**Protoform:**

**An Action Based Platformer**

**Software Design Description**

**Author:** William Chen

Chenergy IndustriesTM

November, 2017

Version 1.0

**Abstract:** This document describes the program requirements for Protoform, a 2-D action based platformer where the player traverses terrain while avoiding and/or defeating enemies. The program interface, functionality, and overall goals will be described.

**Based on IEEE Std 830TM-1998 (R2009) document format**

Copyright © 2017 Chenergy Industries, which is a made up company and doesn’t own any of the art work used. This project and document is developed for CSE 260 to practice writing a design document, and does not serve any other purpose.

*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 Protoform game application. Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

* 1. **Purpose**

This document will describe the planned build of the Protoform application. UML class diagrams will be used to enumerate in detail packages, classes, instance variables, class variables, and method signatures. UML Sequence diagrams will explain interactions between objects after the application is launched (i.e. in response to user interactions and timed events).

* 1. **Scope**

Protoform will be developed as a stand alone game application. However, different maps and levels may be added onto the base game in the future. As such, terrain and monster generating functions must be robust and reusable. This design will contain descriptions for the generating framework and game framework. 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.

**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.

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

**Platformer** – A genre of video games where the player moves along a map while avoiding obstacles, winning upon reaching a set destination.

**2-D** – 2 dimensions, as in with only an X and Y axis.

**NPC** – Non-player characters with scripted actions.

**Hitbox** – The rectangle that describes the area an entity interacts with. In this case, collisions occur when hitboxes overlap.

* 1. **References**

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

**Protoform SRS** – Chenergy Industries’ Software Requirements Specification for the Protoform game application

* 1. **Overview**

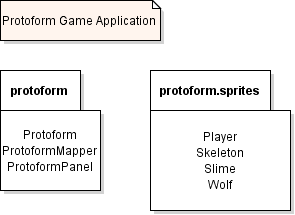
This Software Design Description document provides a working design for the Protoform software application as described in the Protoform Software Requirements Specification. As a standalone project, all classes will be in one package, and image specifications will be in another package (See Figure 2.1).

1. **Package-Level Design Viewpoint**

In building Protoform, the Java and JavaFX frameworks will be heavily used. In this section, uses and components of the Java API will be enumerated.

**2.1 Protoform overview**

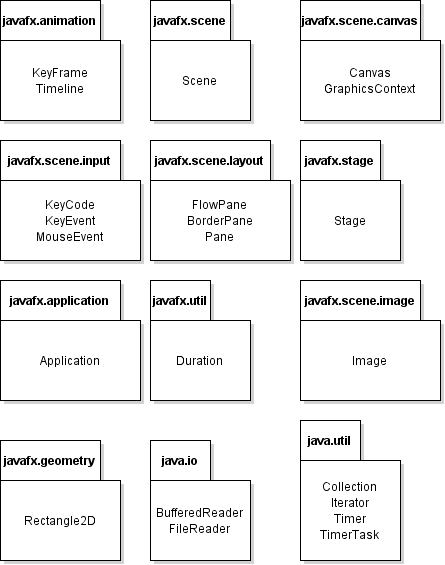
Figure 2.1 specifies all the components organized in packages. Since Protoform is a relatively simple, standalone game, most files with the exception of images will be in the Protoform package.



**Figure 2.1: Design Packages Overview**

**2.2 Java API Usage**

The entire game application will be developed using the Java programming language. The design will use the class specified in Figure 2.2.



**Figure 2.2: Java API Classes and Packages to be Used**

**2.3 Java API Usage Descriptions**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **KeyFrame** | For specifying the states of sprites at certain times. |
| **Timeline** | For maintaining and executing a list of KeyFrames. |

**Table 2.3.1: Uses for classes in the Java API’s javafx.animation package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Scene** | For containing other components of the game. |

**Table 2.3.2: Uses for classes in the Java API’s javafx.scene package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Stage** | For generating the application window. |

**Table 2.3.3: Uses for classes in the Java API’s javafx.stage package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Canvas** | To create a node to draw sprites on. |
| **GraphicsContext** | To issue draw commands to the canvas. |

**Table 2.3.4: Uses for classes in the Java API’s javafx.scene.canvas package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **KeyCode** | For specifying keys on the keyboard. |
| **KeyEvent** | For registering key presses. |
| **MouseEvent** | For registering mouse clicks. |

**Table 2.3.5: Uses for classes in the Java API’s javafx.scene.input package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **FlowPane** | For organizing nodes in a line. Used for end and start screens. |
| **BorderPane** | For organizing nodes along the border of the screen. Used for main gameplay window. |
| **Pane** | Basic node for inserting elements. |

**Table 2.3.6: Uses for classes in the Java API’s javafx.scene.layout package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Application** | Begins the Javafx Application thread. |

**Table 2.3.7: Uses for classes in the Java API’s javafx.application package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Duration** | For specifying lengths of time (use in KeyFrames). |

**Table 2.3.8: Uses for classes in the Java API’s javafx.util package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Image** | For containing and loading images. |

**Table 2.3.9: Uses for classes in the Java API’s javafx.scene.image package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Rectangle2D** | For representing hitboxes. |

**Table 2.3.10: Uses for classes in the Java API’s javafx.geometry package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **BufferedReader** | For reading text files to load maps. |
| **FileReader** | For reading files. |

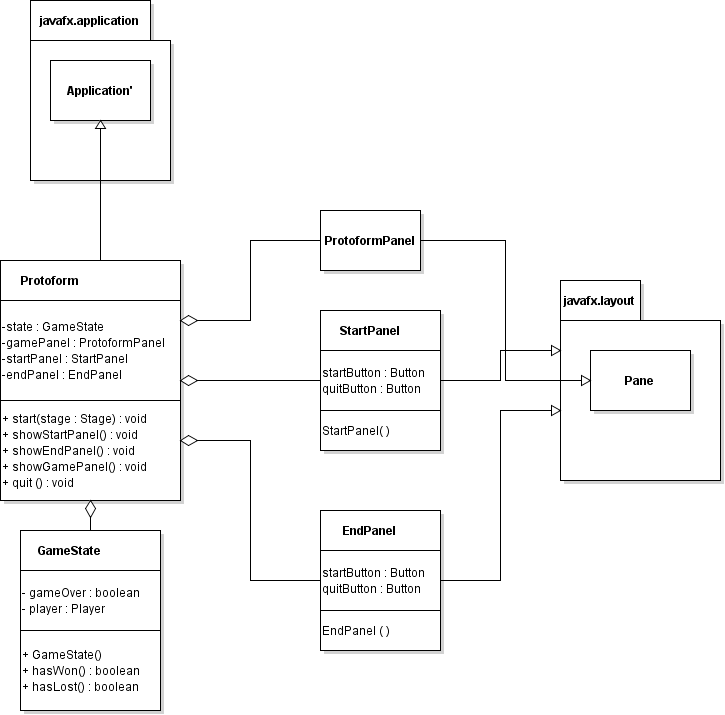
**Table 2.3.11: Uses for classes in the Java API’s java.io package**

|  |  |
| --- | --- |
| **Class/Interface** | **Use** |
| **Collection** | For storing groups of data. Collections will be iterated through for rendering. |
| **Iterator** | For iterating through collections. |
| **Timer** | To execute a task at fixed intervals. |
| **TimerTask** | The custom task to execute – will be extended and implemented. |

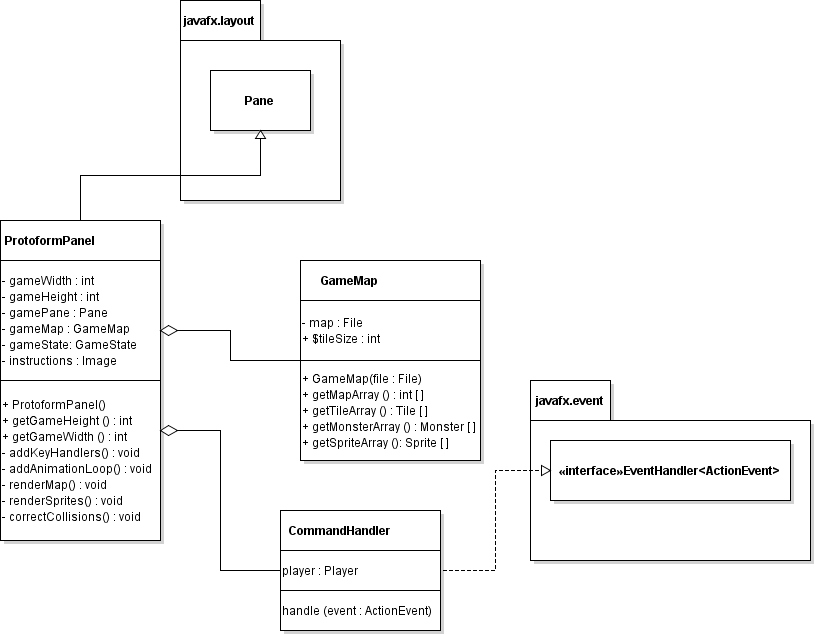
**Table 2.3.12: Uses for classes in the Java API’s javafx.util package**

1. **Class Level Design Viewpoint**

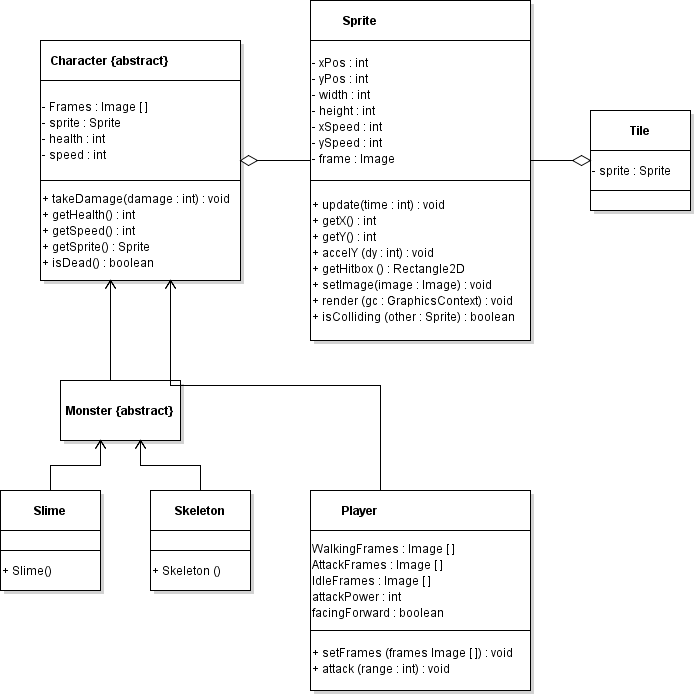
Class design is presented from overview diagrams down to detailed ones.



**Figure 3.1 : Protoform Game Framework Overview**

****

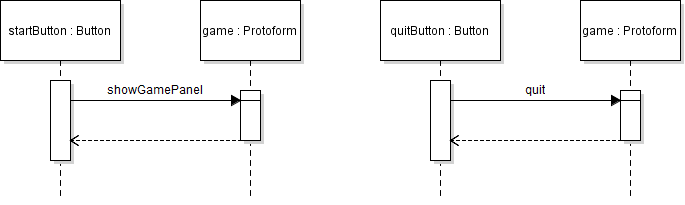
**Figure 3.2: Game Panel Overview**

****

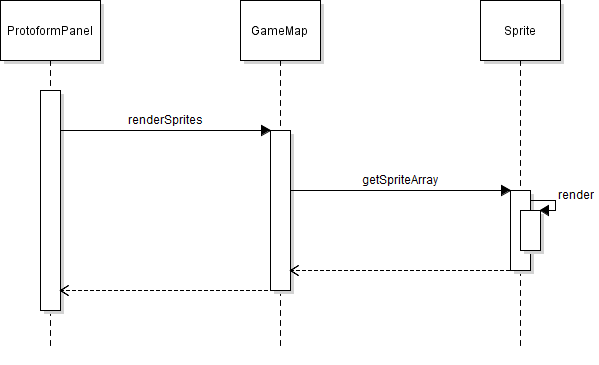
**Figure 3.3: Detailed Sprite Class Diagrams**

1. **Method Level Viewpoint**

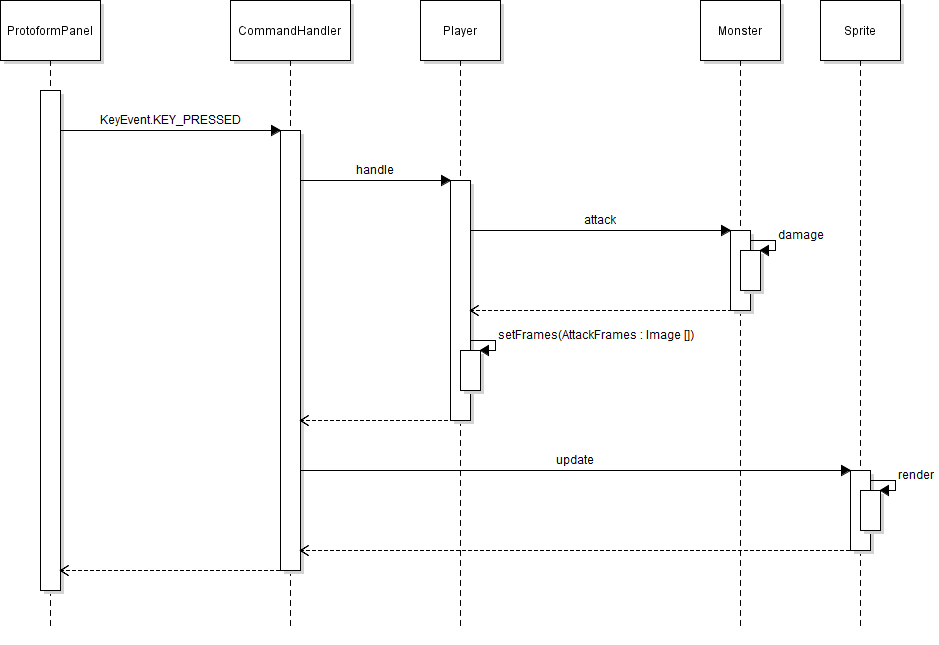
The following UML sequence diagrams will describe how objects interact with one another in methods in response to events.



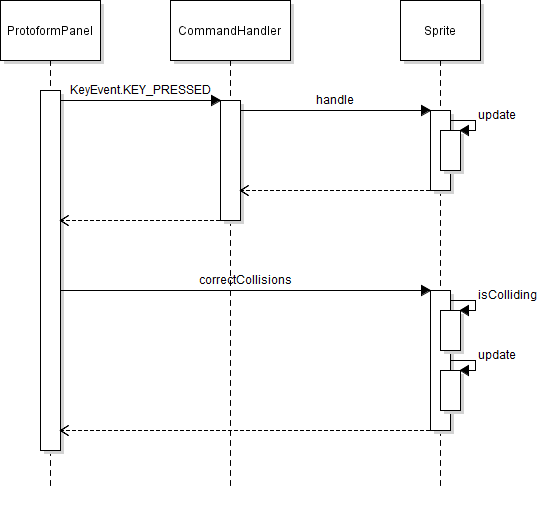
**Figure 4.1: Menu Sequence Diagrams**



**Figure 4.2: Rendering Sequence Diagram**

****

**Figure 4.3: Successful Player Attack Sequence Diagram**

****

**Figure 4.4: Movement Sequence Diagram with Possible Collisions**

1. **File Structure and Formats**

Game files will be organized as follows.

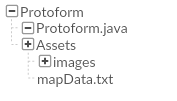


Image files will be stored in the “images” folder. Map data will be stored in a plain text file, with different numbers representing tiles and different types of monsters. All files should be included in the final, executable JAR file titled Protoform.jar.

Map data is encoded in numbers with each object type having a unique ID. For example, the following:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2

0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1

0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0

1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0

This text depicts a staircase with a one tile large enemy at the top right corner (0’s represent empty space, 1’s represent tiles, and 2’s represent a type of enemy). Note that positions are relative, and are based off tile size (with each tile a certain number of pixels wide).

1. **Table of Contents**
2. Introduction 2
   1. Purpose 2
   2. Scope 2
   3. Definitions, acronyms, and abbreviations 2
   4. References 3
   5. Overview 3
3. Package-Level Design Viewpoint 4

1. Game overview 4

2. Java API Usage 4

3. Java API Usage Descriptions 6

1. Class-Level Design Viewpoint 7
2. Method-Level Design Viewpoint 11
3. File Structure and Formats 14
4. Supporting Information (Table of Contents) 14