PLATFORMER GAME: THE VANISHING OF WILL (PGVW)

CS 3337 Software Engineering

Software Requirements, Software Design and Software Test Plan Document

Prepared by:
Bollinger, Bernard
Cisneros, Sasha
Cruz, Chelle
Oceguera, Michael

November 14, 2016

CALIFORNIA STATE UNIVERSITY
LOS ANGELES

Los Angeles, California

PLATFORMER GAME: THE VANISHING OF WILL (PGVW)

Prepared by:

Bollinger, Bernard
Cisneros, Sasha
Cruz, Chelle
Oceguera, Michael

Approved by:

_____	_____
Jose Macias	Date
_____	_____
Richard Cross	Date

Document Change Log

Update	Date Released	Changes
Draft #1	10/20/2016	Delivery of Software Requirements, Design Phase, and SRD documents
Draft #2	11/14/2016	Updated DFD Level 0, added page numbers, added code in Section 5
Final Document	n/a	Last update and upload to CSNS

List of TBD Items

Page	Item	Description	Status

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Purpose

The purpose of this document is **four-fold**:

a) Completely define a full set of requirements for the PGVW – Section 3.0.

b) Completely define the design for the PGVW – Section 4.0.

c) Define and partially implement feasible modules for the PGVW – Section 5.0.

d) Completely define the Test Plan for the PGVW – Section 6.0.

The complete definition of all PGVW requirements provides the source requirement inputs for the development of the subsequent supporting software subsystems documents.

1.2 Scope

The documentation developed as part of this CS 3337 class, starts with the SRD including elements of Software Design and parts of a Test Plan.

The scope of this document includes the following:

- All functional and non-functional requirements on the PGVW are captured. This includes Verification & Validation (V&V) requirements, as well as inter-software subsystems requirements.
- A complete set of PGVW Requirements. These requirements are organized by key PGVW functional units shown on the Level 1 DFD. The Level 1 DFD is shown on page 10.
- A trace matrix, relating all PGVW requirements to functional subunits as expanded in lower level DFDs. Level 2 and higher DFDs are provided on pages.
- The functional requirements defined in the PGVW Requirements section have been expanded to include more specific hardware requirements.

1.2.1 Document Organization

The organization of this document provides a natural flow or allocation of requirements to each succeeding section.

Details regarding the overall document are given in sub-section 1.5 below.

1.2.2 Relationship to Other Documents

The Platformer Game: The Vanishing of Will SRD/SDD/STP/SID is a complete self contained document. Some relationships to other documents in the literature are indicated below in sub-section 1.5.

1.3 PGVW System Architecture

1.3.1 Detailed Context Diagram (DFD Level 0)

The PGVW architecture is summarized in the Context Diagram (DFD Level 0) given below. A more complete Functional Description is given in Section 2 of this document. The Context Diagram provides the overall structure of the software modules and all its inputs and outputs. The notation used corresponds to that defined for any Data Flow Diagram (DFD).

Figure 1.1 Level 0 DFD

1.3.2 Description and major functions of the PGVW

"The Vanishing of Will" is a 2d game based on the hit Netflix original series "Stranger Things". The main controller will start the game in the game menu. From there, the user can choose between four characters from the show and start a new game or load previous progress. After the level is loaded, enemies will be loaded based on the position of the main character on the map. The gameplay map is set in a bird's eye view. Throughout the gameplay, the graphics and audio will be automatically updated based on reactions to the main characters' progress. There will be various levels that the player will face, while experiencing the story plot. The player must find clues in order to find Will. The final level will include a boss that the player must defeat in order to beat the game.

1.4 Documentation Development Process

The PGVW detailed functional description is documented in section 2.0. Basically, Section 2 is a succinct software description document. The overall detailed functional description is based on higher level DFDs (above level 1). All major functional units are described in detail in this part of the document.

In general, all requirements affecting PGVW are captured in Section 3.0 of this document. These requirements are a refinement and completion of requirements first collected as part of this Software

Engineering project. The document is cited in Section 1.2.2. This section is the one worked in most detail to become a reasonably complete Software Requirements Document (SRD). It includes both functional and non-functional software requirements together with several detailed “rational” paragraphs whenever necessary to complete the understanding of each requirement.

Section 4 is the detailed PGVW Software Design Description Document (SDD). This part of the document includes all higher level DFDs as described in section 2 plus all interface units. The document is highly technical and it is based on section 2 descriptions. An important component is the addition of a SIS (software interface specification) document in sub-section 4.2.

Section 5 includes elements of a partial implementation of PGVW. This section includes the various constraints that effectively limit the implementation as well as the sub-units that will be coded. The implementation goals are defined and the code and pseudo code are included as an attachment to this section.

Section 6 is the last major section in this document and includes the overall Test Plan (TP) of the PGVW. The test plan details the various techniques used to test the requirements and it also includes a Validation Matrix where each requirement specified in section 3 is listed with its corresponding validation method. The validation methods may include Testing, Analysis and Demonstration, and possible other V&V methods. In addition, the TP specifies the mandated peer reviews needed to validate the stakeholders part of the requirements.

1.5 References

All references used in the creation of this document are listed below.

1.5.1 Controlling Documents

1) There is no document controlling this document.

1.5.2 Applicable Documents

1) Template provided in CS-3337 was used as basic structure and layout for this document.

2) No additional applicable document has been used in the production of this document.

1.5.3 Standards

No Standard has been used in the creation of this document. However, some Standards described in textbooks have been examined as a reference. In particular, the IEEE standard has been briefly discussed in class.

2.0 DETAILED FUNCTIONAL DESCRIPTION OF THE *PGVW*

2.1 Detailed PGVW Functional Description.

The major tool used to design the PGVW is the Data Flow Diagram, DFD. The rational behind the selection of DFDs as the preferred design tool, was their simplicity and versatility. In the future more sophisticated tools may be used particularly if a correlation from Design to Requirement to Implementation and Testing is found to be a necessary addition.

2.1.1 Higher Level Data Flow Diagrams.

The major tool used to design PGVW is the Data Flow Diagram (DFD). The rationale behind the selection of DFDs as the preferred design tool, was their simplicity and versatility. In the future more sophisticated tools may be used particularly if a correlation from Design to Requirement to Implementation and Testing is found to be a necessary addition.

PGVW major functional design components are shown in the DFDs below:

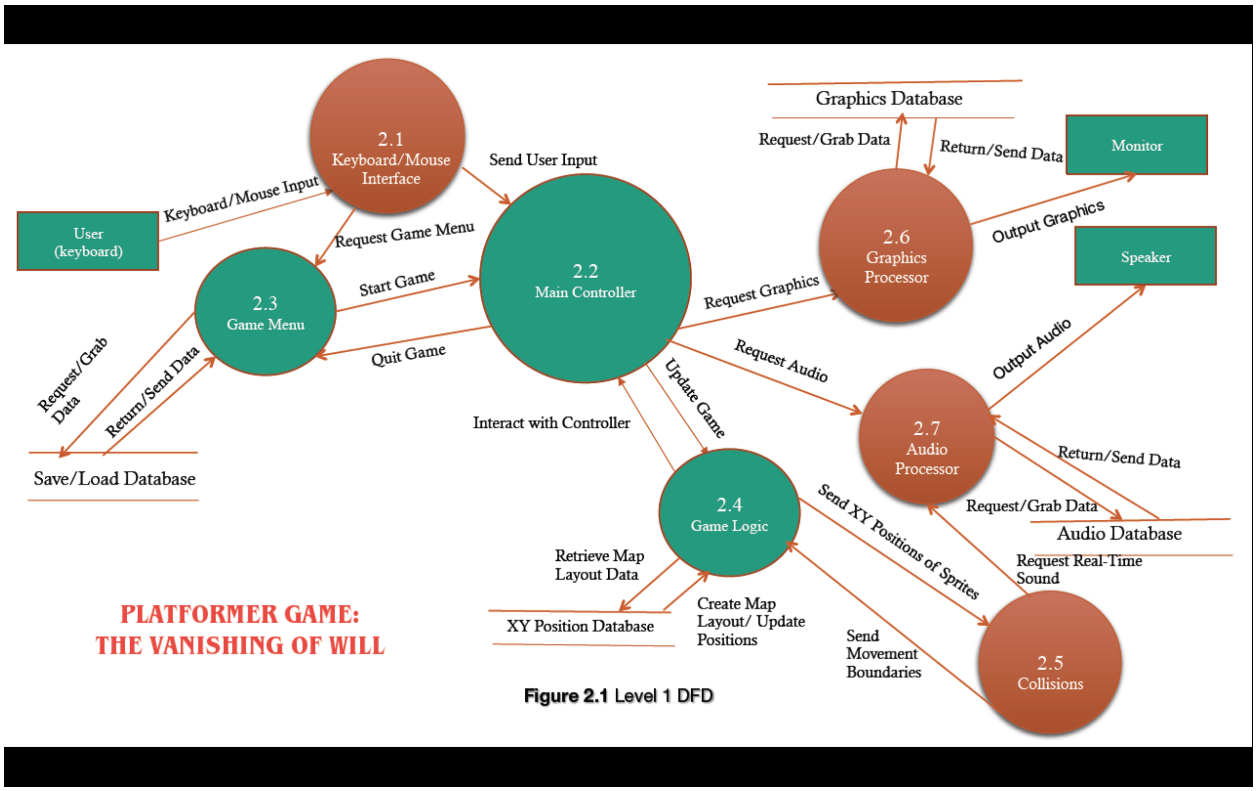

Figure 2.1 Level 1 DFD

2.1.2 Detailed Description of PGVW Major Sub-Units

The PGVW major functional subunits shown in the DFDs in the previous sub-section, are described in detail below:

Keyboard/Mouse Interface - Module 2.1

The Keyboard/Mouse Interface module (KMI) is a layout of instructions that will allow the user to interact with the Main Controller and the rest of the project. These instructions are mapped to certain keys from the keyboard/mouse and will constantly update what the user will be viewing from the monitor.

Main Controller – Module 2.2

(Refer to submodules of Level 2 DFD Main Controller)

The Main Controller module (MCM) receives and stores data regarding the choice of character and current level and level progress from the Game Menu, as well as updating player movement from KMI to Position module. It sends the level progress to the Game Logic so that it can determine x,y coordinates of the player, objects, sprites, and the environment. This information will be sent back through the MCM to the corresponding processing modules(graphics and audio). Throughout the gameplay, the MCM keeps track of the player's progress and continually updates each module it is sending information to.
The module will also monitor the user input from the KMI and determine whether a save or escape sequence is triggered. It also monitors the player's movements and updates the sprite image through the graphics processor, as well as enemies in the game.

The Position module (PM) recieves the starting position of the main character on the screen as an x, y value. The KMI will update this x, y position as the user moves the character with the keyboard arrow keys. The Game Logic Module (GLM) will receive these x, y positions and send them to the collisions module. If any collisions are found, the type of collision will be processed in the GLM and if it restricts the character's movement or affects their health then this information will be sent to the PM. The x, y position will be constantly updated and sent to the Graphics processor to display the character.

The Game Information module (GIM) stores the current game progress for the player such as current level, player health, level difficulty, player inventory of power-ups and weapons, choice of character including attributes and abilities, sprites, and map. It constantly calls on graphics and audio processors to display the elements of the game on screen. It also updates the player's progress on the current level based on the player's position on the map. It also receives the saved data from the game menu when the player starts the game. Whenever the player saves or quits the game, the GIM transfers the data to the Save/Quit module, where it stores the player's overall game progress.

The Save/Quit module (SQM) stores all of the game information in a compressed data file. It also transfers the player back to the game menu when the Esc key is pressed, thus quitting the game. The player's progress will be automatically saved when quitting the game. It also transfers the saved data file to the game menu, which will prompt the user for a saved file name. The game menu module will store the saved data file to the save/load database.

Game Menu – Module 2.3

The Game Menu module (GMM) provides a GUI that will allow a user to decide to play the game and use other features within the project, such as character choice. This will also allow the user to retrieve saved game data from a previous session, if any exists, from the Save/Load database. The Game Menu also receives a saved data file from the SQM after the user enters a saved file name. It then sends the saved data file to the Save/Load database. Once data has been retrieved from the database or a new game is initiated, information is sent to the Main Controller where the game will initiate.

Game Logic – Module 2.4

The Game Logic module (GLM) constantly receives information from MCM throughout the gameplay and decides which objects, maps, and enemies to edit into the game interface. It gathers the positions through a database and sends it to the MCM, where the MCM updates the level progress through the graphics processor. It works with the collisions module to update different parts of the game. Depending on the object, certain types of data such as health will be updated. For example, if the player collides with an enemy or a projectile, the player's health will decrease a certain amount. If a player collides with a power-up object, the player will have enhanced abilities and/or health.

Collisions – Module 2.5

The Collisions module (CM) receives all positions from the player, enemies, objects, sprites, and wall boundaries from the GLM. It checks whether or not the player collides with an object. It then sends that data back to GLM, where it handles the overall level progress.

Graphics Processor – Module 2.6

The graphics processor module (GPM) will send out information to the display based completely on the player's levels and positions throughout the game. The updated information constantly sent to the MCM will be requested by the GPM. From the received data, The Graphics Processor Module will grab the necessary images from the graphics database to be displayed for the user for a smooth gameplay.

Audio Processor – Module 2.7

The Audio Processor module (APM) will cue different audio sounds depending on in-game events, which includes game level and collision audio. On receiving information from the MCM, the APM will locate the correct preset cue from the Audio Database and output the sound through the speakers. The APM will be constantly updating and playing sound.

3.0 PGVW REQUIREMENTS

3.1 PGVW Functional Requirements

This Section collects all PGVW Functional Requirements. The Section includes the complete set of functional requirements with explanation and rational where the statement of the requirement was deemed insufficient or needing additional background/justification. All requirements relate to the design modules described in Section 2. An effort has been made to standardize the correlation between the design modules and the requirements to make their access and organization more consistent. For example, module 2.1 requirements are labeled 3.1, sub-module 2.1.1 requirements are labeled 3.1.1 and so on. The list of requirements follows.

Requirements Related to Design Module 2.1: Keyboard/Mouse Interface	
Requirement No.	Requirement Description
3.1.1	The KMI shall have instructions mapped to the input of the user.

Requirements Related to Design Module 2.2: Main Controller	
Requirement No.	Requirement Description
3.2.1	The MCM shall route data to correct modules.
3.2.2	The MCM shall constantly update information being sent to modules.
3.2.3	The MCM shall contain submodules Position, Game Information, and Save/Quit.
	2.2.2 Position Submodule (PM)
3.2.4	The PM shall receive user input from the KMI.
3.2.5	The PM shall track user position.
3.2.6	The PM shall send user position to GLM.
3.2.7	The PM shall retrieve updated map and enemy boundaries from GLM.
3.2.8	The PM shall send updated information to GIM.
	2.2.1 Game Information Submodule (GIM)
3.2.9	The GIM shall receive updated information from the PM.
3.2.10	The GIM shall receive game information from the GMM.
3.2.11	The GIM shall send game information to the GPM to be processed.
3.2.12	The GIM shall send game information to the APM to be processed.
3.2.13	The GIM shall store user progress within the current game.
3.2.14	The GIM shall send user progress to SQM upon request.
	2.2.3 Save/Quit Submodule (SQM)
3.2.15	The SQM shall create a compressed data file that stores the current game information.
3.2.16	The SQM shall send the compressed data file to the GMM to be stored in the save/load database.

Requirements Related to Design Module 2.3: Game Menu	
Requirement No.	Requirement Description
3.3.1	The GMM shall provide a menu with multiple options.
3.2.2	The GMM shall receive information from the KMI.
3.3.3	The GMM shall request data from the save/load database.
3.3.4	The GMM shall retrieve data from the save/load database.
3.3.5	The GMM shall initiate new data upon request.
3.3.6	The GMM shall send retrieved data to MCM.
3.3.7	The GMM shall retrieve data file from MCM.
3.3.8	The GMM shall send data file to save/load database.

Requirements Related to Design Module 2.4: Game Logic	
Requirement No.	Requirement Description
3.4.1	The GLM shall receive map and enemy positions from the x/y position database.
3.4.2	The GLM shall receive information regarding objects, maps, and enemies throughout gameplay.
3.4.3	The GLM shall send all positions for comparison to CM.
3.4.4	The GLM shall receive collision updates from the CM.
3.4.5	The GLM shall constantly send all requested updated information to the MCM.

Requirements Related to Design Module 2.5: Collisions	
Requirement No.	Requirement Description
3.5.1	The CM shall receive all position data from the GLM on the player, objects, enemies, and wall boundaries throughout the gameplay.
3.5.2	The CM shall compare the positions between two objects to determine if the two collide.
3.5.3	The CM shall send updated collision data to the GLM.

Requirements Related to Design Module 2.6: Graphics Processor	
Requirement No.	Requirement Description
3.6.1	The GPM shall receive updated information on current game's positions and levels from the MCM.
3.6.2	The GPM shall grab necessary images from the graphics database.
3.6.3	The GPM shall send the correct images based on received data from the MCM, to the display.

Requirements Related to Design Module 2.7: Audio Processor	
Requirement No.	Requirement Description
3.7.1	The APM shall receive information from the MCM, upon request.
3.7.2	The APM shall receive information from the CM, upon request.
3.7.3	The APM shall retrieve correct audio files from the audio database.
3.7.4	The APM shall send retrieved files to the speakers.
3.7.5	The APM shall be constantly sending audio to the speakers.

3.2 PGVW Non-Functional Requirements

This Section collects all the PGVW Non-Functional Requirements. All non-functional requirements are numbered “NF – n” where “n” indicates the n^{th} requirement.

NF - 1 PGVW requires sufficient data storage to store files that range from audio and image data.

NF - 2 PGVW requires to run as a JavaFX application (normally through Eclipse IDE).

3.3 PGVW Hardware Requirements

This Section collects all the PGVW Hardware Requirements. All hardware requirements are numbered “H – n” where “n” indicates the n^{th} requirement.

H - 1 PGVW will require keyboard and mouse for controlling the player within the game.

4.0 PGVW DETAILED DESIGN

4.1 Introduction

In this section the PGVW described in Section 2 with requirements listed in Section 3 will be designed in detail possibly including higher level DFDs. Each major module detailed design is included in correspondence with the design sections defined in Section 2 and responding to the requirements listed in its correlated sub-section in chapter 3.

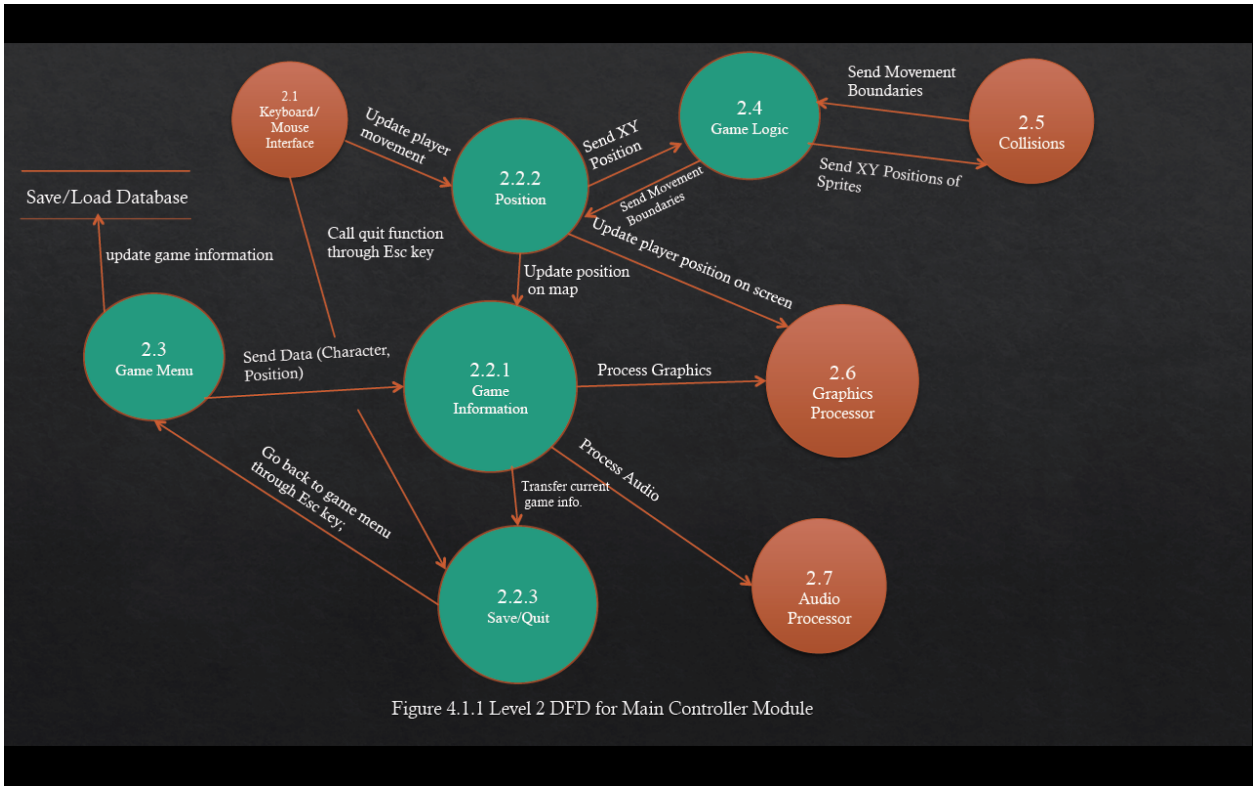

Figure 4.1.1 Level 2 DFD for Main Controller Module

Figure 4.1.1 Level 2 DFD for Main Controller Module

Game Information - Module 2.2.1

The Game Information module (GIM) stores the current game progress for the player such as current level, player health, level difficulty, player inventory of power-ups and weapons, choice of character including attributes and abilities, sprites, and map. It constantly calls on graphics and audio processors to display the elements of the game on screen. It also updates the player's progress on the current level based on the player's position on the map. It also receives the saved data from the game menu when the player starts the game. Whenever the player saves or quits the game, the GIM transfers the data to the Save/Quit module, where it stores the player's overall game progress.

Position - Module 2.2.2

The Position module (PM) recieves the starting position of the main character on the screen as an x, y value. The KMI will update this x, y position as the user moves the character with the keyboard arrow keys. The Game Logic Module (GLM) will receive these x, y positions and send them to the collisions module. If any collisions are found, the type of collision will be processed in the GLM and if it restricts the character's movement or affects their health then this information will be sent to the PM. The x, y position will be constantly updated and sent to the Graphics processor to display the character.

Save/Quit - Module 2.2.3

The Save/Quit module (SQM) stores all of the game information in a compressed data file. It also transfers the player back to the game menu when the Esc key is pressed, thus quitting the game. The player's progress will be automatically saved when quitting the game. It also transfers the saved data file to the game menu, which will prompt the user for a saved file name. The game menu module will store the saved data file to the save/load database.

5.0 PGVW ELEMENTS OF IMPLEMENTATION

5.1 Introduction

In this section (some of) the modules designed in Section 4 with requirements listed in Section 3 will be implemented initially at least at the level of pseudo code. Where possible, actual code will be provided. Each module is implemented in correspondence with the design sections defined in chapter 2 and responding to the requirements listed in its correlated sub-section in chapter 3.

5.2 PGVW Module 2.1: Keyboard/Mouse Interface

```
scene.setOnKeyPressed(e -> {
switch (e.getCode()) {
        case UP:
                //Animates player going up        animation.play();
                break;
        case DOWN:
                //Animates player going down                                        animation.play();

                break;
        case LEFT:
                //Animates player going down
animation.play();
                break;
        case RIGHT:
                //Animates player going right
animation.play();
               break;
        case Z:
                //Animates sword in correct direction of player
               //Checks for collision with enemies
      scene.setOnKeyReleased(e -> {
        switch (e.getCode()) {
        default: FE.setViewport(new Rectangle2D(0, 160, width, height));
        case UP: animation.pause();vpx = -1; vpy =-1;
        case DOWN: animation.pause(); vpx = -1; vpy =-1;
        case RIGHT: animation.pause();vpx = -1; vpy =-1;
        case LEFT: animation.pause();vpx = -1; vpy =-1;
        case Z:
            //Removes weapon from pane
        }
      });
```

5.3 PGVW Module 2.2: Main Controller

- **5.3.1 PGVW Submodule 2.2.1: Game Information**

```
static final int CANVAS_WIDTH = 1250;
static final int CANVAS_HEIGHT = 657;
public static final String TITLE = "Animated Frame Demo";
public int dir = 1; //1 is right, 2 is down, 3 is left, 4 is up
private int width = 56;      // width of the sprite
```

```
    private int height = 56;      // height of the sprite
    public int xloc = 12;
    public int yloc = 320;
    public int vpx = -1;
    public int vpy = -1;
    private int h = 30;
    public int healthY=0;
    public int bossHealth = 30;

    public int currentroom =1;
    public int level = 1;
    public int speed=5;
    public Circle linkcollision = new Circle(xloc+30, yloc+30, 5);
    public ArrayList<ImageView> enemies = new ArrayList<ImageView>();
    public ArrayList<Boolean> enemyHealth = new ArrayList<Boolean>();
    public ArrayList<Animation> enemyAnimation = new ArrayList<Animation>();
    private int start = 0;
    private int playerC = 0;
```

- **5.3.2 PGVW Submodule 2.2.2: Position**
  The position module is built-in to all objects used that are of type or sub-type “Node” in the Java Framework that include methods getx(), gety(), setx(), and sety();
- **5.3.3 PGVW Submodule 2.2.3: Save/Quit**
  (not implemented)

5.4 PGVW Module 2.3 Game Menu

```
//Character Select Option in GM
  characterSelect.setOnAction(e->{
          if(playerC == 0){
               playerC = 1;
               //Sets image to B/W image
               FE.setImage(linkFE2);
               Label bwCharacter = new Label("You've selected Character 2");
               bwCharacter.setStyle("-fx-text-fill: #efefef;");
               border.setTop(bwCharacter);

          }else if(playerC == 1){
               playerC = 0;
               //Sets image back to regular
               FE.setImage(linkFE);
               Label cCharacter = new Label("You've selected Character 1");
               border.setTop(cCharacter);
               cCharacter.setStyle("-fx-text-fill: #efefef;");
          }

       });


//Start Game Option in GM
   startGame.setOnAction(e->{
          if(start == 0){
             primaryStage.close();
            primaryStage.setTitle(TITLE);
            primaryStage.setScene(scene2);
```

```
        primaryStage.show();

        AnimationTimer gameLoop = new AnimationTimer() {

…}
```

5.5 PGVW Module 2.4 Game Logic

```
public void newRoom(Pane pane){
        pane.getChildren().clear();
        if (currentroom ==1){....//add enemies,powerups for room 1 to pane}
        if (currentroom ==2){....//add enemies,powerups for room 2 to pane}
        ...
        //etc
        }
```

5.6 PGVW Module 2.5 Collisions

```
  private boolean checkCollision(Node node1, Node node2) {
  boolean collisionDetected = false;
   if (node1.getBoundsInParent().intersects(node2.getBoundsInParent())) {
      collisionDetected = true;
    }
   else {
       collisionDetected = false;
   }
   return collisionDetected;
}
```

5.7 PGVW Module 2.6 Audio

Audio is implemented for the soundtrack that will play throughout the duration of the game, as soon as the window is opened (the soundtrack is looped throughout). The sound for the Demogorgon is played when the player reaches the final level of the game, where the player faces the Demogorgon boss. The stick sound is played every time the player hits “Z” on the keyboard, which allows the player to use their stick weapon upon enemies.

```
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

 //Music

File themeMusic = new File("stranger-things.mp3");
Media themeTrack = new Media(themeMusic.toURI().toString());
MediaPlayer mediaPlayerTheme = new MediaPlayer(themeTrack);

//Demogorgon sound

File demoSound = new File("Demogorgon.m4a");
Media demoTrack = new Media(demoSound.toURI().toString());
MediaPlayer mediaPlayerDemo = new MediaPlayer(demoTrack);

//Stick sound
```

```
File stickSound = new File("stick-sound.mp3");
Media stickTrack = new Media(stickSound.toURI().toString());
MediaPlayer mediaPlayerStick = new MediaPlayer(stickTrack);
```

(MediaPlayer and Media class are imported from JavaFX and are mainly used to display media on the application.)

5.8 PGVW Module 2.7 Graphics

```
//the graphics module includes the following code used to turn spritesheets into animations,
//as well as built in JavaFX classes and methods used to draw to the screen

package Link1pkg;

import javafx.animation.Interpolator;
import javafx.animation.Transition;
import javafx.geometry.Rectangle2D;
import javafx.scene.image.ImageView;
import javafx.util.Duration;

public class LinkAnimator extends Transition { //spriteAnimation extends part of the Animation class making new objects "Animations"

    private ImageView imageView;
    private final int count;
    private final int columns;
    private int offsetX;
    private int offsetY;
    private final int width;
    private final int height;

    private int lastIndex;

    public LinkAnimator(
            ImageView imageView,
            Duration duration,
            int count,   int columns,
            int offsetX, int offsetY,
            int width,   int height) {
      //instantiates all variables and objects passed in to constructor
        this.imageView = imageView;
        this.count     = count;
        this.columns   = columns;
        this.offsetX   = offsetX;
        this.offsetY   = offsetY;
        this.width     = width;
        this.height    = height;
        setCycleDuration(duration);
        setInterpolator(Interpolator.LINEAR);
    }

    public void setOffset(ImageView I, int x, int y){
      this.imageView = I;
      this.offsetX = x;
```

```
        this.offsetY = y;
    }

    protected void interpolate(double k) {
        final int index = Math.min((int) Math.floor(k * count), count - 1);
        if (index != lastIndex) {//if indexed frame is different than previous animation frame
            final int x = (index % columns) * width  + offsetX;//indicates xval of new frame
            final int y = (index / columns) * height + offsetY;//indicates yval of new frame
            imageView.setViewport(new Rectangle2D(x, y, width, height)); //sets the view of the animation to
a 2D rectangle and x,y location
            lastIndex = index;
        }
    }

} // SpriteAnimation

/* REFERENCES:
 *
 * SpriteAnimation.java borrowed from the following site:
 * Creating a Sprite Animation with JavaFX | Mike's Blog
 * http://blog.netopyr.com/2012/03/09/creating-a-sprite-animation-with-javafx/
 */
```

6.0 PGVW TEST PLAN

6.1 Introduction

In this section the testing methodology to be used to V&V each of the requirements listed in section 3.0 has been identified. At points some additional testing may be required and they shall be documented as an attachment to this document.

The methodologies and testing strategies identified at this point include four major approaches: TESTING, DEMONSTRATION, INSPECTION, and ANALYSIS with various variations to adapt to the PGVW characteristics:

- **Testing** using additional ad-hoc created software including a correlation testing unit.
- **Demonstration** of the specified capability
- **Inspection** of the software code possibly using additional inspection techniques
- **Analysis** of the specific code operation/algorithm to prove functionality.

6.2 FUNCTIONAL REQUIREMENTS VALIDATION MATRIX

The PGVW Functional and Performance Requirements Validation Matrix is given below.

6.2.1 Keyboard/Mouse Interface Module

Requirement No.	Requirement Description	V&V Methodology
3.1.1	The KMI shall have instructions mapped to the input of the user.	Demonstration

6.2.2 Main Controller Module

Requirement No.	Requirement Description	V&V Methodology
3.2.1	The MCM shall route data to correct modules.	Demonstration
3.2.2	The MCM shall constantly update information being sent to modules.	Demonstration
3.2.3	The MCM shall contain submodules Position, Game Information, and Save/Quit.	Testing
	2.2.2 Position Submodule (PM)	
3.2.4	The PM shall receive user input from the KMI.	Demonstration
3.2.5	The PM shall track user position.	Demonstration
3.2.6	The PM shall send user position to GLM.	Demonstration
3.2.7	The PM shall retrieve updated map and enemy boundaries from GLM.	Demonstration
3.2.8	The PM shall send updated information to GIM.	Demonstration
	2.2.1 Game Information Submodule (GIM)	
3.2.9	The GIM shall receive updated information from the PM.	Testing
3.2.10	The GIM shall receive game information from the GMM.	Testing
3.2.11	The GIM shall send game information to the GPM to be processed.	Testing

3.2.12	The GIM shall send game information to the APM to be processed.	Testing

6.2.3 Game Menu Module

Requirement No.	Requirement Description	V&V Methodology
3.3.1	The GMM shall provide a menu with multiple options.	Testing
3.2.2	The GMM shall receive information from the KMI.	Testing
3.3.3	The GMM shall request data from the save/load database.	Testing
3.3.4	The GMM shall retrieve data from the save/load database.	Testing
3.3.5	The GMM shall initiate new data upon request.	Testing
3.3.6	The GMM shall send retrieved data to MCM.	Testing
3.3.7	The GMM shall retrieve data file from MCM.	Testing
3.3.8	The GMM shall send data file to save/load database.	Testing

6.2.4 Game Logic Module

Requirement No.	Requirement Description	V&V Methodology
3.4.1	The GLM shall receive map and enemy positions from the x/y position database.	Demonstration
3.4.2	The GLM shall receive information regarding objects, maps, and enemies throughout gameplay.	Demonstration
3.4.3	The GLM shall send all positions for comparison to CM.	Demonstration
3.4.4	The GLM shall receive collision updates from the CM.	Demonstration
3.4.5	The GLM shall constantly send all requested updated information to the MCM.	Demonstration

6.2.5 Collisions Module

Requirement No.	Requirement Description	V&V Methodology
3.5.1	The CM shall receive all position data from the GLM on the player, objects, enemies, and wall boundaries throughout the gameplay.	Demonstration
3.5.2	The CM shall compare the positions between two objects to determine if the two collide.	Demonstration
3.5.3	The CM shall send updated collision data to the GLM.	Demonstration

6.2.6 Graphics Processor Module

Requirement No.	Requirement Description	V&V Methodology
3.6.1	The GPM shall receive updated information on current game's positions and levels from the MCM.	Analysis
3.6.2	The GPM shall grab necessary images from the graphics database.	Analysis
3.6.3	The GPM shall send the correct images based on received data from the MCM, to the display.	Analysis

6.2.7 Audio Processor Module

Requirement No.	Requirement Description	V&V Methodology
3.7.1	The APM shall receive information from the MCM, upon request.	Testing
3.7.2	The APM shall receive information from the CM, upon request.	Testing
3.7.3	The APM shall retrieve correct audio files from the audio database.	Testing
3.7.4	The APM shall send retrieved files to the speakers.	Testing
3.7.5	The APM shall be constantly sending audio to the speakers.	Testing

A. ACRONYMS

PGVW Platformer Game: The Vanishing of Will

KMI Keyboard/Mouse Interface

MCM Main Controller Module

PM Position Module

GIM Game Information Module

SQM Save/Quit Module

GMM Game Menu Module

GLM Game Logic Module

CM Collisions Module

GPM Graphics Processor Module

APM Audio Processor Module

GUI Graphical User Interface

B. DATA DICTIONARY

Graphical User Interface (GUI): A visual way of interacting with a computer using items such as windows, icons, and menus, used by most modern operating systems.

Sprite: a computer graphic that may be moved on-screen and otherwise manipulated as a single entity; it interacts with other sprites and game objects during gameplay