Clicker Heroes

Objective

The objective here is to make an interactive game and use several operators we are learning in the process. This lab is based on the popular game, Clicker Heroes released in 2014 according to wikipedia.

Instructions

This game does not have to look like the picture above but I've included the image used here for your convenience to test your sketch.

If you see "CheckPoint" next to a step it indicates that you should be able to verify that your sketch works.

Step 1

Comment your name and determine a size for your sketch. Mine was 600, 600. Add setup() and draw(). Save your sketch with whatever name you feel appropriate.

Step 2

You will need to acquire some assets for your sketch.

1. You need to create a font. Here is a website that shows you how to create a font. I made my font size 48 but you can choose whatever font or size you would like.
2. You need to find an image or use the duck image provided. Note: Using an image that is square and lacks too much asymmetrical whitespace is ideal. We will use a simple method to calculate a hitbox on a character so the picture choice makes this easier or harder. The duck I've provided is approximately 250 x 250 pixels. Drag the duck or picture of choice into the sketch. Literally put the picture on top of your code and you'll see that an image was added to your data folder.

Step 3

Create the variables needed for your PImage and PFont. Initialize the variables in setup. Draw the PImage in the middle of your screen. Use the reference sheet to guide you on how to use these non-primitive variables. You will use the loadFont("somename.vlw") and loadImage("duck.png") methods.

Step 4

Create variables to store the health of the duck and the time elapsed. Consider what data type makes sense. Initialize your health variable to some arbitrary number. I started with 1000. Since no time has elapsed, start with 0.

Step 5 - CheckPoint

Using your font variable to set the font and the text() methods build the top portion of the sketch. This is achieved with multiple text() method calls. You will display the damage and time labels and then values that your health and time variables have below.

Run your sketch until these are placed in your desired location. Your font won't look like my font unless we chose the same font and size for our sketches.

Step 6 - CheckPoint

Overwrite the mousePressed method so that when you press the mouse anywhere on the screen, it decrements the health variable by some amount. I chose 5 arbitrarily. Use the shortcut assignment operator to this. DO NOT USE THE `mousePressed` variable. You do not need an if statement at this point. Run your sketch and notice if text information is changing at the top of your sketch. If it isn't, are you sure you are displaying the variable in your text() method call?

Step 7 - CheckPoint

Use your PImage variable to display your image at the center of the screen. Run your sketch. Do you see it? Refer back to the reference to see how to display a PIMage.

Step 8 - CheckPoint

We need to refactor the code in your mousePressed() method. We only want to decrease the health of the duck if the duck is actually clicked. There are some complicated ways to do hit boxes but we will keep it simple. We will find out how far away from the center of the picture is your mouse press to determine if there was a hit.

Before you display your image, set imageMode(CENTER). This will put the origin of the image in the center of the image. Make a variable to store the distance. You will then calculate the distance from your mouse's x and y location from the center of the image. Code the formula below.

i. $$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

ii. There is no sqrt operator or exponent operator so you must use methods to turn this formula into code. Create as many local variables as needed to get the value of the distance between your mouse and the center of the image. *You may NOT use the method dist()*

iii. pow(base, exponent) -> float num = pow(2, 3); -> num has the value of 8.0

iv. sqrt(num) -> sqrt(25) -> float num = sqrt(25); -> num has the value of 5.0

Once you have the distance calculated. Check to see if the distance is less than about half the size of the picture. The duck picture is roughly 250 x 250 pixels. What is half of your picture size?

```
if ( distance < 125 )

 {
     //decrease your health using the shortcut operator

 }
```

Run your sketch to verify. You have basically made a circular hit box. You consider reducing the distance check to reduce the size of the hit box.

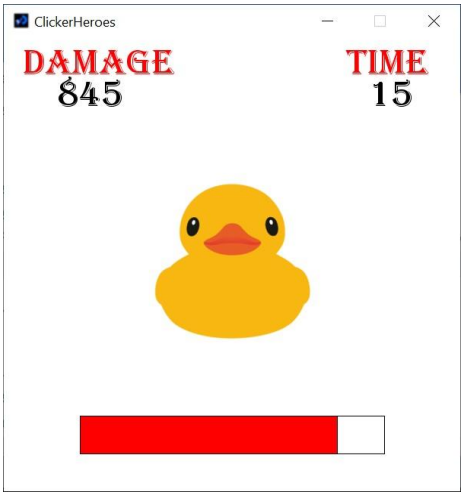
Step 9 - CheckPoint

Get your time variable to work. Every time 60 frames have passed, increment your time variable by one. Processing has a built-in variable called frameCount. It counts the number of frames that have occurred since the start of the sketch. Since processing runs 60 frames per second we can check to see if the number of frames has reached a number that is evenly divisible by 60. The remainder (mod) operator in java can be used to do this. 10 % 3 will equal 1 because 10 divided by 3 is 3 but the remainder is 1. 10 % 5 will equal 0 because the remainder is 0. Using this knowledge, write an if statement in draw that will check to see if frameCount has a remainder of 0! Equality is checked with ==. If it is then update your time variable by one. Use the right operator to update. I do not want to see time = time + 1;

Run your sketch to verify that your time increases by one.

Step 10 - CheckPoint

Add a health bar. The health bar is achieved by drawing two rectangles. One rectangle is filled with a color of your choosing and the other is noFill(). (just the border). At first, you won't see the border-only rectangle because the two rectangles completely overlap. However, you will draw the colored rectangle a percentage of the width each time based on the amount of health you have taken from the duck (or image clicked).



In order to calculate the width of the colored rectangle you must solve a proportion in your code. Remember how to solve a proportion? When I made my health bar, it was 400 pixels wide. You can choose whatever width you would like to look good for your sketch. Here is the proportion I solved

$$\frac{health}{starting\ health} = \frac{missing\ rectangle\ width}{starting\ rectangle\ width}$$

So my actual values were

$$\frac{health}{1000} = \frac{width}{400}$$

The width of my rectangle is health * 400 / 1000. If I calculate this every frame then it will draw the health bar the right proportion each time.

Run your sketch to verify.

Step 11 - End Game & Restart

So the game doesn't technically have to end. Currently we are decreasing the damage of the duck until...well it doesn't end. Here is where we would add an end game or change levels.

If the health of the duck reaches 0 then the game should stop. Special message at the end and then instruction to restart the game.

How could you change the sketch? Here is one approach:

Make a boolean variable called gameOver. Initialize it to false. We'll assume that the game isn't over initially.

Refactor the code in draw using this below

```
if ( !gameOver ) //means if the game is not over
{
     //all of the code currently in draw goes here
}
else
{
   //your new end game message goes here
}
```

Clicker Heroes

The if/else structure simply means that if something is true then do it, else (otherwise) do this.

The only thing that you need to add is another if statement in draw that checks whether health is 0 and if it is then it sets the gameOver variable to true.

Finally, overwrite the keyPressed() method so that if the user hits any key, the game is reset. What needs to be reset to restart the game?

Challenge

Feel free to add levels or effects not indicated in this lab.