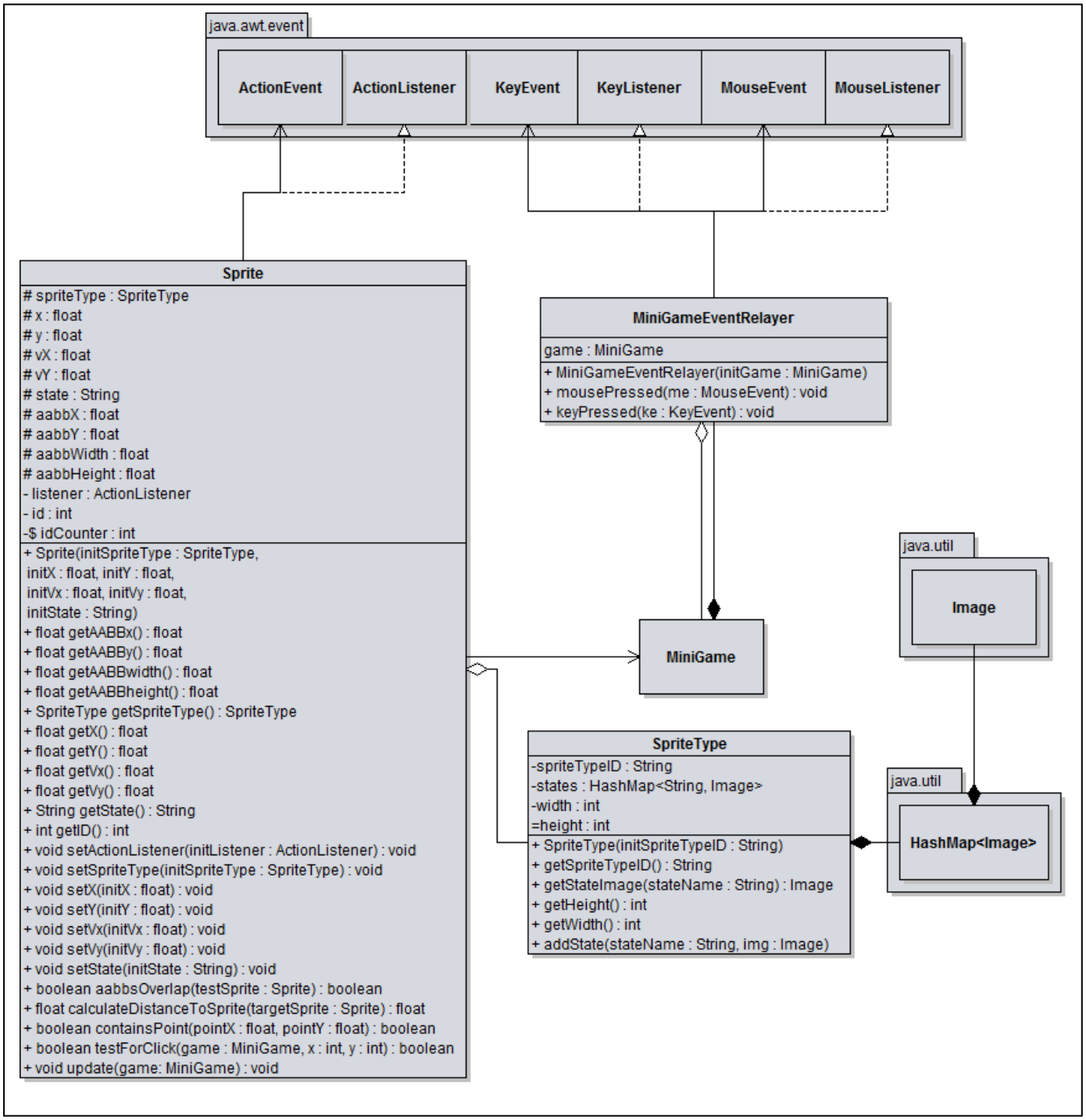

Figure 3.5: Detailed Sprite, Sprite, and MiniGameEventRelayer UML Class Diagram

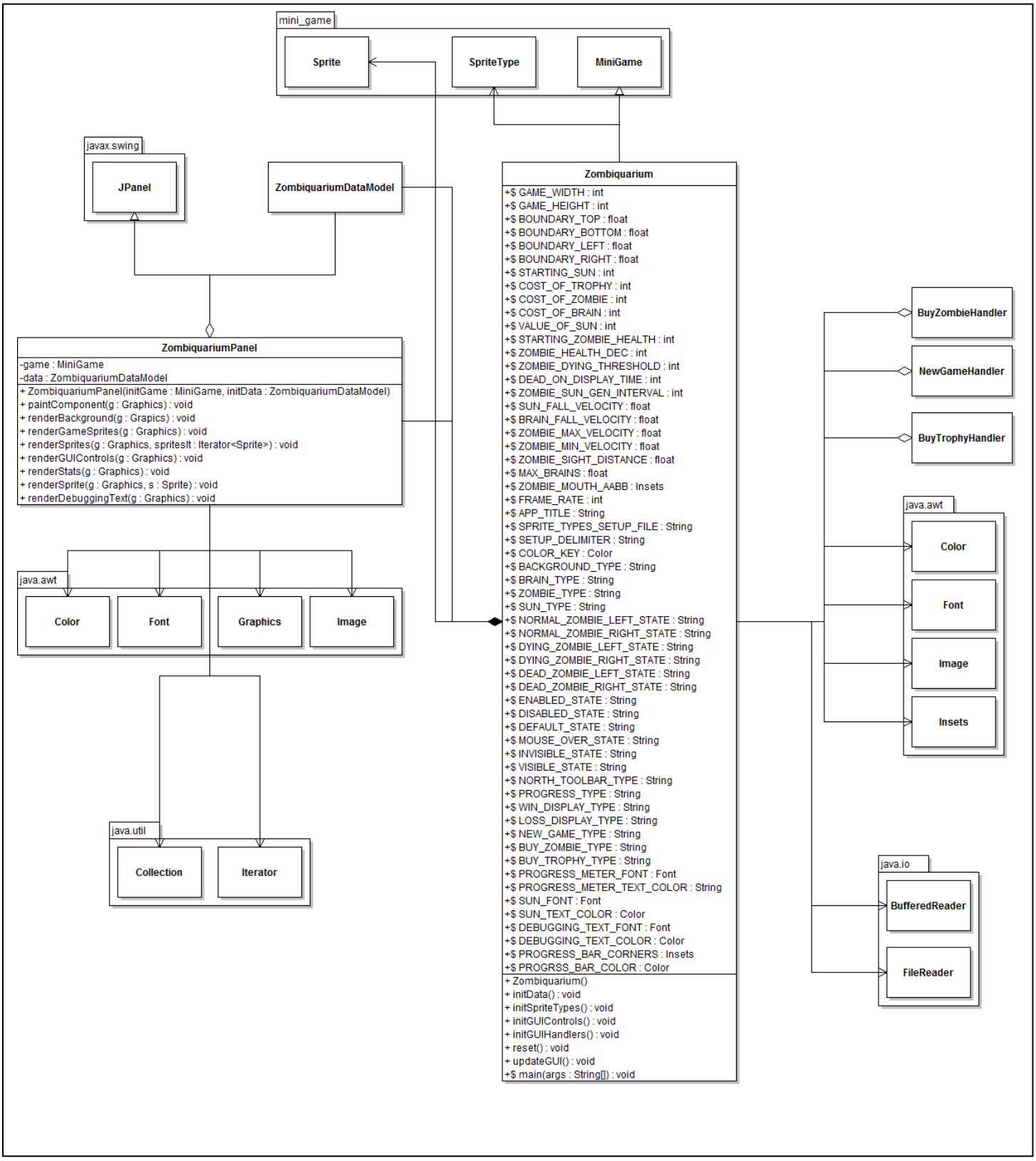

Figure 3.6: Detailed Zombiquarium and ZombiquariumPanel UML Class Diagram

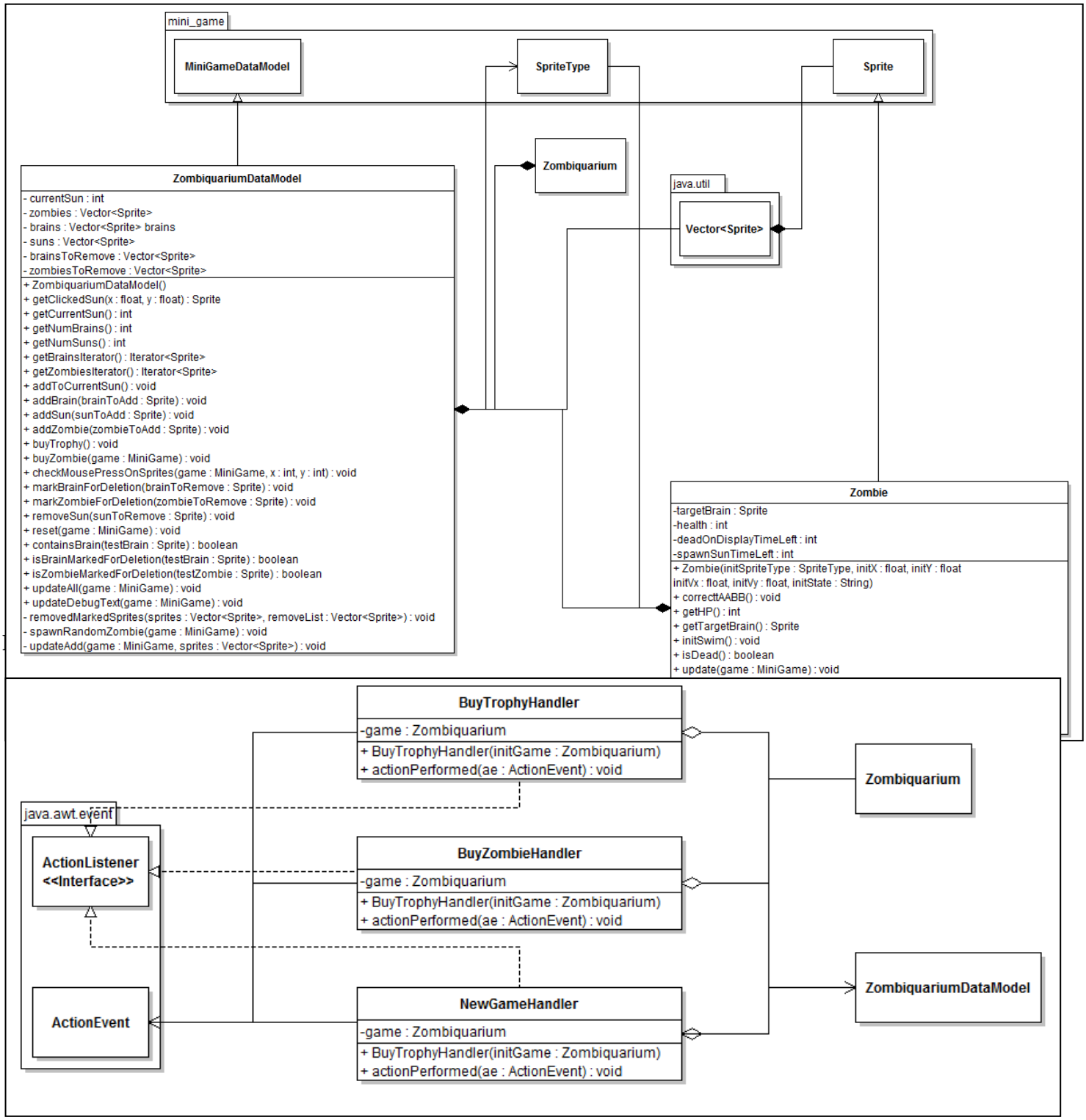

Figure 3.8: Detailed BuyTrophyHandler, BuyZombieHandler, and NewGameHandler UML Class Diagrams

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.

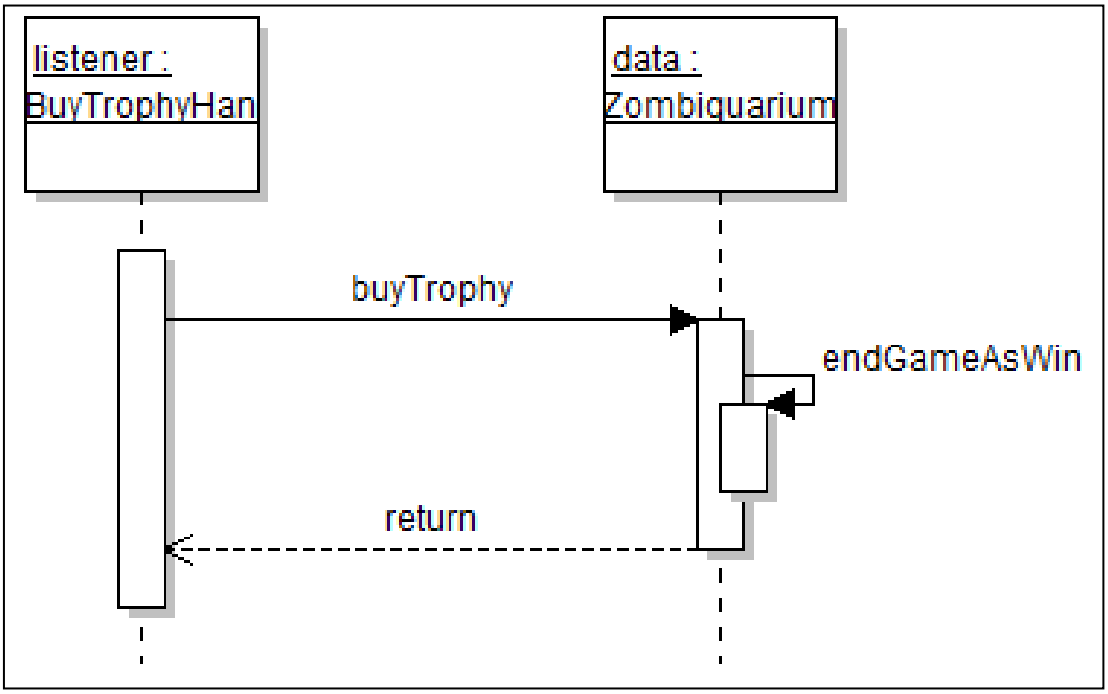

Figure 4.1: BuyTrophyHandler UML Sequence Diagrams

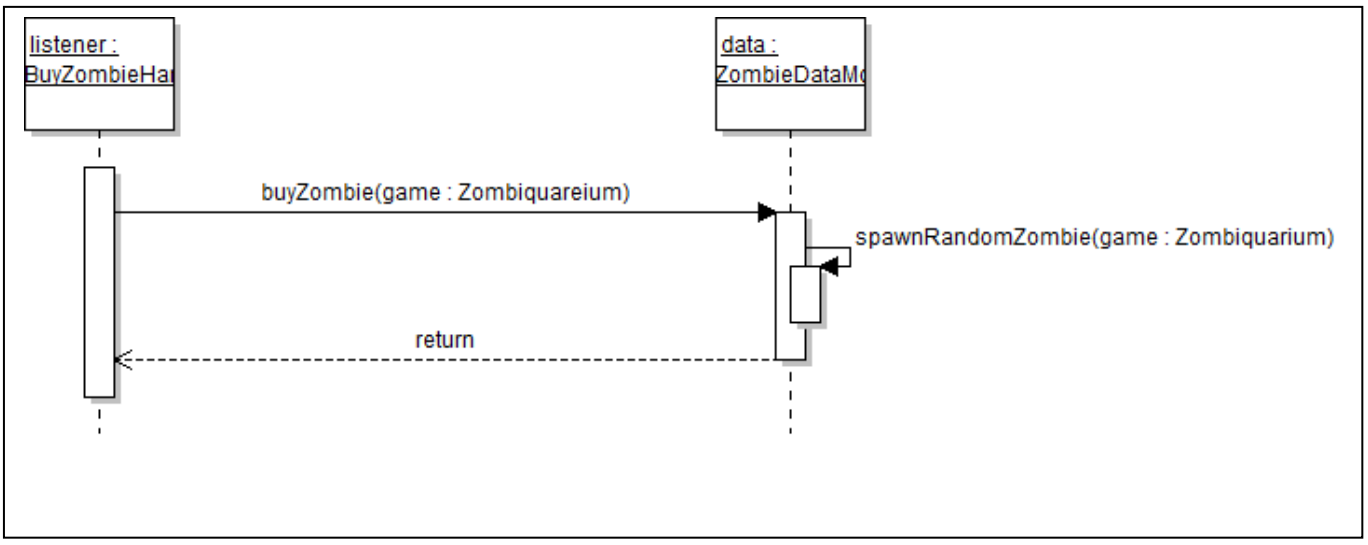

Figure 4.2: BuyZombieHandler UML Sequence Diagrams

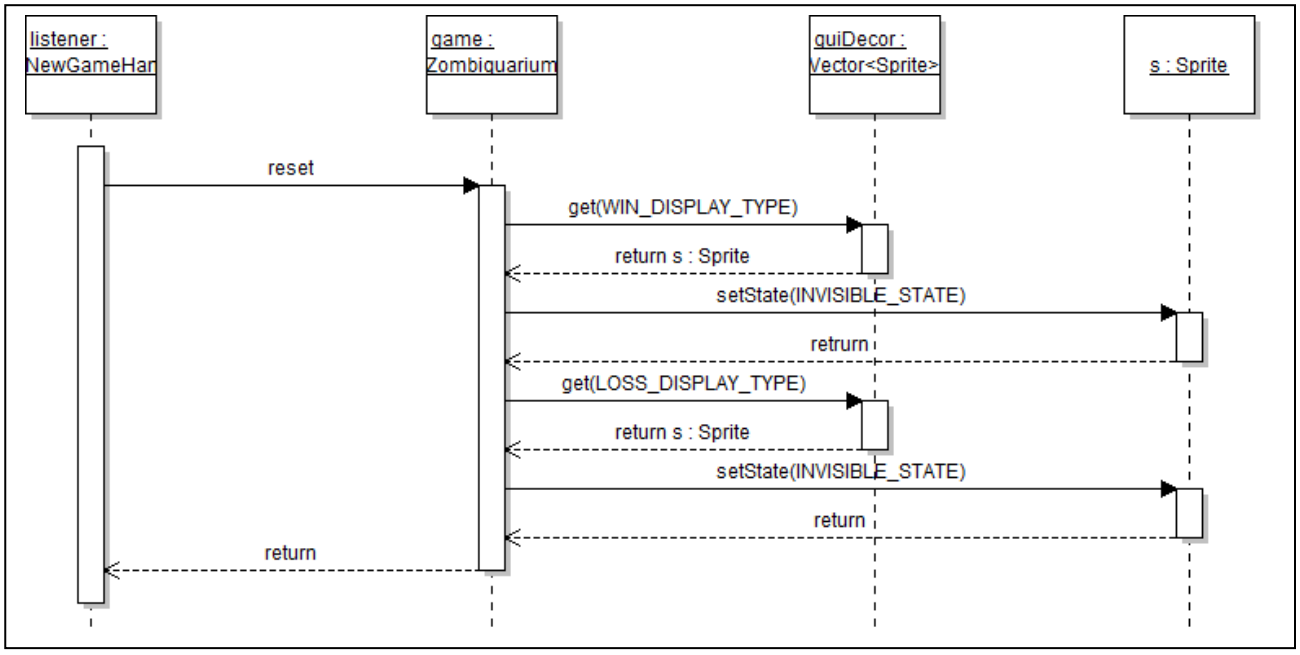

Figure 4.3: NewGameHandler UML Sequence Diagrams

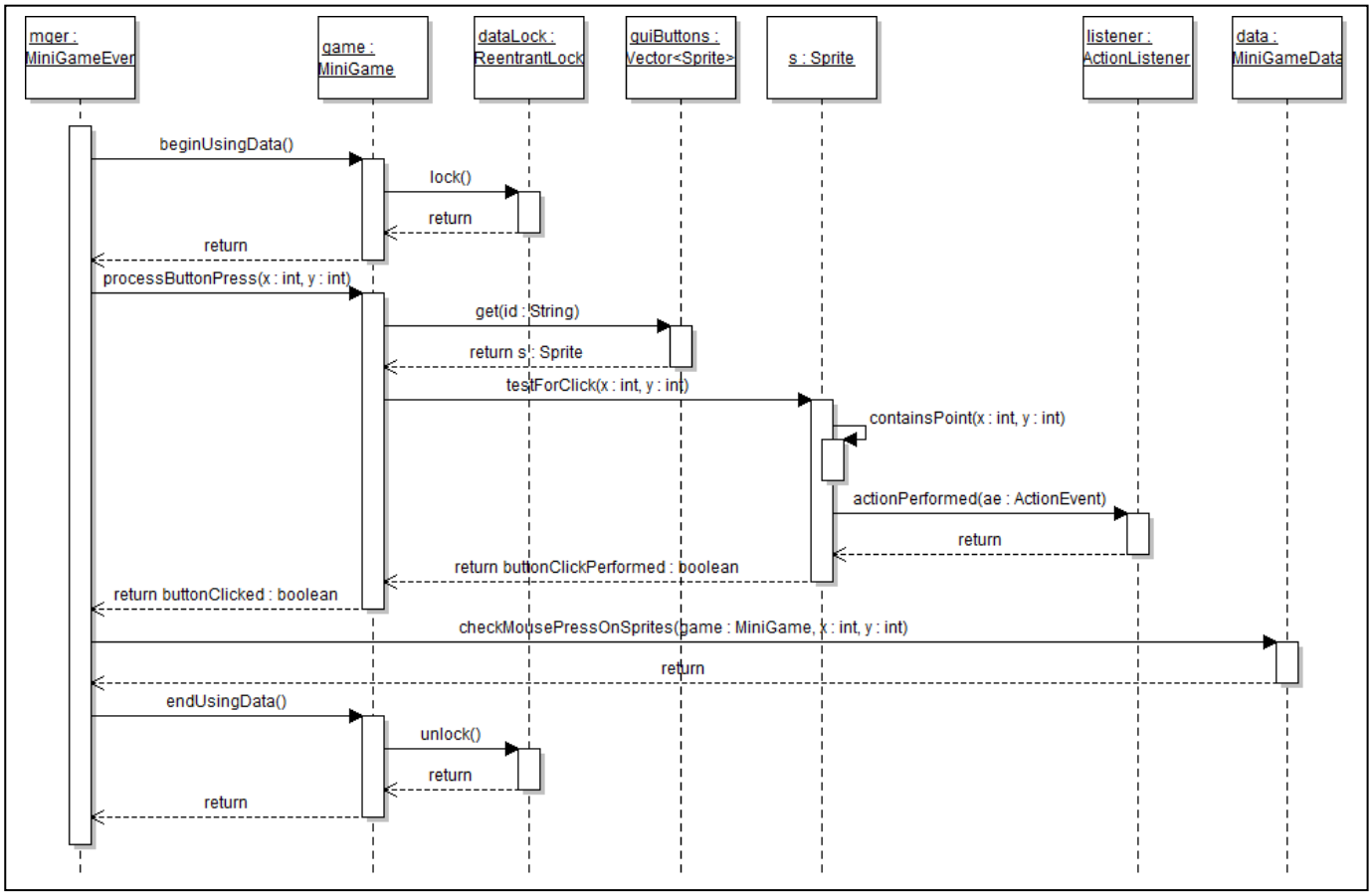

Figure 4.4: Mouse Click on Canvas UML Sequence Diagrams

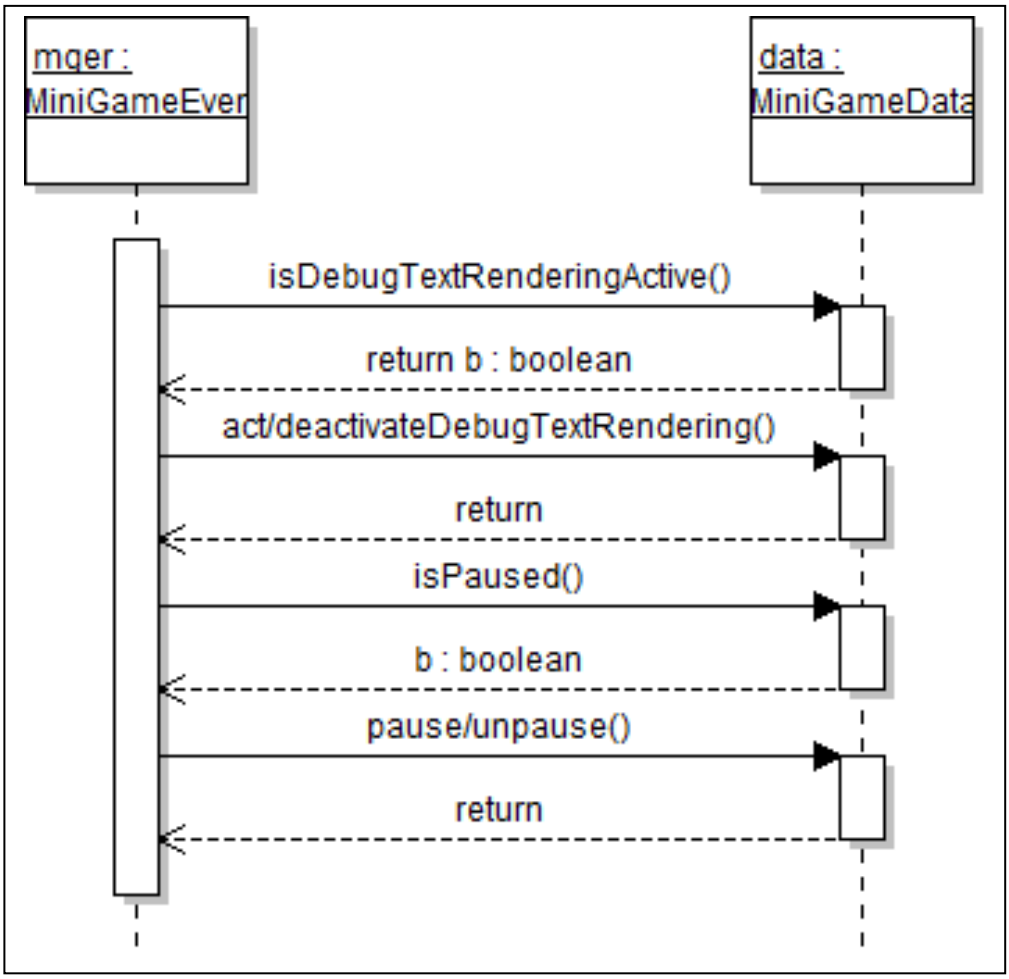

Figure 4.5: Key Press UML Sequence Diagrams

5. File Structure and Formats

Note that the Mini Game Framework will be provided inside MiniGameFramework.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the Zombiquarium application and will be included in the deployment of a single, executable JAR file titled Zombiquarium.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.

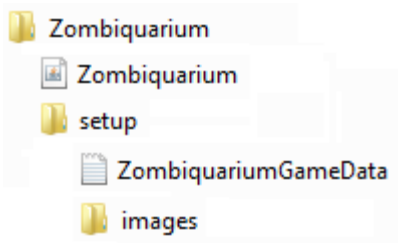

Figure 5.1: Zombiquarium File Structure

The ZombiquariumGameData.txt provides the file and state names for all sprite states in the game. The file is a raw text file that can be used to describe M Sprite Types, each with their own N states as follows:

```
NumSpriteTypes
SpriteType_1|NumStatesFor_1
SpriteType_1STATE_1|SpriteType_1STATE_1_FileNameAndPath
…
SpriteType_1STATE_N1|SpriteType_1STATE_N1_FileNameAndPath
…
SpriteType_M|NumStatesFor_M
SpriteType_MSTATE_1|SpriteType_MSTATE_1_FileNameAndPath
…
SpriteType_MSTATE_N2|SpriteType_MSTATE_N2_FileNameAndPath
```

We can describe these values as follows:

NumSpriteTypes – an integer that lists the total number of Sprite Types to be loaded. Note that this value must correspond to the number of Sprite Types that will be described by the file using the proper format.

SpriteType_X – the textual name of a type of sprite, which should correspond to a Zombiquarium named constant.

NumStatesFor_X – an integer that lists the total number of states for Sprite Type #X. Note that this value must correspond to the number of states for that Sprite Type that will be described by the file using the proper format.

SpriteTye_X_Y – the textual name of a state for sprite X. Note that this value must correspond to a Zombiquarium named constant.

SpriteType_XSTATE_Y_FileNameAndPath – the textual relative path (from the game's home directory) and file name for the image to be used to represent sprite type X when it is in state Y.

6. Supporting Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.

6.1 Table of contents

6.2 Appendixes

N/A