Stack Overflow

Project Documentation

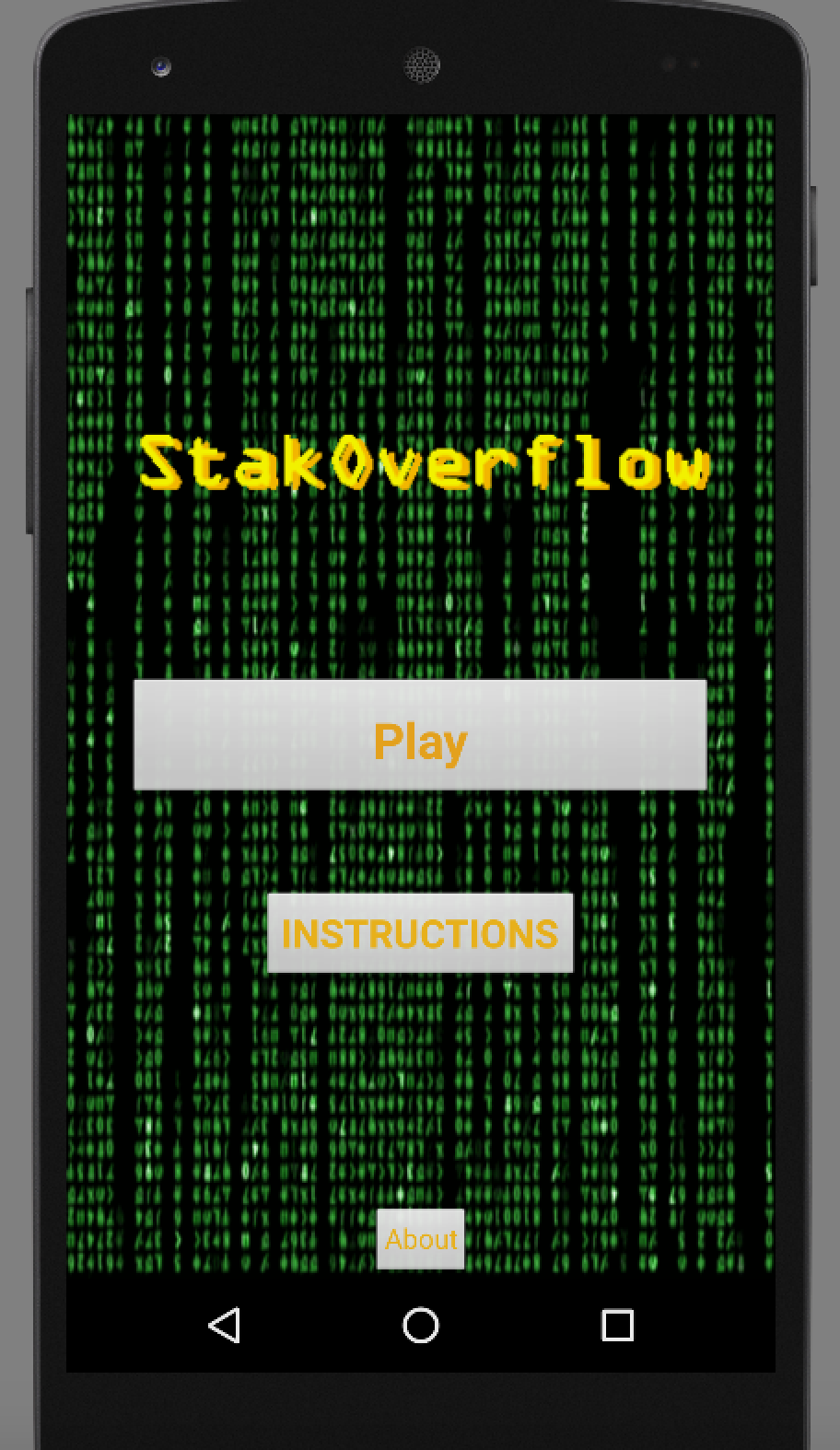
Basic Game Overview:

Our project idea was to create an app called “Stack Overflow”. The game would feature Professor Densmore as the main character, who would move at the user’s direction across the bottom of the screen. If the user pressed on the right side of the screen, Professor Densmore would move to the right, and vice versa with the left side of the screen. The main objective of the game would be to avoid “error codes” falling from the top of the screen. If Professor Densmore hit one of these error codes, he would lose a life; after a certain amount of lives lost, the game would end.

Front End information:

Note: No one in the EmergenC++ group had an android phone. Pictures shown are from the Nexus 5 emulator in Android Studio.

The user is first presented with the home screen of the app. It has three buttons: one for crediting the app developers, one for instructions, and one to actually play the game. A picture of the home page can be seen below: (Note: Stack Overflow was intentionally misspelled)



Next, when the user wants to play the game, they tap on the “Play Game” button, and are presented the main game screen.

The user can hold down on the sides of the screen to make Professor Densmore move. Holding the left side of the screen produces a leftward motion, and holding the right side of the screen produces a rightward motion.

Error codes begin to fall from the top of the screen. The user’s goal is to help Professor Densmore avoid these error files. If they cannot, and Professor Densmore ends up colliding with too many error files, then the stack overflows, and the game ends.



The user will then be brought back to the home screen, with the option to play again.

Back End Information: (Stack Overflow 1.0)

In our original rendition of the code, we started out by creating multiple java and xml files and linking them together with buttons using Intent. This is how we created the “Instructions” and “About” pages and linked them back to the main menu page.

Button Linking example:

**private** Button **PlayButton**, **InstructionsButton**, **AboutButton**;

@Override

**protected void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.***activity\_main***);

Button InstructionsButton = (Button) findViewById(R.id.***InstructionsButton***);

InstructionsButton.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View view) {

Intent intent = **new** Intent(MainActivity.**this**, InstructionsPage.**class**);

startActivity(intent);

}

});

The main game ran using a simple game engine. The majority of the code for this part of the project was supplied by Game Code School, and can be found here: <http://gamecodeschool.com/android/building-a-simple-game-engine/>.

For libraries, we used the libraries included in the game engine code example. We used the graphics libraries for Bitmap, BitmapFactory, Canvas, Paint, and Color. These were used to draw out graphics on the screen when the draw() function was called. We also used three different view libraries for touch and motion detection from the user. These libraries gave us the functions we needed to move Professor Densmore’s sprite across the screen based on user input.

Here is the pseudo-code that Game Code School uses to describe how the engine works:

SimpleGameEngine class{

onCreate method{

Initialize new GameView object

Set new GameView object as the view

}// End onCreate

GameView class{

// An inner class within SimpleGameEngine

Setup the class in special constructor method

Run method calls update, draw and controls frame rate

Update method

draw method

resume method:runs when game starts and sets up our thread

pause method:runs when the game is shut down and stops our thread

onTouchEvent method is called by Android when screen touched by player

}// End of GameView inner class

// Back in SimpleGameEngine class 2 more methods

onResume method is called by Android when game starts and calls GameView resume

onPause method is called by Android when game shuts down and calls GameView pause

}// End of SimpleGameEngine and all of our code

Essentially, the game engine contains a GameView class, which holds everything needed to run the game. It starts by initializing all of the required parameters, and preparing the graphics we want in the view (i.e. Professor Densmore, Error Codes). Then, the engine draws these items for the first frame. After this first frame, the game updates the positions of Professor Densmore and the error codes with its update() function, and then uses the draw() function to redraw the objects in this position.

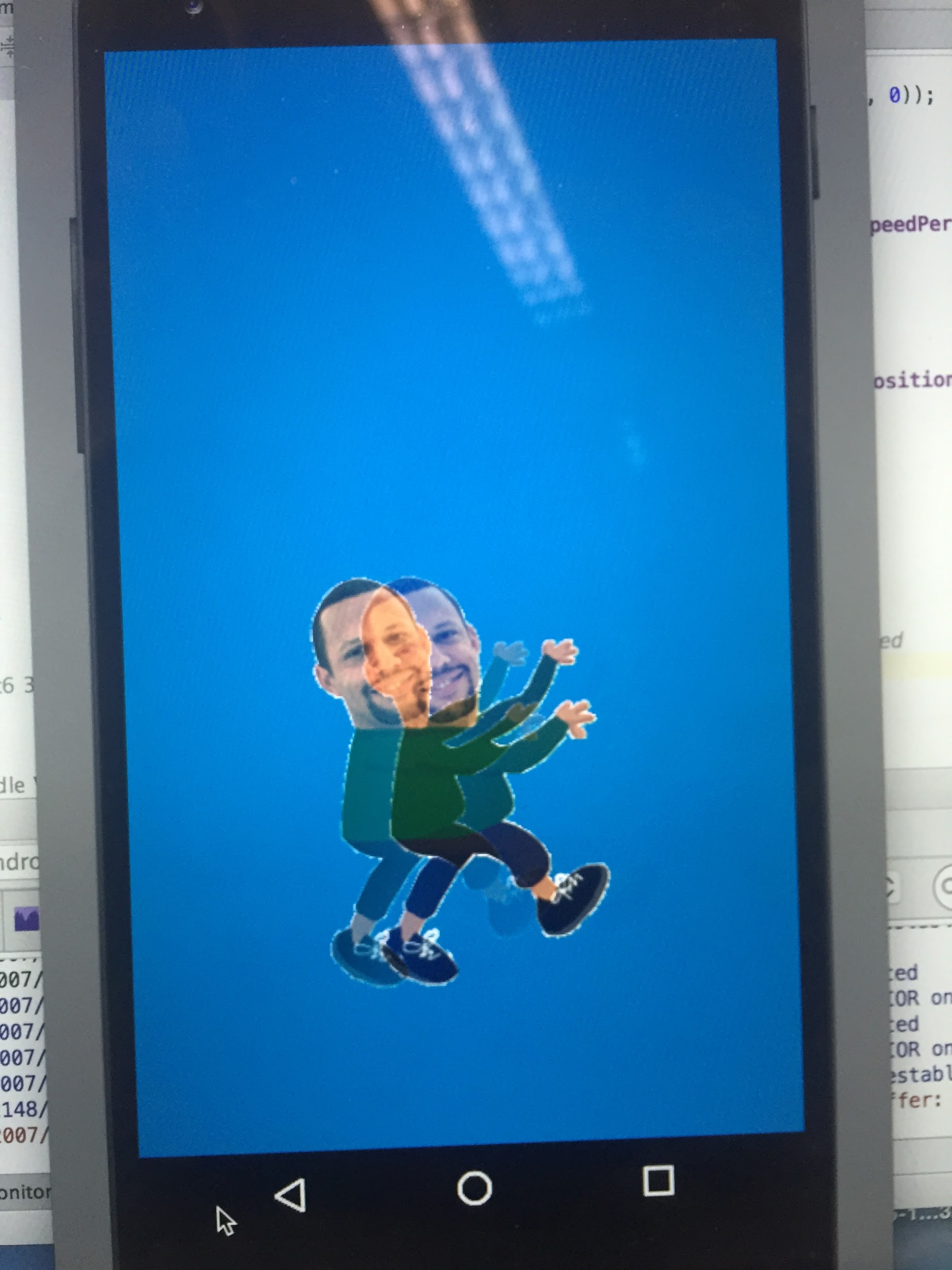
While this way of coding seemed like a good idea in theory, it proved to be a challenge to program our game in this format. the sample game engine was made to just show a character walking on a screen; we had to program our falling objects in addition to this.

Some of the issues we had include:

* Professor Densmore’s sprite moving only in one direction
* Professor Densmore’s sprite moving off the screen and into infinity (crashing the app)
* The sprite moving with one touch, but never stopping (also crashing the app)
* Programming the error codes to fall, without the sprite moving at the same time
* Programming multiple errors to fall
  + Tried programming it so that a new error would spawn when the first one got to a certain y value
  + Trying to program the same error falling when it reached a certain y value.
    - i.e. once it gets to 0 then the y value resets and the error goes to the top of the screen
* Trying to randomize the x value for spawning the errors, resulting in rapid side to side movement because of the update function
* Figuring out collisions, which we knew how to do in theory and with classes, but not in this type of coding with images

In the end, these errors proved to be too challenging to solve. More often than not, solving one error would create another error, and this error could be solved, but then the previous error would happen again. This mostly happened when trying to randomize the x value for the spawning of error code. In these cases, we could only solve one or the other, but not both. It turned into a loop going from one error to another and back again.

Example of error in Stack Overflow 1.0:



We’ve included the files for this version of the project in addition to Stack Overflow 2.0. Even though it didn’t completely work, it’s still a record of the work we put into trying to code this game.

Back End Information: (Stack Overflow 2.0):

About three-quarters of the way through making our app, we discovered a tutorial for a game where a bucket collects raindrops that fall into it. The code for this game can be found here: <https://github.com/libgdx/libgdx/wiki/A-simple-game>.

However, we continued to work on our game engine code up until Friday. When we realized that our original idea wouldn’t work the way we wanted it to, and that we couldn’t resolve enough of our errors to make the game functional, we went back to the raindrop tutorial.

We based Stack Overflow 2.0 heavily on this tutorial. While we didn’t include the code for a sound to play upon collisions, we did add an end condition to the game. The original raindrop code was on an infinite play state because its goal was to be relaxing, with no real objectives. Usually, the objective of programming is to finish an assignment or project with as little errors as possible. Our game’s fail state was inspired by the “fatal error: too many errors” compiling error.

For the libraries of Stack Overflow 2.0, we used the custom libraries that were included in the raindrop tutorial. These libraries included functions that allowed us to scale the app to phones of different sizes, obtain user input through the touch screen (this time, in the form of a dragging motion instead of a holding motion), and use the current system time to time spawning and animations.

The first part of this code involved loading in the graphics we chose from our game, and creating Rectangle class objects in order to represent them in logical operations and functions (such as collision detection) and access their values with ease. Then, our error and compile code objects were placed in arrays. This allowed us to spawn in multiple code objects at the same time, an error we couldn’t resolve with the code in Stack Overflow 1.0.

The next part of the code focused on updating the game and processing user input. It also included statements to prevent the character sprite from going out of the screen’s bounds (again, another major error we had in Stack Overflow 1.0). We set the time for new code to spawn, as well as the time for new errors to spawn. Then, the entire game was rendered based on what state was active (MENU, GAME, END).

All of the java code in Stack Overflow 2.0 was in one class, instead of multiple classes like in Stack Overflow 1.0. The background images were drawn based on events and if/else statements rather than linking buttons or making custom layouts in the XML files. This made the graphics harder to customize, but it made the coding of the game itself much easier.

We learned a lot from doing this project. We discovered that Java is similar to C++, but not quite the same. We can now say that we have basic knowledge about how to use Android Studio. Also, Android Studio is a helpful development platform because it tells you exactly where your errors are, and even gives you hints about how to fix them.

Things we learned in class that were reaffirmed during this project include the benefits of using Classes, how Object Oriented Programming makes the life of the programmer easier, and the usefulness of arrays.