**Project Journal**

Name: *Matthew Meade and Abhinav Agrahari*

#### Mon Oct 19th, 2015

* created “in-game physics” that simulates gravity
  + includes a set rate of acceleration that accumulates in “fVelocity”. The velocity then adds to the location of the “Y” value for the main character

#### Tues Oct 20th, 2015

* Created an Arraylist of walls

#### Wed Oct 21st, 2015

* started to implement collision detection for all walls.

#### Oct 26th, 2015

* Abhi got walls to be drawn through a looped function
* We created an image of “the box” we wanted to use

#### Oct 27th, 2015

* Matt has been trying to get collision detection to loop through the array list and it didn’t want to work
* We started taking a look at Box2D

#### Oct 28th, 2015

* We found that Box2D is a lot harder than it seems, and trying to make programs in it but it hurt his brain
* <http://natureofcode.com/book/chapter-5-physics-libraries/> Daniel Shiffman, who is part of the processing foundation, made a website that really helped us to see what did what
  + This did not work out so well though

#### Oct 29th, 2015

* We tried to figure out Box2D but stuff started to happen, and we had no idea what was happening

#### Oct 30th, 2015

* We decided to give up on Box2D and go back to our original plan
* Matt figured out collisions with arraylists, still trying to debug as if the user holds down a key, the hero goes through the boxes.
* Abhi started to create a moving level, and not let the “camera” go past the boundary
  + this isn’t working out too well as when the main character moves right, it is fine and scrolls as intended (along the x axis), but when the main character moves left, the “camera view” either goes past the left side of the level or doesn’t translate left at all

#### Nov 4th, 2015

* Matt has been trying to get the collision detection with array lists to work for a long time and it’s still really broken. The hero constantly falls past the boxes, but at a slower rate, which means that the Hit testing is sort of working, but not entirely…
* Abhi found a sample processing game that is a platformer , and uses a “camera” like we wanted our game to. We understood the code: the player is the one to move, and the “camera” is centered upon the position of the player, and then the entire level translates by the value of the “camera”
  + Scrolling along the x axis works, and are currently trying to also have scrolling along the Y axis but it’s not working out so well…The “camera” seems like it doesn’t care about the main character.

#### Nov 5th, 2015

* Abhi made the “scrolling along the Y axis” code work… or at least it seems like it will work.
* Matt is getting the hit testing to work with boxes arraylist. It has progressed from yesterday, but is still very glitchy.

#### Nov 11th, 2015

* We tried to come up with another scratch program that has the working collision detecting with arrayLists, and gravity.
  + this works on simple boxes, thanks to Matt! We will implement it to our actual game next class

#### Nov 12th, 2015

* We implemented the scratch program into our game, and it WORKS!
* Abhi added scrolling to the game (as a new release version), and it also works in all directions.
  + There is one a range of certain “y” coordinate values for the player where the frameRate is reduced by half, which appears to the user as lag
    - We can easily fix this by dividing the default frameRate in half in setup (60/2), and then doubling how much the player moves by two, to seem like it doesn’t lag

#### Nov 16th, 2015

* We created a new class “Door” and the hit testing on it, which works
  + when the player touches a door, a message appears on the screen (depending on which door they touched)
* We also figured out how to display a timer in the form “Hour : Minute : Seconds : Milliseconds”
  + We got help from the Processing forums (link is in references)
* created architecture for the code that supports multiple levels

#### Nov 17th, 2015

* We designed three levels for our release “Clone Trooper”, and this is the most stable, and newest/most complete release at this point in time

#### Nov 19th, 2015

* As per your (Mr. Grondin’s) suggestions from yesterday, we looked into classes that “extend” each other and tried to design levels using that architecture of code

#### Nov 23rd, 2015

* We showed you our code and let you work with it
  + After you showed us how to use “super()” on extended classes for constructors, we hard-coded the “if/else” structure to null a class (to clear everything in it)

#### Nov 25th, 2015

* We were able to successfully use inheritance on classes to create multiple levels based on the classes“Level*n*”, where *n* is the level, and a “LevelBase” class

#### Nov 26th, 2015

* As you suggested, we created a separate arrayList for the platforms so that when the level changed, we could null and recreate that arrayList instead of the entire Level Base

#### Nov 27th, 2015

* We are working on a scratch that we can implement into our main project: an eye that looks at you but remains stationary
  + We think that your “shoot4” code from your GitHub will help us with using PVectors for this
  + Abhi has a “sort-of” working scratch code that has a circle “look” in your direction (you are a rectangle). The only thing is that by sort of working we mean that when you move left, the circle has an odd behaviour when it “looks” towards you. Unless you move off the screen, the circle doesn’t move past the it’s origin (the translated one)
* After asking you, we are more clear on the function of “Sprites”, Matt is looking into creating the Sprite class and reorganizing all our code to match.

#### Nov 30th, 2015

* Mr. Grondin’s “shoot” note on Skill Set 9 helped a lot more than the “shoot4” code from his GitHub
  + The eye does look at you now, like it does in the actual Give Up game. When you move right, the eye moves right, when you move up , the eye moves up, etc.
  + What happens is that there are two PVectors: one of the location of “the eye” (the pupil technically), and one of the center of “the eye”. The PVector that is the location of the eye is set to the x/y coordinates of the player. The PVector of the center of the eye is then subtracted from that. After doing so, the location Vector is normalized to keep the ratio of the positions, but “shortened”. It is then multiplied to give the look of an eye
* We also made a scratch that tests a give up button
  + When you hover over it, it changes colour by changing the image, and when clicked on, it displays another image that stays on the screen for 3 seconds using a timer class

#### Dec 1st, 2015

* Matt has got version Colo Claw Fish working, which includes the reorganized code with the sprite class.
* We started to add audio and the give up button into our game

#### Dec 2nd, 2015

* We added music to our game, and redesigned the first three levels
* We also updated the give up button scratch to include audio
  + When the button is pressed, an image is displayed, and an audio clip is played. To do this we used an array of audio clips. The length of the timer is set to the length of the audio clip so that the image is displayed while the audio clip is playing, and as long as the audio clip is playing. This also allows the audio clip to finish playing, and not stop abruptly

#### Dec 3rd, 2015

* We successfully added the give up button and spikes to our game.
  + We created an arrayList of spikes, and hit-testing for them
  + We had to modify the button when adding it to our game, as originally we had not split the spritesheet into an arraylist. Rather, we just showed part of the image. Creating the arraylist of the images from the spritesheet took a while to get working, but it now works!

#### Dec 7th, 2015

* We created a scratch that rotates an image to face the player, using atan2()
  + this did take a while as it was very confusing to why it works
* We integrated this scratch into our main program and it works
  + the image is actually a laser gun that shoots at the player using an arrayList of Bullets (sprites)

#### Dec 9th, 2015

* Matt found a bug with the bullets getting an out of bounds error when they hit the platforms, he attempted to fix it, and probably succeeded, but he’s not entirely sure
* We planned how to program saws
* We made an array of PVectors to update the sprite based upon “nDirec”

#### Dec 10th, 2015

* We created saws and edited the sprite class to match in our main program
  + took a long time to reorganize the constructor
  + the array of PVectors for direction was very very helpful
* We made the lasers an arrayList and found out why we get an “IndexOutOfBounds” error: the bullets were being checked through both lasers, so by the time one laser got to that part, it was already removed by the other laser
  + We fixed this by putting the function in LevelBase instead of the LaserGun Class, and then called it from the Sprite instance “hero”, and also added a boolean to “skip” (through the if structure) the rest of the loop if a bullet was removed.

#### Dec 14th, 2015

* We were able to make text appear near the eye using an array of messages
* We learnt how to animate using a spritesheet - for the character and door
* We created 2 basic scratches for it “spritesheet\_explosion” and “spritesheet\_character”. The explosion scratch is a scratch for calling animations on demand (such as the door that we will have to animate) and the character one is for.. well… the character.

#### Dec 15th, 2015

* We started designing architecture for a menu screen. Currently, we have a title screen, a settings screen ( that does nothing right now), and the levels themselves.

#### Dec 17th, 2015

* We found out how to mute audio in the minim library
* We added a mute button/hotkey into the settings part of the menu - this section will also allow you to enable “master mode” in a later release.
* We started a scratch that has file input/output
  + we’ll use it tomorrow to have the user input names, and based on whether they’re have played before or not, make decisions.

#### Dec 18th, 2015

* We finished the scratch and integrated it into our main program. It works like a charm! After pressing “enter” to begin the actual game, the user must enter their name after clicking the input box.

#### Dec 21st, 2015

* Matt figured out a way to make a sidekick. He repurposed fXStart / fYStart to be the location of the hero. fX/fY are incremented/ decremented until the butterfly is at the position of the hero. Once that happens, it picks a random coordinate within 50 pixels of the hero to be the next position to move to.

#### Dec 22nd, 2015

* Today we added master mode, which was relatively easy to do. We have a boolean to activate Master mode, and if true, when you die it sets nLevel to 1 in masterMode().

**Dec 28th, 2015**

* Abhi created a class called “Window” for parallax scrolling. We were very stumped before on what to add to our game to include this feature (parallax scrolling), so we decided to go with a window in the level. Out of it you can see clouds and fog moving past the moon. We did this by translating the images at different rates relative to the “xCamOffset” which is used for scrolling.

**Dec 29th, 2015**

* Matthew created 9 more levels to add to the game and designed the architecture for falling platforms. When the user “hits” a falling platform, the boolean bActivateGravity is set to true after *n* milliseconds, and then in **draw**() we cycle through the falling platforms and call gravity() for that platform (a function in Sprite).

**Dec 30th, 2015**

* We found a way to reduce the number of lines of code in “updateCameraPosition()”. It turns out in processing there is a function called “constrain()” It works like the limit function for PVectors. So we were able to replace the code below, with only two lines!
  + We replaced this:

if (fXCamOffset <= 0) {   
 fXCamOffset = 0;  
 } else if (fXCamOffset >= nRightEdge) {  
 fXCamOffset = nRightEdge;   
 }  
  
 if (fYCamOffset <= 0) {   
 fYCamOffset = 0;   
 } else if (fYCamOffset >= nTopEdge) {  
 fYCamOffset = nTopEdge;   
 }

with this:

nXCamOffset = constrain(nXCamOffset, 0, nRightEdge);  
 nYCamOffset = constrain(nYCamOffset, 0, nTopEdge);

**Jan 4th, 2015**

* We found out that our scrolling along the y axis wasn’t being rounded, which is why our game seemed to be lagging. It seems like translate() does not like translating by float numbers with many decimal places. We have rounded this value and changed it to an integer.

**Jan 5th, 2015**

* We created an image for our falling platforms for our release Qui-Gon Jinn.
* We were able to create a spriteSheet for the hero (the crab). Our next step will be to integrate this spritesheet (with our sprite class) in a separate scratch.

**Jan 6th, 2015**

* We created a scratch called “spritesheet\_crab”.

**Jan 8th, 2015**

* After talking to you and looking at the possibilities, we have decided that we will pursue JSON to load our levels.

**Jan 11th, 2015**

* We extended Sprite to a class called “SpriteAnimated” as integrating sprites with animation in the sprite class itself would be super complicated. We were able to animate the doors, butterfly, and crab in our game using this technique.

**Jan 12th, 2015**

* We looked into JSON a little more. As variables or comments cannot be used in JSON files, and as we wanted to keep the game with as little hard-coding as possible, we decided to load the co-ordinates as strings, and then create a function in Processing that can determine what the co-ordinates are.
  + Besides the string of coordinates in the JSON file, there is also another “item” in the object that is a string of what type of coordinate it is. For example, if the type is “MultBox” then in our funciton in processing “fDetermineCoord()”, since the type is equal to MultBox, it will multiply nBoxSize by the float number in the string coordinate
  + We can do this using the JSON functions built into Processing, and splitTokens()
  + We created the scratch “loadJSONArray\_levels\_split” to be able to test out the functions mentioned above integrated into LevelBase

**Jan 13th, 2015**

* We were able to load Levels 1-9 through these JSON files.

**Jan 14th, 2015**

* Today we loaded Level 10 through 15 from a JSON file. There are many objects in Level 15, but on the bright side there are many less tabs in Processing...

**Jan 15th, 2015**

* We were able to load level 16 and 17 from a JSON file
* We decided that JSON is much more useful than a text file, so we will switch our class that uses names for text files to use JSON files. This way we can also load a death count, and a boolean for whether or not the user has gotten to Level 20 (boss level), and if they have then they start the game there.

**Jan 16th, 2015**

* We found a “saveJSONArray()” and “append()” for JSON that will allow us to use JSON to load and save user related objects.
* We created a scratch for this called “loadAndSaveJSON” that works like the “loadAndSaveStrings” scratch, except now… with JSON!

**Jan 17th, 2015**

* We designed Level 17 - 19

**Jan 18th, 2015**

* We added Level 17 in JSON files

**Jan 19th, 2015**

* We added Level 18 in JSON files

**Jan 20th, 2015**

* We added Level 19 in JSON files
* We also sorted (and removed some of) the many useless scratches that we had sitting around.
* We finished writing the Description of Scratches document that will be submitted on Friday

**Jan 21st, 2015**

* We added Level 19 in JSON file format
* We finished writing the remaining documents that will need to be in our final project package for tomorrow

**Jan 22nd, 2015**

* This is the last journal entry **)^:**
* As we never did our boss level (yet?) we have reduced nLastLevel to 20.
* We know that we have come a very long way since starting this project. It’s been fun to learn to program while making a game; not as fun as seizure inducing faces though.