

Software Engineering – From Idea to Interface

SUMMARY:

Throughout a majority of my senior year in Programming and Web Development, Yannick Almeida, Ethan Guillotte and I created a 2D platformer in Java. First, we brainstormed concepts of the game and jot down notes for what the game would be like. Next, my team and I created the core functions for the game based around the concepts that we initially forged. Lastly, we started implementing a story and a style to our game to make the appeal for the players of our game. With this basic layout in mind, we were able to create a game for a virtual software engineering event for Business Professionals of America.

BACKGROUND:

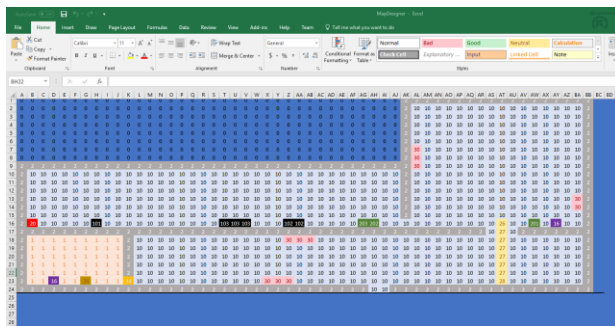
All of us started the project with no prior experience to creating games from scratch in Java. There were many times where we were stumped or had to backtrack in order to make the game more efficient for the hardware to handle. In order to solve these problems, we drew many diagrams and spent a lot of time consulting each other and our programming teacher. We also had this project on GitHub in order for us to work on the game from school and from home together. We spent countless hours working on the project both individually and as a group.

SOFTWARE:

- Java IDE (Eclipse Oxygen is used for this project)
- Slick2D Jar Libraries
- Mail Jar Libraries
- Any browser with access to the internet
- Adobe Photoshop
- Paint Tool SAI

PROCEDURE:

To initially create the concept of the idea, we sat down at a table throwing ideas out into the open and brainstorming how we could fit it into a 2D platformer. Then, once we got an idea, we started to base our thoughts around the concept, which happened to be a platformer in space. The function of reversing the player's gravity was introduced, and everyone accepted the idea.



1: Our Map Designer in Eclipse

all of our computers at home. During the process of creating our environment, the idea of using GitHub to store everything came into mind, so I manufactured a Git repository for us all to access universally. With an idea in mind and the computers to power it, we eventually got to work.

During the beginning of the coding process, progress came relatively slow. We research how to create game states in the Slick2D environment and spent our time getting familiar with the libraries it offered us. Then, we followed a tutorial to create the basic controls for a player so it can move around the screen. Once we had that working, progress began to move a bit quicker. In little to no time, we started implementing gravity, reversing gravity, jumping, and even a camera system. A couple months into the project, however, the game hit a huge roadblock. In order for the game to function, we would need to have

Once this unique function was created, we started sketching level designs that could function well with this idea. After sketching the designs, we started thinking of the resources we would need for the game and how we all would plan for the future. We researched some game libraries for us to use and eventually found Slick2D. We set up our IDE on a couple of the stations at school and

```
package main;

import org.newdawn.slick.Color;

public class BasicGame extends BasicGameState{

    private Image alien = null;
    private Image bg = null;

    public static int id = 1;
    private Rectangle square;
    private Rectangle collide;
    public static int x = 350, y = 350;
    public static boolean collides = false;

    public BasicGame(){
        collide = new Rectangle(350, 350, 50, 50);
        square = new Rectangle(x,y,50,50);
    }

    // This runs as soon as we compile the program.
    public void init(GameContainer gc, StateBasedGame sbg)
        throws SlickException {
        alien = new Image("data/alien.png");
        bg = new Image("data/bg.jpg");
    }

    // Renders content to the game / screen
    public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
        throws SlickException {

        bg.draw(0,0);
        g.drawString("X: " + x + " Y: " + y + " collides " + collides, 10,50);
        g.setColor(Color.cyan);














        collide = new Rectangle(350, 350, 50, 50);
        square = new Rectangle(x,y,50,50);
    }
}
```

2: Code from the First Version of Our Game

a collision system for the player. No matter what we tried the system would either not work or become so flawed that the game was unplayable. In order to get past this road block, we all start drawing several diagrams on how collision would work. Although it took a couple of months and a lot of teamwork, we managed to get our collision system fully functional. After this roadblock was cleared, progress increased dramatically. Enemies were quickly added into the game with their own collision system. Map design became possible and we could now build a health system for every mob. The GUI for the game was able to receive a huge overhaul due to all of us not having to worry about collision. Audio and emailing systems were quickly fitted into the game.

All of the data that we used for the game went from a few separate files to one huge file. Even though we had a few hiccups in the software engineering process, we were able to get something that resembled a 2D platformer video game.

We eventually got our project to the point that we could start filling the game with material that suits our game. I started drawing the graphics for the game and creating sprite sheets for each enemy. One of the requirements for the project was an in-game tutorial, so eventually we started designing around that. To create the tutorial, we created a script for the first

Name	Done	Demo	Description	Number
Blank Tile	Y		Blank (Transparent Tile)	0
Floor w/ Rust 1	Y		Floor Tile with slight rust	1
Floor w/ Rust 2	Y		Floor Tile with more rust	2
Normal Floor	Y		Floor Tile with no rust	3
Glass	Y		Pane of Glass	4
Antigravity Machine (OFF)	Y		Antigravity machine while it's off	5
Antigravity Machine (ON)	Y		Antigravity machine while on	6
Antigravity Machine Light (ON)	Y		Light above machine while machine is on	7
Circular Light (OFF)	?		A circular light that is not lit	8
Circular Light (ON)	?		A lit circular light	9
Light (OFF)	?		An unlit light	10
Light (ON)	?		A lit light	11
Warphole	Y		(TILE TO BE UPDATED)	12
Metal Plate	Y		Metal floor plates	13

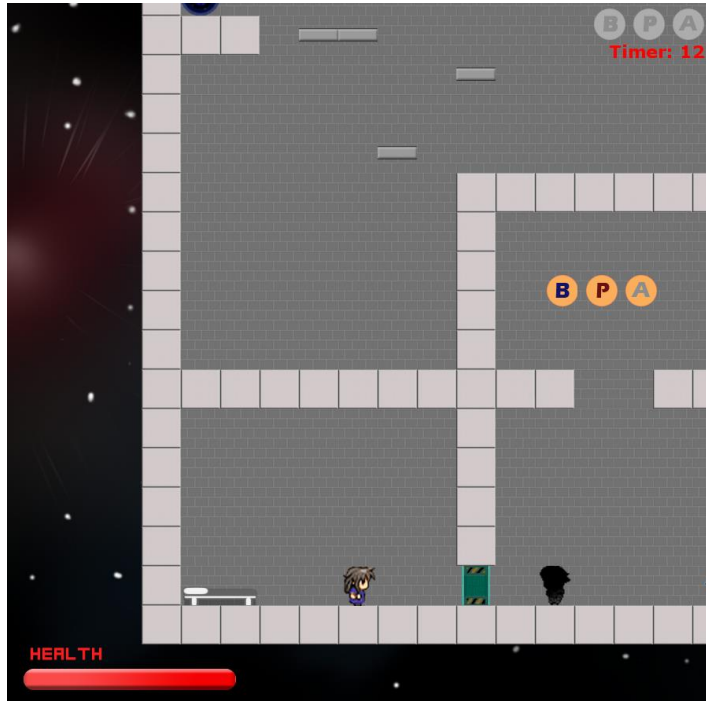
3: A Word Document with Textures For Us To Use

level. We then created a cut scene system to give our story some style to it. Lastly, we designed different

scenarios in our levels which require the player to discover the game's full array of different functions. Tons of Photoshop and Paint Tool SAI files were created filled with different GUI images and textures for us the use in-game. To make the game seem livelier, sound effects for different functions of the game and ambience were added. With all of the core functions done, we were able to tailor the game to our desires and produce an original project all by ourselves.

RESULTS:

In the end, we managed to create a fully functional game tailored to the competition's criteria and still contain original ideas implemented into the project. We managed to have a game with maps, audio, events, mobs, items, cut scenes and even a user interface. The game was manipulated in such a way that once one level was done, we can design infinite levels.



4: A Screenshot from the Final Version of Our Game

```
for(int j = 0; j < tileDs[i].length; j++) { //Parse through a single row
    // Check for Player placement
    if(tileDs[i][j] == 999){
        this.playerStartingPosition = new Point(j*50, 1*50);
    }

    // Map Creation
    if(creatorID == 0) {
        // Static Tiles
        if(tileDs[i][j] < 11)
            tiles[i][j] = new Tile(i, j, tileDs[i][j]); // Create a static tile, texture and passability based on ID
        else {
            // Create a default tile where any dynamic tile will be placed
            if(tileDs[i][j].isPassable())
                tiles[i][j] = new Tile(i, j, tileDs[i][j].getId());
            else if(tileDs[i][j].isPassable())
                tiles[i][j] = new Tile(i, j, tileDs[i][j].getId());
            else if(tileDs[i][j].isPassable())
                tiles[i][j] = new Tile(i, j, tileDs[i][j].getId());
            else if(tileDs[i][j].isPassable())
                tiles[i][j] = new Tile(i, j, tileDs[i][j].getId());
        }
    }

    // Event Creation
    if(creatorID == 1) {
        // Dynamic Tiles
        if(tileDs[i][j] > 10 && tileDs[i][j] < 101) {
            if(tileDs[i][j] == Tile.EVENT_TILE_ID) { // Gravity Pad
                tiles[i][j] = new EventTile(i, j, tileDs[i][j], 0, 0, 50, 50);
                Main.utl.levelEvents.addLevelEvent(i, j, "event", false);
            }
            else if(tileDs[i][j] == Tile.GRAVITY_PAD_ID) { // Gravity Pad
                tiles[i][j] = new Dynamic(i, j, tileDs[i][j], 0, 40, 50, 10);
            }
            else if(tileDs[i][j] == Tile.LEVER_ID) { // Lever
                tiles[i][j] = new Tile(i, j, tileDs[i][j], 0, 0, 50, 50);
                tiles[i][j] = new Lever(i, j, tileDs[i][j], 0, 0, 50, 50);
                Main.utl.levelEvents.addLevelEvent(i, j, "lever", false);
            }
            else if(tileDs[i][j] == Tile.DOOR_ID) { // Door
                tiles[i][j] = new Door(i, j, tileDs[i][j], 0, 0, 50, 50, new Tile(0,0,tileDs[i-1][j]), new Tile(0,0,tileDs[i+1][j]));
                Main.utl.levelEvents.addLevelEvent(i, j, "door", false);
            }
            else if(tileDs[i][j] == Tile.SPIKE_ID) { // Spike
                tiles[i][j] = new Spike(i, j, tileDs[i][j], 0, 0, 50, 50, new Tile(0,0,tileDs[i-1][j]), new Tile(0,0,tileDs[i+1][j]));
            }
            else if(tileDs[i][j] == Tile.LADDER_TOP_ID || tileDs[i][j] == Tile.LADDER_MIDDLE_ID || tileDs[i][j] == Tile.LADDER_BOTTOM_ID) {
                tiles[i][j] = new Ladder(i, j, tileDs[i][j], 0, 0, 50, 50);
            }
            else if(tileDs[i][j] == Tile.PLATFORM_ID) { // Platform
                tiles[i][j] = new Dynamic(i, j, tileDs[i][j], 0, 16, 50, 17);
            }
            else // Default Dynamic Tile
                tiles[i][j] = new Dynamic(i, j, tileDs[i][j], 0, 0, 50, 50);
        }
    }
    tiles[i][j] = new Tile(i, j, 0); // Create a default tile where any dynamic tile will be placed
}
```

5: A Screenshot of Our Final Code, More Specifically the LevelManager

CONCLUSION:

By the end of the project, we had a 2D platformer that we decided to call "Linavity". This game consisted of multiple features, tons of graphics and sound, and quite a few well designed levels for the player to trek around. Not only did we create something we can call our own, but throughout the entirety of the project we applied all of the knowledge we gained throughout our three years in Programming and Web Development. We delved into the software development process by testing things out, retracing our steps, and physically planning things out. With this experience, we are able to head out of high school already filled with some experience in our desired field.