Catani WIP 4

# Breakthroughs:

**Major Efficiency changes:**

Although on the last WIP our game worked it was not the most efficient. Both members of team Catani have optimized many sections of our code. To start we developed a proper public final class for constants. In the class there are commonly used variables such as floor and ceiling measurements and world width and height. These variables can be accessed by calling Constants.FLOOR for example.

Next I migrated classes into proper packages such as screens and obstacles. Also created an enum for all the different screens instead of calling each screen by an int.

Also optimized repeated code used for drawing text by making a function in game that can be called to draw font and you only need to pass it the enum and the necessary info for drawing text.

public void drawText(Assets.Fonts font, String text, float x, float y){

}

**Pitfall & Tree Obstacles**

I worked on making the pitfall and tree obstacles. The easy part was creating the images and then creating the hitboxes for the images. I made each obstacle a childclass to the superclass “SprObstacles”. This allowed me to use the “isHit” function that Ameer created in the SprObstacles class. After I got the hitboxes working, I began making the obstacles spawn on the ceiling and floor. This is when the code started getting messy:

if (obstacle == obsTree) {  
 if (obsTree.getX() >= obsPitfall.getX() && obsTree.getX() <= obsPitfall.getWidth()) {  
 obstacle.setX(Constants.WORLDWIDTH + 200);  
 }  
  
 isTreeFlipped = MathUtils.randomBoolean();

if (isTreeFlipped) {  
 if (!obsTree.isFlipY()) {  
 obsTree.flip(false, true);  
 }  
 obstacle.setY(Constants.CEILING - obsTree.getHeight());  
 } else {  
 if (obsTree.isFlipY()) {  
 obsTree.flip(false, true);  
 }  
 obstacle.setY(Constants.FLOOR + 3);  
 }  
 }

I wanted the program to choose a random boolean value that decides whether the image will spawn on the ceiling or roof. I did this using:

isTreeFlipped = MathUtils.randomBoolean();

After this I check if the boolean is true and then I check if the image is already flipped:

if (isTreeFlipped) {  
 if (!obsTree.isFlipY()) {  
 obsTree.flip(false, true);  
 }

}

If isTreeFlipped is true and obs.Tree.isFlipY() **is not** true, then the image flips. However if isTreeFlipped is true and obs.Tree.isFlipY() **is** true, then the image will not be flipped. This ensures that the tree will be upside down when it is on the ceiling, and it will be right side up when it is on the floor.

For the Pitfall, I used the same format. But instead of flipping the image, I just reset the texture:

Using:

“obsPitfall.setTexture(game.assets.manager.get("ceilinghole.png", Texture.class));”

Looks something like this:

If (isPitfallFlipped) {

obsPitfall.setTexture(game.assets.manager.get("ceilinghole.png", Texture.class));

obstacle.setY(Constants.CEILING - 2);

}

**Organization:**

In order to make the code more readable, I have seperated code into separate functions such as: HitPitfall, HitTree, redrawObstacles.

***Counters:***

**Ameer created the score counter:**

Using a conversion of pixels ran to meters which is the in game score system. Using pixels ran allows the score to dynamically increase with game speed

fPixelsRan += fGameSpeed;

nScore = (int) ((fPixelsRan/ Constants.WORLDWIDTH) \* 3);

**Tim created the coin counter:**

This was done using a simple integer that increases whenever the player interacts with the coin:

If (sprCoin.isHit(hame.chrMain.getBoundingRectangle())) {

ncoinCounter += 1;

…

}

# Challenges:

**Player Hitbox Issues:**

As I was preparing to create the pitfall obstacle, I noticed that the player hitbox was really off. The hitbox was too small for the image, and the image would shift up and down whenever it hit the floor and ceiling. I used shaperenderer to show me what the player hitbox looks like, and also used it to draw lines where we made the “FLOOR” and “CEILING” y-coordinates. I noticed that our ceiling y-coordinate was perfectly fine. But our floor y-coordinate was about 10 pixels under the visibly drawn floor. I also noticed that when the player hitbox would hit the floor or ceiling, the hit box would stop, but the player would keep going for another few pixels before it stops. This was due to this code:

if (game.chrMain.getY() > Constants.CEILING - game.chrMain.getHeight()) {

game.chrMain.setY(Constants.CEILING - game.chrMain.getHeight());

}

if (game.chrMain.getY() < Constants.FLOOR) {

game.chrMain.setY(Constants.FLOOR);

}

I realized that in order to fix this, we have to change the characters velocity variable and not the Y-coordinate. So in order to fix this I came up with:

|  |  |
| --- | --- |
| If (getY <= Contants.FLOOR || getY() >= Constants.CEILING - getHeight()) {  setYVel(0)  }  This fixes the problem of the FLOOR and CEILING y-coordinates and makes it more efficient. The second problem, the wrong hitbox, I fixed by changing the “this.setSize()” code within the sprChar. This is due to the fact that I the player animations are not true sprites, but are in fact textures that act like sprites. |  |
|  |  |
|  |  |

# 

# Updated Release Schedule:

|  |  |
| --- | --- |
| **Release Name** | **New incremental features of this release** |
| **1.0** | **Scrolling background and ability to flip gravity** |
| **2.0** | **Scrolling obstacles with hit detection** |
| **3.0** | **Add larger variety of obstacles such as pitfalls and spawn obstacles with increasing difficulty for the player as score progresses** |
| **3.5** | **Add coin with random spawn locations** |
| **4.0** | **Develop Main Menu and Gameover screen that transition into one another as well as Game screen** |
| **4.5** | **Add dynamic score in meters and speed up scrolling background as game progresses** |
| **5.0** | **Attempt cloud based score and possible multiplayer on android devices** |