The Creation of Bob’s Pirate Game

Group: ESP 10

Jeronimo: Balsamiq, video, background, graphics

Alan: Processing code

Max: Java code

Video: In this folder

Balsamiq: <https://fau.mybalsamiq.com/projects/esp10/Pirate%20Ship%20Game>

Background:

Our background knowledge with coding and apps was quite minimal to start with. Maxwell knew a little bit on the basics of Java and code, and Alan knew the basic fundamentals of processing. Jeronimo also knew just the basic fundamentals of Photoshop. With this knowledge they could make a very rudimentary app. The one that they initially decided on was a recreation of Dietel’s Cannon Game with a twist. We would import sprites into the game instead of the cannon, the blue and yellow (Were those the colors) rectangles, and the black blocker. We would also add a tilt sensor to allow us to move up and down on the screen. Armed with this information that we had, we began to design our app, Bob’s Pirate Game. However, due to much difficulty with editing the images that Dietel’s code was dynamically drawing on the canvas, the idea of the game was eventually changed to one in which the user would weave his character through various pirates to reach a pile of gold.

Abstract:

This is a paper on the process of making our app “Bob’s pirate battle” which was for the FAU ESP class “Applications for Google’s android”. This will detail how we made our app and the difficulties that we ran into making the app. This will detail the processing of switching ideas and providing insight on designing graphics and how we made the code and giving what we could improve and change if we could do the app over again. Bob’s pirate battle was made to be a similar game to Brick breaker but with different mechanics, but ultimately was changed to an avoid the object game.

Methods:

In addition to the graphics, which were primarily created in Piskelapp by our designer, Jeronimo, the actual code for the game was done in Java and Processing. The controls for the game utilized an accelerometer, which measures the static forces acting on the phone/tablet and changes the speed and direction of the character that is to be controlled accordingly. This tilt sensor was crafted primarily in Processing by Alan with the aid of the sensors example provided by the example offered by Processing. A class called FallingBalls was created and within that class the constructor FallingBalls, with parameters PImage yo, int xa, int ya, int si, float sp, which are each arbitrary representations of the image to be loaded (the protaganist), the x and y coordinates of the image, and the size and speed of the image, respectively. Within the class, there is also a method called update that contains the function image(img, x, y, esize, esize), img, again being a reference to the image of Bob, the protagonist, x and y, his coordinates at any given moment, and esize his dimensions. X and y were defined as x-=ax\*speed and

y+=ay\*speed, so that the x and y coordinates are dependent on the image’s speed and the acceleration vectors acting on the tablet, which depend on its position relative to the surface of the earth. The update method also contains if statements that ensure that the image does not go off of the screen. Within another class called TiltSensor, an array of one item of the class and constructor FallingBalls was created. Then two for loops, one in the void setup method and the other in the void draw method were created, so that the image would move according to the information in the FallingBalls class. These loops are for(int i = 0; i < fallingballs.length; i++) { fallingballs[i] = new FallingBalls (yo, x, y, si, sp);} and for (int i = 0; i < fallingballs.length; i++) {fallingballs[i].update();}, respectively. Within void setup all of the initial variables are defined, such as int si = 100, loat sp = si\*0.2, int x = 0, and int x = 0. The portrait orientation of the application and the various loadImage functions, which refer to the pictures within the project’s assets directory, were specified. Two more arrays, Pirate[] **pirates** and PImage[] **pImages**; were defined and used in the setup method to generate the various pirate sprites with the aid of several more variables, for loops, and loadImages functions. Within the main draw method, in addition to making the background black, the gold pile is drawn by the image function, as are the pirate sprites and Bob, though the latter two make use of for loops that reference the pre-defined arrays and the update method, respectively. It is noteworthy that all dimensions were defined in terms of displayWidth and displayHeight, so that they would scale in accordance to the device used. The shakeEvent and accelerationEvent methods work in tandem with the AccelerometerManager.java file taken from the examples in Processing to make the accelerometer function correctly. Lastly, several for loops and if statements were used in the class FallingBalls, to code the main functions of the game, such as respawning at the starting position if there is a collision with an enemy pirate an image appearing if the player successfully reaches the gold pile, and the game resetting 2.5 seconds after the player wins. The positions of all of the enemy pirates that are to be avoided are continuously generated by a while function for each pirate in the array such that the pirates are distributed randomly across the screen and are not too close to the upper left hand corner, which is Bob’s initial position. This occurs every time the app is launched or reset, so that the player can play different games in succession.

The menu that was coded but not implemented into the final app, had 3 button a start button a help button and a about button. When a click is registered on the button it is picked up by the onClick listeners which then change the activity to which ever button you pressed, either giving credits to the designers, or telling you how to play the game, or starting the game.

Results:

The overall results of the final app are satisfactory. The graphics created by Jeronimo were well done and perfectly implemented by Alan’s code, which also successfully allowed the user to move Bob by tilting their phone/tablet. The game mechanics, such as the respawning, victory display, timer, and resetting,which were coded with Java principles in the Processing IDE in Android Studio by both Max and Alan functioned smoothly. The only noteworthy problem was integrating the menu, which was coded in Java by Max, with the rest of the app, as whenever the Start button was clicked the app crashed. The other functions of the menu, such as the help and about buttons both work correctly. In general, the final app is much better than the previous prototype in which the pirate and gold pile sprites were represented by red and green squares and there was no indication of loss or victory other than a red or green rectangle appearing at the top of the screen.

Discussion:

The idea for Bob’s pirate battle was created from finding a bland app that was boring and attempting to update the artwork and the gameplay of the app. The graphics for the pirates were coming along fine but since the code for the Canon game was different from what we expected as it used the canvas to draw images on it instead of importing images from a library. And since the Canon game code that we were using manipulated the objects as they were defined in the code editing the code was almost impossible without redefining the game. So we switched to the “avoid the objects” game. Since this was fairly easy to code Maxwell Willett and Alan Fruge make most of the code themselves, they had to borrow the accelerometer code to detect the tablet tilting to make the pirate move. We are overall happy with the app produced as it implemented the accelerator and the graphics that we wanted. However the hitboxes for the objects didn’t align with the graphics. If we had more time, the coders would match the objects and the hitboxes so they could be more aligned. Also with more time we would like to add the menu that we coded into the game.

Conclusion:

In the end, if we were given more time the app could've been improved greatly, adding multiple levels of difficulty and randomly generating levels for an infinite experience and we could have implemented a high score system and changed with varying game modes. The app was ultimately underdeveloped.

References:The Processing tutorial uploaded to Youtube by a user named theJaboston was very helpful in making the accelerometer used for moving the pirate function, as much code for this part of the app was borrowed from it. This tutorial can be found by clicking the following link: <https://www.youtube.com/watch?v=gqasfHHsezY>. The AccelerometerManager Java file was taken from the examples provided in the Processing progran and was important in making the tilt controls work, as well. This code can be found by opening Processing and going to File> Examples>Mode Examples>Sensors>Accelerometer.All other code was done from scratch by Alan and Max. All images are original and were created by Jeromino.