Project Documentation

The basis of our project was that we created a game called Helicopter. The game includes a helicopter that moves in the positive x direction with a negative gravity. When the user clicks on the screen, a positive y velocity is added to the helicopter so that the user can control its position. The objective of the game is to avoid the bombs set up on the map and collect gas items also presented on the map. When the user moves past a bomb, it gains five points. When the user collects a gas, it gains 50 points. When the user hits a bomb, it resets from the beginning with a score of 0.

The way that we organized the project is that we first started off by created a project using the LibGDX Project Generator. We created a project file that included an android, core, and desktop folder. All of our code goes into the code>java>com.project. Inside our com.project we had 4 packages called game, idea, sprites, and states. We start off in the game package and we create a class Helicopter that creates the SpriteBatch, GameStateManager, and Texture. The SpriteBatch includes all the sprites that will be displayed on the screen. The GameStateManager is all the different states that will be included in the game.

When started off by creating the different states of the game in the States package. We go into the GameStateManager to create a stack of the different states that we will be using. We then created our first state, the MenuState, which extends to the stack State. Inside the MenuState we added a background image and a play button to the constructer. We included an update function which calls the handleInput function and a dispose function to dispose of the background and playbutton. We rendered everything onto the screen and we set the handleInput function to enter the PlayState if any part of the screen is touched.

Our PlayState starts off by extending to the GameStateMangaer. We first initialized all the variables we needed and then added the constructor which sets the background and ground for the state. It also initializes the Bird object which is what the user will be controlling. Additionally, we set the camera to follow the object so the user can see it at all times. In our PlayState we also have the handleInput function we calls the jump function from the Bird class whenever the screen is touched. PlayState also includes an update function which updates PlayScreen as time goes on. It updates the helicopter, bombs, and what to do when the user collides with a bomb. There is a separate function to update the ground called updateGround. Additionally, the PlayState has a render method which displays ground, helicopter, bomb, and coins. There is a dispose function to dispose of everything when it is done.

Inside the state package we also have the GameOverState. The GameOverState sets the images for the GameOverState when the user has died. If the screen is touched, the game goes back into the PlayState and restarts. The GameOverState also outputs your score on the screen.

The next package to talk about is the spirites package. The spirites package contains all the objects called Bomb, Bird, and Coin. The first package Bomb creates the bombs included in the PlayScreen. We first do this by importing a bomb image and setting the position of the bombs on the screen. In this class we also include the bounds of the bomb. The bounds of the bomb are necessary because we need to know when the helicopter collides with the bomb. We created a Boolean function called collides which returns true when the user overlaps with the bounds of the bomb. The Bomb class also includes a reposition function which creates the position of the new bombs that need to be added to the screen. This function is called in the PlayState class. There are four other functions to return the object bombs and position of each bomb so they could be accessed in the PlayState.

The next class included in the package spirites is the Helicopter class. The helicopter class creates the helicopter object. It sets the position of the helicopter and also creates the bounds of the helicopter so that we know when it collides with other objects. The update function in the Helicopter class updates the movement of the helicopter. It moves in the positive x direction and makes sure the bounds change as the position of the helicopter changes. There are two other functions to return the bird and position so they could be accessed in the PlayState. Similar to the other classes, there is a dispose function to delete everything so there is no overriding memory. The Helicopter class also includes the jump method which changes the y velocity so the helicopter moves up when the jump function is called in handleInput in PlayState.

Another class in the spirites package is the Coin class. The coin class uploads the gas tank image and sets its position around the screen. It randomly positions itself similar to the bomb images. It includes a collide function that returns true when the helicopter collides with the gas tank. If it collides with the helicopter, the score increases by 50 points.

Lastly, we have an Animation class in the spirites package. The Animation class basically animates the helicopter. It creates an array of frames for each different frame of the helicopter. It has an update function which updates the frames as time goes on. It also has a getFrame method to return the frames. We then call the animation in the helicopter class so that the helicopter is animated.

Over the course of creating the game we encountered many challenges. The first challenge was the animation. The big problem with it was getting the right frames for the image. We tried changing up the code to adjust to the image but we kept on getting the wrong frames. The way we solve this problem was that we adjusted the image using editing software. We now knew the amount of pixels that were contained in the image and where to set the exact positions. After doing this, we ran into another error with the amount of frames we needed. We realized that we needed to initialize the amount of frames that we were going to use, which was 3, for our Animation function. The Animation would cycle through 3 frames, each different images of the helicopter.

Another issue that we had was generating an algorithm to position the bombs. In our Bomb class we created two different bombs that we placed a certain distance away from each other and away from the bounds of the image so that the game would be more difficult. We spent a lot of time tested the positions and making sure the random function was working correctly since we did not want to have the bombs in the same position every time. After doing this, we ran into difficulty in placing the bombs as time moved on. We did some research on this and realized we needed to include a reposition method that could be called in the PlayState. This method would return the position of the bombs, which was basically the same as the ones we initialized before. The biggest issue was when we had to call the reposition function in the update method in PlayState to keep adding more bombs as the helicopter went on. We had to create an equation that would add bombs as the screen moved on by taking the camera position into consideration. This required a lot of tested which took us a lot of time and research to figure out.

One other challenge that we faced was setting the boundaries of the bombs. Since our bombs position kept on changing, we had to make sure that the bounds of the bomb also changed. We figured out how to do this by setting the bounds to the position of the bomb. We had to generate methods to return the bombs and their positions so that we could access them when adding the bounds for each of the bombs. We kept on getting lots of errors in the bounds which required a lot of testing and editing. This problem also led to another problem where the bombs would disappear before they reached the end of the page. We fixed this error by editing the update function so the screen would have to hit the end of the bomb to be removed.

Overall had many challenges with the errors in running the program. When we ran the program to test it, the program would compile without any errors but the program would crash unexpectedly. These errors were challenging because we did not know where the error would come from. We would always have to go back and look at the code line by line. Each time we realized that we had small minor errors such as a misspelling when we were importing a file. The program would just crash but it would not tell us there was an error when importing the image.