**2019-05-03**

**Nick:**

For the past few days, our group has been working on organizing the code into legible and easily understood syntax, while also trying to keep the program functioning (so far, ½ of that has been accomplished). Our biggest challenge we have enjoyed has been organizing our code properly and getting things out of our main program and into classes. Our biggest breakthroughs have been, organization aside, accomplishing our goals for release 1.0 and having a semi base for our game (basic moving, platforms both independent and not, gravity, some sprites). Currently “Rogue Thumbtacks 1.0” has our character jump around, 2 platforms (one of which we want to move independently, the other one only moving with the background when it scrolls to show more of the landscape), and a functioning start screen with a custom start button.

**2019-05-07**

**Nick:**

Today we organized some more, and have been trying to fix the main release code that was broken during the organization process. I’ve managed to make the independently moving platform to function in every way except actually moving on its own.

**2019-05-08:**

**Anish:**

For the last few classes, I have been trying to fix our code, which has been mostly organized. After taking a step back to make sure our code’s foundation is completely solid, many aspects of our game that were previously functional have been completely broken, such as:

* Jumping on platforms: our hero is able to move and jump around freely when on the ground, however once our hero hops onto a platform, he is completely unable to jump. We currently believe that the “canJump” boolean used to determine whether Bill is allowed to jump (he is only allowed to jump when his feet are on the ground or a platform) is being switched to false before Bill’s “nDY” changes and lets him jump. We have tried changing multiple things, and have even tried adding redundant “canJump = true” lines wherever possible, but we will attempt to find the source of the problem by using “println()” functions. By using “println”, we can print out the value of “canJump” when ‘w’ is pressed or when Bill hits the top platform to see if this boolean is the source of our agony.
* The sides of our platforms

**2019-05-10**

**Nick:**

Today our team learned how to make our program less jittery (our hero having a seizure whenever he happens to make contact with something, etc.) by using a boolean to check for collision, and if collision is detected then another boolean, canMove, determines that our hero can not move in that direction, instead of constantly having our hero be pushed into the platform by gravity, and having to correct that by pushing him back up resulting in a jittery program. Anish also had an idea if that doesn’t work, which is to have a boolean turn off gravity when the hero is stationary on the platform, but turn it back on when he jumps just so that the gravity does not push him into the platform at all.

**2019-05-15**

**Anish:**

Today I quickly whipped up a health bar that can accompany our hero Bill. The scratch code for the health bar is extremely simple, being two rectangles with heights of 20px (can be modified to be taller or shorter) and lengths of multiples of 100 (specifically 300 right now) since Bill’s full health is 100. The rectangle at the front has a green fill and can be modified (length wise) and the rectangle behind is jest a flat white. Although we have written down, on our release schedule, a different next step, I currently feel that we should get a nice base game first, before trying to work on a larger “level generation” mechanic.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Nick:**

We decided to remove the gravity toggle when our hero hits the platform since Mr. Grondin said that “Gravity never turns off” and is always used for interactions and you have to work with it.

**2019-05-17**

**Anish:**

I have decided to work on the “level generation” mechanic. My plan right now is to create a random path on a large 4x4 grid from “A”, the start of the stage, to “B” the end of the stage. This is the main path of the stage and needs to be completely unobstructed. The rest of the grid blocks will then be filled with extra rooms that do not necessarily connect to the main path.

I believe the first required step is to start building **individual rooms**. I want to create 3 types of base rooms, constructed with tiles rather than randomly chosen sized platforms and a ground, one with two openings going left and right, one with three openings going left, right and down, and one with four openings going in all directions (up, down, left and right).

Individual room > Divided into a 10 x 10 grid with each block holding a coordinate > platform & ground blocks placed in specified coordinates

**2019-05-21**

**Anish:**

I have started to work on building individual rooms. I have started to look onto matrices (or two-dimensional arrays). I am having trouble figuring out how to load each cell, but I believe it should be the same as normal arrays where a for loop is used to load each sprite until the max amount of sprites are loaded. I will hard code the level layouts with a matrix, marking the platforms with “1” and empty space with “0”s.

**2019-05-23**

**Nick:**

I began to work towards making enemies/ making enemies a threat today by spending about 20 seconds on Piskel and then having the enemy constantly move towards Bill’s X coordinate. The “better object collision” scratch I’ve been working on for the past few days has been scrapped as there wasn’t anything to improve it without doing some very advanced level code.

**2019-05-24**

**Nick:**

I fixed the enemy a little bit but so far I have only managed to organize some of the code and make it so that the enemy is on the same level as bill, and am attempting (right now) to make him move towards Bill.

**2019-05-24**

**Anish:**

I am implementing proper velocity/acceleration/gravity on my free time at home to get rid of the jitter when correcting our hero’s y coordinate on platforms. Here at school, I have started to hardcode platforms for specific “levels”. Having to hardcode all of these platforms requires me to copy and paste this code for each platform (since I am not using arrays):

nSide = pltOne.nHit(sprBill);

if (nSide == 1) {

sprBill.backUpY(pltOne);

} else if (nSide == 2) {

sprBill.bumpHead();

} else if (nSide == 3 && nDir != 2) {

//fDx = 0;

sprBill.setDx(0);

} else if (nSide == 4 && nDir != 1) {

//fDx = 0;

sprBill.setDx(0);

// println("right wall");

} else {

}

I really don’t like this code right now since it had to check what direction our hero is moving, and our hero’s movement is in the “screen1” class but the collision is based in the platform class… I would like to clean up my code and move collision into the level/screen rather than keep it in platform. I would also like to get rid of the platform class and the walls that the platform uses as-well and just use the platform img/sprite itself for collision.

**2019-06-03**

**Anish:**

I got the new gravity/velocity stuff to work in our game. Most of these movements are derived from the past grade ten platformer “Don’t Give Up”. Here are some elements that I took:

- movement/speed vectors:

To declare our vectors, a set of three in an array, we used this line:

PVector[] vD = new PVector[3];

And to bring them to life we used these lines

vD[0] = new PVector(0, 0);

vD[1] = new PVector(-nSpeed, 0);

vD[2] = new PVector(nSpeed, 0);

\*I thought it was a little confusing that you declare the amount of elements with an integer, ‘n’, but while populating/giving a value to each element, you have to start from ‘0’, thus ending at ‘n - 1’\*

Each element in this array is a vector holding two pieces of information, ‘x’ and ‘y’ in the form of an ordered pair: (x,y). For example, vD[0] holds a vector with a ‘0’ in the ‘x’ coordinate and ‘0’ in the ‘y’ coordinate. One of these vectors is later added to the position vector to move our hero in one direction or keep him in place.

Here is the position vector declaration:

PVector vPos;

Here is the code for updating this vector:

vPos.add(vD[nDirec]);

In the above line of code, vPos.add is getting nDirec, the direction the hero is travelling in; this value is updated by the keylistener in “Screen1”: “sprHero.nDirec = n;” ← ‘n’ being replaced with ‘0’ for no key being pressed, ‘1’ for ‘a’ or left, and ‘2’ for ‘d’ or right. This variable, “nDirec” decides which “vD” vector (the movement/speed vectors) is added to the hero’s x and y coordinates (where nSpeed is a constant). These vectors influence our hero’s x coordinates only.

To update our hero’s y coordinate, we have gravity, containing velocity and acceleration (but not time) variables and a “jump()” function.

void gravity() {

fVelocity += fAccel;

if (fVelocity >= fVelocityMax) {

fVelocity = fVelocityMax;

}

vPos.y += fVelocity;

//println(fVelocity);

refreshCoord();

 }

This code above is the gravity code. Velocity is updated by the key listeners: when “up” is pressed, velocity is given a value of “-21” and when the hero hits the ground or a platform, it is given a value of ‘0’. This velocity is constantly being added to by acceleration, acceleration staying at a constant of “0.8”, and is then added to the y coordinate of our hero. If velocity becomes higher than the max velocity, which is at 16 right now, velocity adopts the value of the max velocity so it doesn’t keep increasing indefinitely. This code is also taken from “Don’t Give Up” and I have successfully been able to adapt it into our game to replace the previous “nDy” and “nG” (nG = gravity) code.

**2019-06-04**

**Anish:**

Today, I got the collision code to work seemingly seamlessly between our hero and the platforms in our program. Screen1 is responsible for checking whether our hero is hitting anything; Screen1 directs the sprite to run a specific function to correct the collision. The sprite, however, is the one that is constantly checking collisions:

Screen1 updates the sprite:

sprHero.update();

sprHero updates the movement and checks collisions:

fXStart = vPos.x;

fYStart = vPos.y;

move();

scr1.collCheckLeftRight();

gravity();

scr1.collCheckUpDown();

This is the only way I could get this to work using the movement/gravity code from “Don’t give up”. I also had to split up “collCheckLeftRight” and “...UpDown” because our hero would keep falling through the floor without doing so. My theory is that the “collCheck...” was correcting the y position of our hero before his y coordinate was even being updated. Thus, “coll..LeftRight” has to also be called only after “move()” since that is where our hero’s x-coordinate is being changed.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Today I also got our hero to traverse the tops of platforms smoothly without getting stuck inside of their borders. Before this update, our hero was able to get into the borders of our platforms and move inside of them. I found that the widths of the top and bottom collision walls of the platforms were offset to the right by 10 pixels, and then shrunken back 20 pixels. This shrinkage left behind gaps, 10 px each, on either side of the platform. When trying to correct thin and make the top and bottom walls the same size of the platform, I ran into the issue of our hero being able to pass directly through the sides of platforms when he collides with the corners. This happened because our “nSide” variable in “Screen1” was only able to hold 1 value representing 1 side of the platform that our hero was hitting. To fix this, I had to split the “isHit” boolean in the platform class into two sections, one that does a check between our sprite and the top & bottom walls and another that does the sprite and the side walls.

* Ideally, I think it would be best to separate the “isHit” booleans completely and check for each wall individually, but I think this should do for now.

After our “isHit” boolean was split, I was able to create two “nSide”, one to check the collision for the top and bottom walls (I used “nSideUD” for this), and ont to check for the left and right walls (I used “nSideLR” for this).

**2019-06-07**

**Nick:**

It’s been a while since I wrote a journal entry mainly just because of absences, but also a lack of any notable activity. I’ve been working on the enemy class for like a week now and got rid of about 10 NullPointerException errors and some static versus non static errors. It’s gotten me to understand what Mr. Grondin meant about people getting frustrated at their code, and the satisfaction of finally getting it to stop crashing. Some edits I made were just making sure sprBill and sprEn actually existed in the places they were being called from (Sprite classes’ public update). Nothing really notable, but important nonetheless. I’ve also received a wake up call from mr Grondin that I should be working WITH Anish, which has proved much more productive.

**2019-06-10**

**Anish:**

Today I got enemies to work. I started basically from scratch: rather than using a separate class for enemies to work like Nick tried to do, I incorporated enemies into the sprite class. All I did was make another sprite, one that is only on Screen1, and updated that sprite based on this code:

void follow(Sprite s) {

if (sprHero.fX >= s.fX && s.fX + 200 >= sprHero.fX) {

nDirec = 2;

} else if (sprHero.fX <= s.fX && s.fX - 200 <= sprHero.fX) {

nDirec = 1;

} else nDirec = 0;

}

I put this code in the sprite class and it basically updates the given sprite’s direction based on sprHero’s x coordinate. If the hero is within 200 pixels right of the enemy, the enemy will move right towards the hero, if the hero is within 200 pixels left of the enemy, the enemy will move left towards the hero.

In Screen1, I wrote:

sprEnemy.follow(sprEnemy);

sprEnemy.update();

These two lines updated the direction of the enemy based on it’s position compared to the hero’s position and then updated the enemy image & checked collisions. The difference between the enemy sprite and the hero sprite is that the hero is updated by the key listeners and the enemy is not.

In the constructor, I also added an extra variable for the sprite to receive; “nType“. This will help me set up collision walls for each side of the sprite so I can add update the health of the hero.

**2019-06-11**

**Anish:**

Today I was able to fix the enemies. Rather than keeping them in the sprite class, I moved them into their own separate “Enemy” class. Their movement is the same (based on the hero’s position) and their collision is the same; the only difference is the enemy specific collision and that fact that the enemy is only instantiated in Screen1 (not the starting screen). Rather than having to check if the sprite is an enemy and then update the walls accordingly, the collision walls for the enemy are always being updated. The walls now work and all I have to do is add collisions in Screen 1.

**2019-06-14**

**Anish:**

For the past few days, I have been trying to work out how to get our hero to react properly to the enemy’s collision walls. Getting the “isHit” to work for the enemy is as simple as the “isHit” for the platforms (I just copied the code in screen 1 and pasted it lower using the enemy isHit rather than the platform isHit), but I would like bill to be “bumped” back when he is hit/hits the enemy (his direction changes -he moves backwards- and he is launched into the air). I have a “bumpbackLeft()” and a “bumpBackRight()” function for the left and right sides of the enemy respectively and am trying to get our hero to be bumped back and **stop** after a couple seconds (so he doesn’t keep moving backwards after getting hit). Everything that I have tried so far hasn’t works:

* I tried taking the frame number using “frameCount” and created a max frame number by adding my desired delay/bump duration to that frame. If the frame was less than the max frame number, I set the direction opposite of the collision wall and I made him jump by making “fVelocity = -10”. I also said to set the hero’s direction to 0 after it checked for this. This didn’t work since the “bumpBaackRight()” function only did what was inside of it once since it is only called when our hero hits the enemy.
* I then tried to replace the if structure with a while loop, added to the “nFrame” variable that sored the frame number, and broke (break;) the while loop if nFrame was >=the max frame number. This made our hero simply stop since our hero was unable to move during the while loop (only the values of the variables could increase.
* I also tried making a timer class, but that ended up completely stopping the program aswell

If the game didn’t completely stop, our hero would just bounce and move backwards until you pressed a key to stop him.

I will try to fix this by first creating a better timer; using the timer to make the hero respond for a cetain amount of time to the collision in “Screen1” rather than in the function

**OR (if the method above does not work)**

I will just work around this and make our hero stop if he collides with the ground after being bumped back