**Mason Sanders – 437 Final Project Documentation – Gravity Room**

**Game Design**

The game’s Engine class acts as the entry point, makes a JFrame and adds a Scene. The Scene class mainly acts as a control for the Levels. In paintComponent, it passes Graphics2D to the Levels, which in turn pass Graphics2D to the Sprites. Level is an abstract class that keeps track of some basic data including the list of sprites, the level ID, the number of lives/tries that the player has, the number of tokens the player has, and whether or not the level is in a completable state.

There are a total of 5 subclasses of Level within the game. Three of these I consider to be “actual levels.” Which are Level1, Level2, and Level3. The other two are StartScreen and GameOverScreen. Levels 1, 2, and 3 each create a complete set of sprites in their constructors. StartScreen and GameOverScreen do not contain any sprites and instead directly render text on the level itself. StartScreen and GameOverScreen also both implement KeyListener to control transitioning directly from one level to another. The abstract methods nextLevel and resetLevel are implemented in each level to tell the Scene class which level to set the currentLevel property to. nextLevel will go to the next level, for example, startScreen goes to Level1, which goes to Level2 etc. resetLevel just creates a new copy of the same level, meant to be used when the player fails the level.

Sprites in the game consist of the Player, Platform, Life, Token, SpikeTrap, Exit and HUD classes. The Player handles most of the collision in the game, while Token, Life, and Exit have some minor collision detection, which includes Token and Life increasing their counters for the level respectively and setting themselves to uncollectible, while exit tells the Level to move to the next Level if it’s in a completable state.

Sprites will die permanently when Java’s automatic garbage collection deletes the level from memory upon setting Scene’s currentLevel to a new Level. Other than that, they don’t “die” within the level. Tokens and Lives will disappear and become uncollectable when the Player collides with them. And the Exit becomes invisible when the player has collected enough tokens, which in turn means that the level is in a completable state. But since the Exit still needs collision to move to the next Level, it cannot be deleted. The Player, SpikeTraps, HUD, and Platforms do not disappear or die during the level.

**States and State Changes**

The Scene’s only controllable state is its currentLevel property, which gets changed as a result of nextLevel or resetLevel getting called in a Level. Another way the currentLevel property can change is within a Level update method, if the number of lives is less than or equal to 0, the level gets set to a GameOverScreen.

The Level’s main state is the completable property, which when true, tells the Exit to change its behavior so that it disappears and nextLevel can be called when the Player collides with it.

All sprites that actually update their collision (Player, Life, Token, Exit) utilize a HashMap that contains a set of other Sprites colliding with that object, as well as the direction state that object is colliding from, which can be above, below, to the left, or to the right. The player also utilizes the gravity direction as a state, which can also be top, bottom, left or right. The gravity state is changed using the arrow keys, and changes movement behavior accordingly, such as whether the horizontal or vertical state should be used for jumping or moving. Movement states can be changed with the A, D, and Space keys. A moves the player left or up depending on the gravity, and D moves the player right or down depending on the gravity. Spacebar causes the player to be in a jumping state. Movement states for the player can also consist of falling or landed, if the player is falling off of a platform, or if the player is landed on the ground.

Tokens and Lives, as mentioned previously, have a collectible state that tells whether or not numTokens and numLives should be incremented if they collide with the player. The visibility state for these sprites also changes under this condition. Exit uses its visibility property as well, but relies on Level’s completable property for changes in its behavior.

Upon colliding with a SpikeTrap, resetLevel will be called and numLives will decrease. The SpikeTrap has no states that it sets on its own other than its collision type which is set to kill other in its constructor, but the player handles what to do with this collision type.

The HUD is unique in that it just renders text tied to numLives and numTokens. It has no states of its own and is entirely dependent on the state of the Level.

NextLevel in StartScreen and GameOverScreen is called by pressing any key, and they move Scene’s currentLevel to the next level. StartScreen goes to Level1, and GameOverScreen returns to StartScreen.

**Main Levels**

The only moving piece of each level is the player, while other sprites may disappear, they do not change x and y positions. The main point of the Game is to be an obstacle course. Use gravity to avoid the spikes while collecting all of the Tokens to open the exit to the next level. The player can collect one extra life on each level but this is optional.

The main Difference between the levels is the layout, and the number of Tokens you must collect to complete the level. Level 1 requires 1 token, Level 2 requires 2 Tokens, and Level 3 requires 3 tokens. The levels are open ended and the player can take whatever path they like to get to a token, though some paths may be easier than others.

**Software Engineering Plan**

Since this was a solo endeavor, the entire project was made by me, Mason Sanders. The roles I needed to fill were designing the structure and class hierarchy, writing the code, fixing bugs along the way, and creating images for the sprites. The game is written in Java using swing and Graphics2D.

The class hierarchy is as follows: The entry point is the engine class, which extends JFrame and creates a Scene which extends JPanel and implements ActionListener. The scene controls the Level, which is abstract and has StartScreen, Level1, Level2, Level3 and GameOverScreen as subclasses. The level contains a list of abstract Sprites, whose subclasses are Player, HUD, Platform, SpikeTrap, Exit, Token, and Life.

I created the images of the sprites using GIMP. Since I am not a great artist, the image for the player is fairly abstract, just being a purple and white square. And the platforms I just filled in with some shade of gray and added a noise filter on top to make it look like they have some texture. The most interesting part of the artwork was deciding to write 鍵 which means “key” and 生 which means “life” in Japanese on the tokens and lives, though it’s hard to read since they appear very small on the screen.

A known bug that still exists in the game is when the player moves parallel across two connected platforms while still being in a landed state, collision triggers in the direction the player is moving when trying to cross from one platform to the other. In level design, I tried to avoid having platforms connected like this, though on level 3, the bug can be encountered in one area near the top-right.

**Development Timeline**

November 19th – Edited collision from the project 3 engine to use the current HashMap implementation.

November 22nd – Created artwork for the player and the platforms and designed the walls around each level, as well as implemented the changing gravity system for the player.

November 29th – Implemented the Level and Level1 classes and moved from having everything directly on the Scene class.

November 30th – Created the SpikeTrap class and implemented it to reset the player’s position. Created the Token and Life and implemented their behavior.

November 27th – Created the HUD and Exit classes and implemented their behavior, gave Exit temporary behavior since other levels hadn’t been created yet.

November 30th – Created Level2 and Level3 and designed each level’s layout and allowed for going from one level to another.

December 4th – Made it so that levels are completely reset instead of just resetting the players position upon death. Created and implemented StartScreen and GameOverScreen. Finished development.

**State Transition Diagrams**

**Diagram

Description automatically generated**

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**User Instructions**

Download and extract the source from <https://github.iu.edu/masosand/437finalproject>

The game is written in Java 17. Though Java is supposed to be cross platform, the game does not behave properly on Solus Linux, and I am unaware if the outcome would change on other Linux distributions. I’m also unaware if the game works properly on macOS. But everything should work fine on Windows.

I have Included a GravityRoom.jar file that can just be executed to play the game. If for some reason the jar doesn’t work. The game can be recompiled and ran with the following commands in a bash shell:

rm \*.class

javac Enigne.java

java Engine

Instructions for controls and objectives are given on the start screen.