Joe Pimble

Rachel Buttimer

Maze Game/Maze Runner

GitHub Link: <https://github.com/rachelbuttimer/maze-game>

Motivations:

The motivation behind this project stems from wanting to create something enjoyable for the user and something challenging for us. We believe a game is the best way to achieve this. Mazes are extremely popular and common games, so to be able to make a game like those ourselves is a good task. It is also something that needs thought and attention to do. It provides the user with a nice balance of fun and thought that anyone would enjoy. It also provides us with a significant challenge because we are using a new software known as LibGDX. Neither of us have used this software, so there was a huge learning curve to overcome. The aspects of being challenged and creating something fun is how we decided on making a maze game.

Summary/Details of Code:

Our code runs through a java game software called LibGDX.

Our code can be split into 4 sections: Launchers, Screens, Objects, and Collision.

Section 1: Launchers:

The Launchers are defaulted by LibGDX and are what allow the game to run. The desktop launcher is designed for desktops and allows you to set screen size, frame rate, and the title. The game launcher (titled MazeGame) is what starts the game itself. It initializes the screen, rendering, and disposing for the game.

Section 2: Screens:

The screens we made implement the Screen interface from LibGDX. The main menu screen and win screen are similar with buttons and text. The buttons become red when the mouse hovers over them. They allow the user to click onto other screens or exit the game. The Win Screen also displays the time it takes to complete the maze. The maze game screen is the most complicated screen. It holds the code for the user-moved runner, the wall and coin objects, the timer,, the coin counter and the collisions between the objects. This screen is the main workhorse of the project.

Section 3: Objects:

There are two objects, the wall, and the coin. They both have their own methods which are basic. The walls function as borders that the player cannot pass through. The coins function as collectibles that the user can pick up and are needed to beat the game.

Section 4: Collision:

This is used to ensure that the user interacts correctly with the objects. It checks if the player collides with any objects. It allows the main game to interact correctly between the objects if they have collided.

Explanation of Code:

Section 1: Launchers:

Desktop Launcher:

The only code changed and worth mentioning are the configs. The things changed include, the screen size to 1000 by 800 pixels, the game in 60 frames per second, and the game was titled “MazeGame”.

Maze Game Launcher:

Spritebatch is included so that the game batches work correctly. The launcher is also used to set the screen to load up the main menu when booted up. It also has been set up to render and dispose correctly.

Section 2: Screens:

General Information:

All screens use Maze Game Launcher to render batches and have it in their constructors. All screens also use shapeRenderer to render the rectangles. Rectangle is the specific shape used and allows you to customize a rectangle. BitmapFont is simply the font for LibGDX, and it was changed it to a specific font. The specific font is in our GitHub project. GlyphLayout is used for every font to create the text and their positions. The font itself can also be used to set size and the color. Everything font related is in the render section under game.batch. Everything previously mentioned is initialized for each screen with the appropriate sizes, positions, fonts, and color. Each screen also has a set background color.

How Buttons Work:

Each button uses Gdx.*input*.setInputProcessor(new InputAdapter() to set up clicking onto buttons. This can be found in the constructors. It tracks where the position of the mouse is relative to the rectangles. It also sends the user to the specified destination when the mouse is clicked down on the rectangle. The buttons change from white to red and work the same way. Whenever the mouse is hovering over a button, it turns red. Both of these functions can be found in the render section under shapeRenderer.

Main Menu Screen:

All features are included above. Specifically, the general information and the button information. The buttons allow you to enter the main game screen, and also allow you to close the application. There is Start and Exit text over the buttons. There’s an additional text that reads “Maze Runner”. It works the same way as the rest of the fonts, it just can’t be clicked.

Main Maze Screen:

Includes general information. Outside of that, the runner is initialized here with all its attributes. This code also initializes score and time. DeltaTime tracks time in game and it works as a timer for the maze. There are ArrayLists which are initialized here for both wall and coin objects (objects explained later) for ease of creating. CollisionRect is for collision between runner and objects (details explained later). Runner has a texture and rectangle, along with default positions. The “prev” positions are used for collision. Essentially, when the runner hits a wall, it will be sent back to where it was, causing it to be stopped by the wall. This section also has the creation for all the walls, coins, and the font used. Under render, we have the runner’s movements. The runner moves at a specified speed in a direction based off of the inputs of the user. The allowed user inputs in this section are W,A,S,D or Up, Down, Left, Right. While not able to be reached, there are borders for the edge of the screen itself so users can’t travel offscreen. There are for each loops for walls and coins to check for collision. As previously mentioned, if the user runs into a wall, they are sent back to their previous position. For coins, when the user collides with this object, it disappears, and the counter at the top of the screen raises by one. As an additional feature, when 5 coins have been picked up, a wall is removed, and the final coin can be retrieved, sending you to the win screen, which also saves deltaTime. Additionally, deltaTime is printed and constantly updates on-screen. The for each loops in shapeRenderer and game.batch simply load the walls and coins. The fonts work as previously mentioned.

Win Screen:

Uses general and button information. Additionally, it retrieves deltaTime. This is sent to a decimal converter (2 decimal places) to be printed on screen. A “You Win!” message is showed on the screen along with the time taken to complete the maze. The retry button allows you to retry and go back to the main game screen, the main game button sends you to the main menu screen. Both retry and main menu are fonts place on top of the buttons

Section 3: Objects:

P.S. While Coin and Wall are technically classes, but we refer to them as objects

Both coin and wall use collision and have their own rectangles for collision. They also have methods to get position and size, along with checking collisions. These are used to load the objects along with checking collisions with the runner. Both are used as objects so as to not have to initialize every single one. They are sent into a respective ArrayList and are run through a for each loop to check the state and render all of them.

Coin Class/Object:

Each coin has a locked width and height. Each coin also uses a texture, which is a image of a coin. This image is in the GitHub. Calling coin allows you to set the position of each coin. The render method is used to render the coin image for the main game.

Wall Class/Object:

The wall object allows for customizing the position, width, and height. Calling wall allows you to change all these attributes.

Section 4: Collision:

CollisionRect:

This takes in position, width, and height. The move method is specifically for runner. This method matches position of runner so the visual position of the runner matches the rectangle collision of the runner. CollidesWith returns a true or false. It takes in the parameter of an object (runner, coin, or wall). It checks if any piece of one object overlaps with another object. This method is only used for runner with wall and runner with coin. If the method returns true, it means the runner is overlapping with either the wall or coin. As a result, the specified actions for these objects will occur when this happens.

Performance:

The game/application runs perfectly. The buttons work exactly as intended with no issues. The runner’s movement is also perfect and goes exactly where the user directs. The runner interacts with the coins and walls as intended. The coin counter and timer work perfectly too. Every screen displays exactly what it is supposed to display, and nothing works against each other.

Summary:

Overall, this project was challenging and involved a lot of learning. We went into a completely new field of coding and made the best out of it. The project turned out really well and works how we intended. Luckily, we were able to accomplish most of our goals. We might have been a bit ambitious in the amount of content we hoped to make, but we still made great progress. This project was fun and we’re proud of what we were able to do in the given timeframe.