**Work in Progress Report**

By: Deep Raithatha, Jose Rivas, Jash Pandya

GitHub Link: https://github.com/JoseRivas/FlightlessFury

**Major developments/breakthroughs(reference specific code please):**

**Github Branching**

In a single repository we can have the master branch(main program) and a branch for every scratch project that we can merge to the master later.

How to pull:

Once you have checked out the main repository, In android studio click **VCS>Git>Branches** and there should be a section that says **Remote Branches**. Select the desired branch and you can pull changes.

How to push:

To create a branch you just click **New Branch** and type the name. You can then push to the remote repository and it will create a remote branch.

**Rotational Movement with Text Button Actionlistener**

Github Branch: Angles

<https://github.com/JoseRivas/FlightlessFury.git>

To create the buttons, we had help from <http://stackoverflow.com/questions/21488311/libgdx-how-to-create-a-button>

We used the texture packer to create the texture atlas of the button. the atlas is being called here.

buttonAtlas = new TextureAtlas("button.pack");

The texture atlas contains all information about our buttons. it defines where our buttons are in the png file since there is only one png file for the buttons (when pressed and not pressed).

upButton.button.addListener(new ChangeListener() {

@Override

public void changed(ChangeEvent event, Actor actor) {

penguin.peng.rotate((float)45.0);

}

});

The piece of code above is the actionlistener when the button is pressed. It rotates the penguin. which was one of our major breakthrough. we had 2 images set on the screen but we couldn't make the penguin rotate. The penguin wasn’t rotating because we were trying to rotate the texture instead of the sprite.

This is also a breakthrough because this is the first branch that has multiple classes. all classes are controlled by the main class called jdjgame.java.

We also owe a major thanks to Matt for helping us get our buttons working. Thanks Matt :)

We really enjoyed when we had to implement the actionlistener on the button. We searched all over the internet and edited a lot of code just to find one little problem. we were rotating the texture instead of the sprite.

**Orthographic Camera**

**GitHub Branch: OrthoCam**

Source Link: http://www.gamefromscratch.com/post/2013/11/06/LibGDX-Tutorial-7-Camera-basics.aspx

camera = new OrthographicCamera(1280, 720);

Gdx.input.setInputProcessor(new GestureDetector(this));

……

public boolean pan(float x, float y, float deltaX, float deltaY) {

// TODO Auto-generated method stub

camera.translate(deltaX,0); Gesture method for panning camera.update();

return false;

}

What is Orthographic Camera vs Perspective?

Basically an orthographic camera renders what is in the scene exactly the size it is.

Once the image is panned, simultaneously the boolean pan method with do camera.update(); to simulate panning.

**Collision Detection**

GitHub Branch: Collision

Source Link: <http://programmersweb.blogspot.ca/2012/07/simple-libgdx-box2d-bouncing-ball.html>

**//Ground body**

BodyDef groundBodyDef =new BodyDef();

groundBodyDef.position.set(new Vector2(0, 10));

Body groundBody = world.createBody(groundBodyDef);

PolygonShape groundBox = new PolygonShape(); **// imaginary floor, where the ball is bouncing**

groundBox.setAsBox((camera.viewportWidth) \* 2, 10.0f);

groundBody.createFixture(groundBox, 0.0f);

**Using Box2d we implemented created a world based on vectors.**

World world = new World(new Vector2(0, -100), true);\

Variable : static final int velocity = 10; // set the velocity of the ball (Approx -9.8 m/s {down})

// Dynamic Body (Ball)

BodyDef bodyDef = new BodyDef();

bodyDef.type = BodyType.DynamicBody;

bodyDef.position.set(camera.viewportWidth / 2, camera.viewportHeight / 2);

Body body = world.createBody(bodyDef);

CircleShape dynamicCircle = new CircleShape();

dynamicCircle.setRadius(5f); // radius of the circle/shape

FixtureDef fixtureDef = new FixtureDef();

fixtureDef.shape = dynamicCircle;

fixtureDef.density = 10.0f; ←-----real world attributes such as density

fixtureDef.friction = 0.0f;←-----real world attributes such as friction

fixtureDef.restitution = 1;

**Velocity & Launching the penguin**

Github Branch: master

Sources:

<https://github.com/libgdx/libgdx/wiki/Box2d>

<http://www.gamefromscratch.com/post/2014/08/27/LibGDX-Tutorial-13-Physics-with-Box2D-Part-1-A-Basic-Physics-Simulations.aspx>

<https://github.com/libgdx/libgdx/wiki/Bitmap-fonts>

Here we Added text to the button and made it say LAUNCH

texture = new Texture(Gdx.files.internal("LiberationMono.png"), true); // true enables mipmaps

texture.setFilter(Texture.TextureFilter.MipMapLinearNearest, Texture.TextureFilter.Linear); // linear filtering in nearest mipmap image

font = new BitmapFont(Gdx.files.internal("LiberationMono.fnt"), new TextureRegion(texture), false);

textButtonStyle = new TextButton.TextButtonStyle();

textButtonStyle.font = font;

button = new TextButton("LAUNCH", textButtonStyle);

We then added an actionlistener where if the button was pressed, the penguin would move at a constant velocity using the box2d physics body

button.addListener(new ChangeListener() {

@Override

public void changed(ChangeEvent event, Actor actor) {

//if button is pressed do stuff

body.setLinearVelocity(5f, 5f);

}

});

**Major Challenges/setbacks( reference specific code please):**

**Orthographic Camera:**

Capture.PNG

the program gave us errors, stating that OrthoTesting class should be abstract or we have to implement outside methods. The result of implementing outside methods:

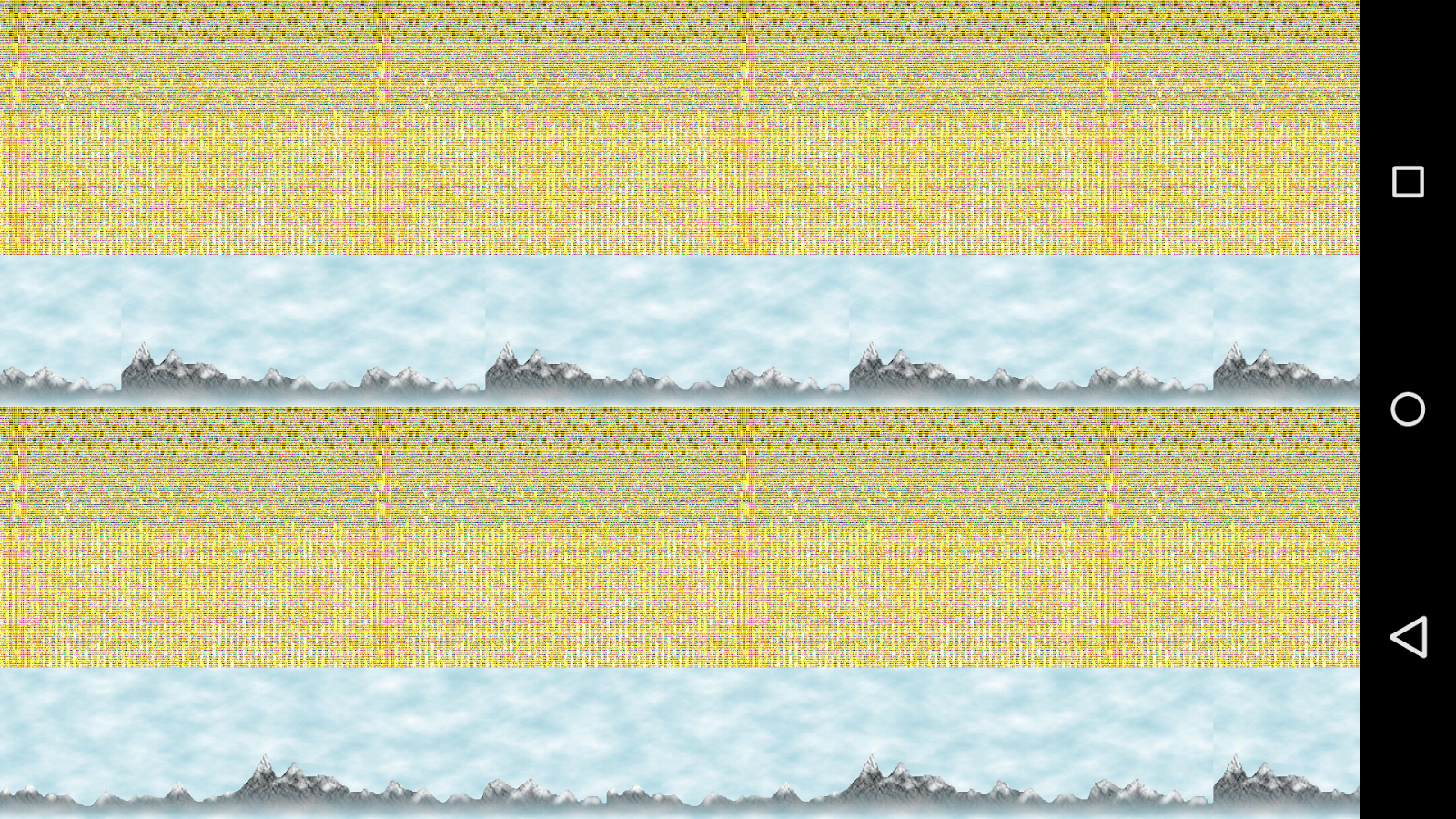
@Override

public boolean panStop(float x, float y, int pointer, int button) {

return false;

}

this function was added and it removed the errors.

**SideScrolling** 

**// draw the first background**

**batch.draw(background, currentBgX - 800, 0);**

**// draw the second background**

**batch.draw(background, currentBgX, 0);**

The picture above dictates our dilemma as the render() method is causing us difficulties. We are currently in the process of

Any modifications to your specifications/release schedule:

|  |  |
| --- | --- |
| **0.1** | **Add launch button**   1. **Launch** |
| **0.2** | **Add action listener to button to perfrom actions** |
| **0.3** | **Add physics Part 1: Gravity** |
| **1.0** | **Press button to launch pengiun** |
| **1.1** | **Side scrolling New: Orthographic Camera** |

**Description of your scratch/test program:**

Describe the generic concept you needed to test out:

Orthographic Camera

Source any web site/book that helped you with that concept:

//Source: http://www.gamefromscratch.com/post/2013/11/06/LibGDX-Tutorial-7-Camera-basics.aspx

Describe the code and the lesson that you learned from it:

camera = new OrthographicCamera(1280, 720);

public boolean pan(float x, float y, float deltaX, float deltaY) {

// TODO Auto-generated method stub

camera.translate(deltaX,0);

camera.update();

The camera is the perfect resolution and size at which the image needs to be displayed. So that we eliminate the need for side scrolling.