**Summative Project – Coin Wizard**

**DEVELOPER’S GUIDE**

Lemuel C., Pingda L., Kevin T.

**Overview:**

This documentation is about the program structure and key parts of the logic behind the program. The purpose of this is to let developers know how our program functions and works so if they were to extend or edit our program, they would know how it functions. This program is largely run on the TestMap class, an extended JPanel class that runs the mechanics of the game. Other classes are present include the Character, Monster, Projectile, Tile, Item classes, which all draw on the TestMap object. Finally, a MapContainer class was made to hold the TestMap panel and to facilitate the menu operations of the game. The specific functions of each class will be described later in this document.

**Main Structure:**

This program is structured using a MapContainer class (which extends JFrame) the which holds the content panels. The content panels are any four of the following: a TestMap object, a ‘lose’ ImagePanel, a ‘win’ ImagePanel, and a ‘menu’ ImagePanel. The latter three are constructed in the MapContainer class, as they are simply drawn and configured using Swing GUI. It should be noted that the ImagePanel class is an object that simply extends JPanel and simplifies the importing and drawing of images, using Java IO.

**Classes:**

**class TestMap extends JPanel implements KeyListener**

This is the class that contains a majority of code. It contains the input processing (i.e. controls for the character) and loads the maps. It also includes the game’s physics engine as well as the drawing engine. Finally, the AI is controlled here as well.

**Map Loading/Reading**

This is done using the Java NIO, the new IO that comes with Java 7. It contains methods which simplify the copying of files. This is crucial to the game as it alters the .txt files with the writeFile(String f) method, in order to preserve the state of a room. If it edited the main files, the maps would be permanently changed, and would have to be manually written each time. As a result, when the game first loads, it reads in all the files in the /maps directory and copies them to the /temp directory. Any accessing/editing is done to the /temp files, leaving the originals untouched. The maps are changed via the setMap(String f) method, which looks in the /temp directory. The setMap(String f) method contains a variety of set (setTiles, setItems, setMonsters) methods which are used to set the location of tiles, items and monsters, respectively.

**Input Processing**

This is done using KeyListener, through the KeyLegend(KeyEvent e) method. Basically it breaks down the event that was fired (from KeyPressed or KeyReleased) and performs actions accordingly. In order to prevent glitches such as simultaneous key presses, the key listener does not perform any thing on the game. Rather, it toggles Boolean variables on and off, which are then checked when refresh() is called.

**Physics Engine**

The entire program behaves much like a physics simulation, where displacement and velocities are defined by the simple kinematics equations. This is facilitated by a Timer object. Every microsecond, the refresh() method is called. In this method contains various “check” (checkMovement, checkMonsterMovement, checkShoot, etc.) methods and computations to update the locations of the characters, monsters, and projectiles. Collision detection is facilitated in this method, as well as win, lose and other special conditions (such as monster drops and picking up items).

**Drawing Engine**

The drawings are done by overriding the paintComponent (Graphics g) of the JPanel class. In this method, various draw(Graphics g) methods of the other classes are called in order to draw everything.

**AI**

This game features simple AI that chases the character by comparing the differences in position and moving in a velocity that would reduce said difference. Due to its simplicity it has no path finding abilities. To balance this, monsters are spawned in hordes, making the game slightly more challenging.

**class Character**

This is the class that contains the character’s properties, including health, number of coins, etc. The majority of methods in this class are getters and setters, which are controlled by the TestMap class. The projectile control methods are also in this class, but are controlled by TestMap as well. The character class contains various Timers to delay its shooting speed and its damage taken speed. That way, you can’t shoot wildly nor can you die in an instant.

**class Monster extends Character**

This is a subclass of the Character class. The only differences between the two are that the Monster do not shoot, and they move with different speed and have different health. However, since they are a subclass of Character, they are theoretically able to shoot. Due to time constraints this was not implemented.

**class Projectile**

Objects of this class are created with initial values for position and velocity. It includes various getters and setters to alter their position as dictated by the timer in the TestMap class, as well as the Image object that represents it.

**abstract class Tile**

Subclasses of this class create objects that display various different images depending on the preloaded one through the draw(Graphics g). An important property that they have is the Boolean variable isPassable, which is used to detect whether a tile can be walked through or not.

**abstract class Item**

Subclasses of this class create objects that also display various images on the Tiles. In fact, each Tile object contains an Item object. The draw(Graphics g) method of this class is called by the Tile’s draw(Graphics g) method. Various Items in this game include coins, health packs, and doors (which are not actually picked up, but facilitate room traversal detection).

**Miscellaneous:**

An important method that was used in most of these classes was the loadImage(String str) method. This was very important as each individual object had its own image (or images) that were needed to show up in the game. Even the menu, winning, and losing backgrounds are Images obtained using loadImage. Due to the heavy use of Images in this program, an extended version of JPanel called ImagePanel was created, as previously mentioned. The implementation of these images led to the creation of the BlankPanel class was made to make implementing transparent JPanels (which was used in GUI formatting) more code efficient.

**Suggestions for improvements:**

Our program has a few areas which could be improved. The main one would be in content variety. There is only one playable character, one type of monster, and two types of pickups. Further development can involve creating more of these. The characters could have varying stats, such as more damage but less health. The monsters can have varying attributes such as health, flying (passing through walls) and being able to shoot projectiles. A smarter AI can be implemented as well. Other pickup suggestions could be speed, max health, damage, or firing rate boosts. New tiles, such as spikes (which would damage the player when they walked on it) could be implemented as well. More improvements could include more game mechanics such as finishing a level by defeating a boss, or having a shop to spend your coins. Non-game related improvements could include having sounds when shooting/killing monsters, having an options menu where you could vary difficulty or change the size of the playing window.