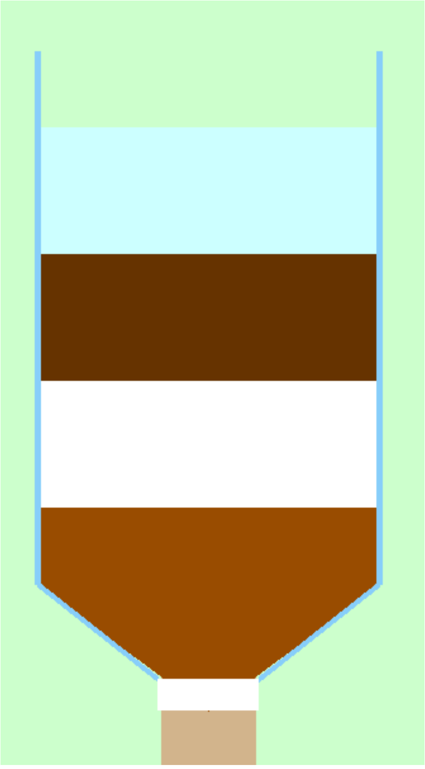
Clean Water Filter

Clean Water Simulator / Game



ICS3U
Edison Qu, with resources from Mr. Reid
April 8th, 2021

Design Section

How the game works

The intention of the Clean Water Filter is to let players a real-life idea of how to make a homemade filter in case they do not have the resources to have potable drinking water, it also highlights the things that one would not put inside a filter. With the simulator being created, the players get to choose from a vast majority of materials to create the ultimate water filter.

Materials

https://www.instructables.com/Homemade-Water-Filter/#:~:text=Homemade%20Water%20Filter%201%20Gather%20Materials%202%20Cut,Add%20Gravel%2010%20Filter%20It%20More%20items...%20

https://thebestwaterpurifiers.com/how-does-a-homemade-water-filter-work/#:~:text=On%20a%20homemade%20water%20filter%20actuated%20carbon%20to,layer%20has%20the%20least%20gap%20between%20two%20sides.

(Cheat Sheet)

Best Combination

1. Gravel - (To get rid of big chunks of dirt and rocks)
2. Fine/Course Sand - (To hold the dirt that passes through the water)
3. Cotton - (To clean the dirt)
4. Coffee filters (Helps filter dirt)

These 4 materials are the best combination for the water filter. Any combination with these 4 will create a perfect water solution.

Inadequate Combination

5. Paper - (Hold some dirt particles inside the paper)
6. Charcoal- (Trap the dirt particles between the surfaces of the charcoal)
7. Rocks - (Trap some dirt particles between the surfaces of the rock)
8. Grass - (Trap some dirt particles inside the stack of grass)

These 4 materials will create a good filter however, this will not create the best filter solution. Any of these will be proven inadequate, even if the descriptions are saying that it is effective as it might or might not perform the tasks at full efficiency.

Worst Combination

9. Dirt (Hard Mud)
10. Sticks
11. Graphite
12. Broken Glass

Any combination of this would result in a 0% win rate for the water filter.

How to win (Logic behind the rating system)

Players will be rated on how well the materials work together to filter the final outcome of the water that has a filter score higher than 75%. The materials are all worth a certain value. With the materials in the "Best Combination" group, each of them is worth 10 points. With the materials in the "Inadequate Combination" group, each of them is worth 5 points. With the materials in the "Worst Combination" group, any combination with one of them is an automatic fail as the water will be deemed not drinkable. Each of the materials in the "Worst Combination" group will have 0 points. Players will need to find the best combinations to filter the water.

Prototypes

The first prototype was in Checkpoint #1 where everything was just lines instead of boxes, the screen size was (1200,900), and everything was hardcoded meaning that none of it made sense. (I could not retrieve it anymore)

The second prototype was in Checkpoint #2 where it was realized that one should not use for loops to draw the boxes as it was far easier to use classes and objects as each of the materials have its own characteristics.

There are many things that were wished to be added but did not, because of how tight the time frame was.

Examples are an opening screen, telling the players what they need to do, and a storyline.

Attached below are the designs for the code.

UML CLASS DIAGRAM

Class:	Boxes

Attributes:	X =int
	Y = int
	Color = (int,int,int)
	Height = int
	Width = int
	Box color = (int,int,int)
	Size = int
	Text = string
	Score = int
	Added = boolean

Methods:	draw(): void mouseclick(pos): boolean drawtopbottle():void drawtopmidbottle():void drawbotmidbottle():void drawbotbottle():void

Inheritances:	sand = Boxes()
	gravel = Gravel()
	cotton = Cotton()

	coffeeFilter = coffeeFilter()
	paper = Paper()
	charcoal = Charcoal()
	rocks = Rock()
	grass = Grass()
	dirt = Dirt()
	stick = Stick()
	graphite = Graphite()
	brokenGlass = brokenGlass()
	go = Buttons()
	remove = Buttons()

Class	Waterbottle

Attributes	X =int
	Y = int
	X1 =int
	Y1 = int
	X2 =int
	Y2 = int
	Color = (int,int,int)
	Height = int
	Width = int
	capcolor = (int,int,int)
	Size = int
	Cap size = int

Methods	Draw_body() = void Draw_neck() = void

Class	Scoreboard

Attributes	X =int
	Y = int
	Color = (int,int,int)
	Height = int
	Width = int

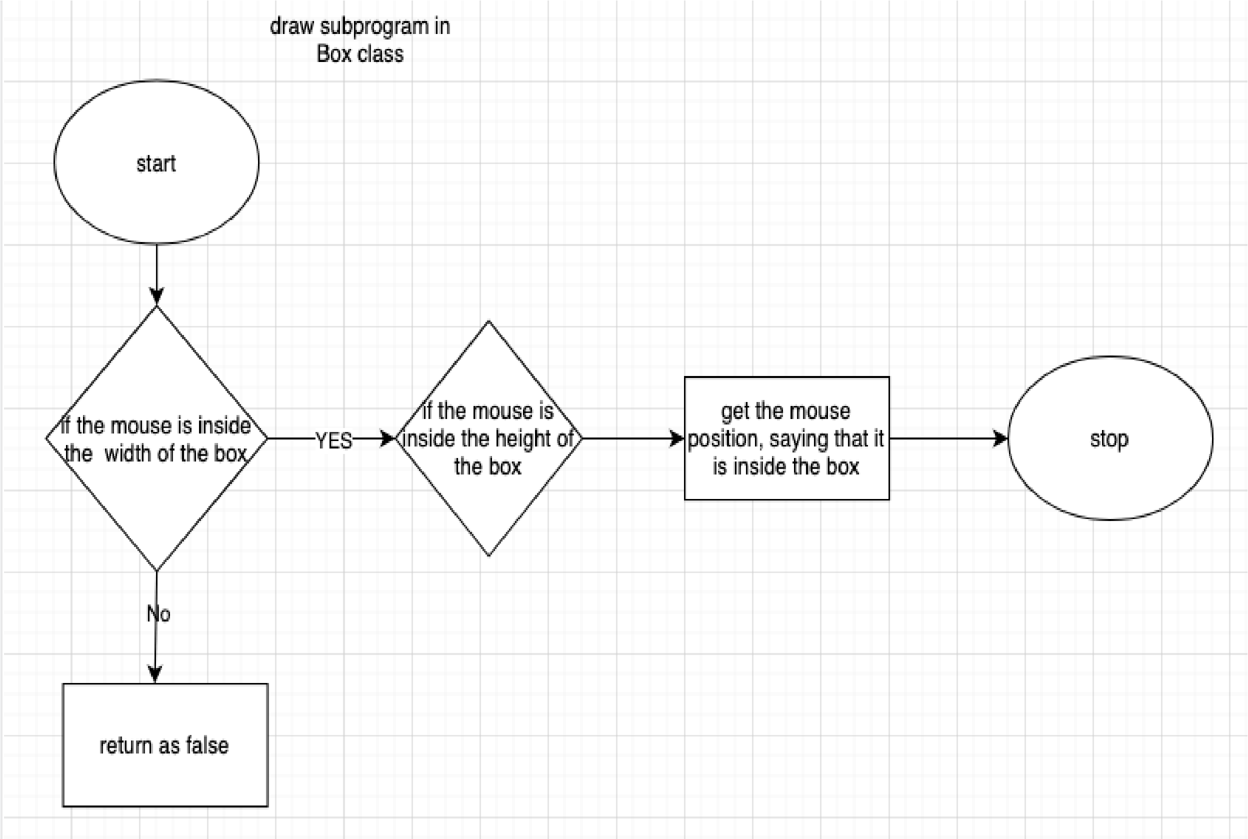
methods	displaySCores(win) = void

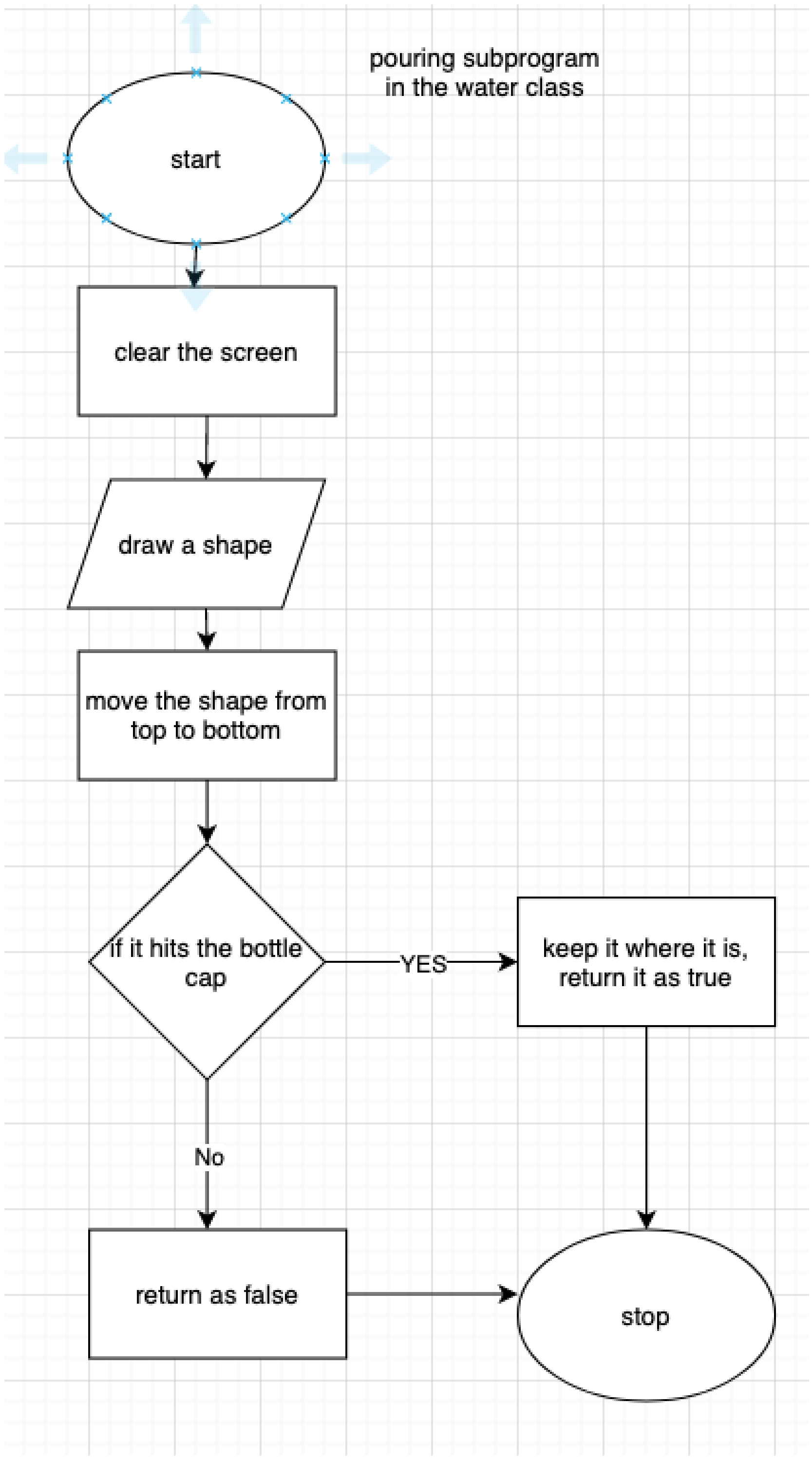
UML State Diagrams

Boxes	sand
Attribute	Value
x	15
y	15
height	80
width	150
boxcolor	(204,255,255)
size	0
text	‘sand’
totalScore	10
added	False

Waterbottle	waterbottle
Attribute	Value
X	265
x1	535
x2	360
y	40
y1	460
y2	440
color	(204,255,255)
size	5
capsize	(204,255,255)
capcolor	(204,255,255)

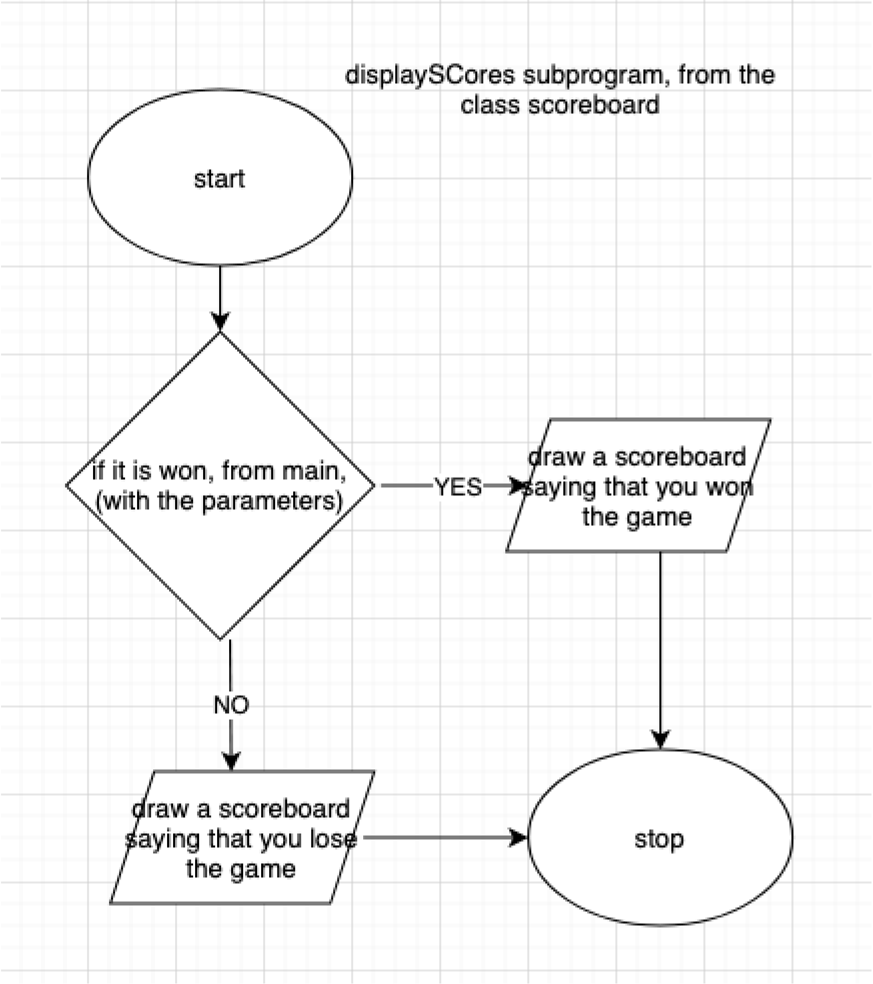
Scoreboard	perfect
Attributes	Value
x	500
y	100
color	(255,255,255)
height	400
width	200


draw subprogram in Box class
start
if the mouse is inside the width of the box
YES
if the mouse is inside the height of the box
get the mouse position, saying that it is inside the box
stop
No
return as false

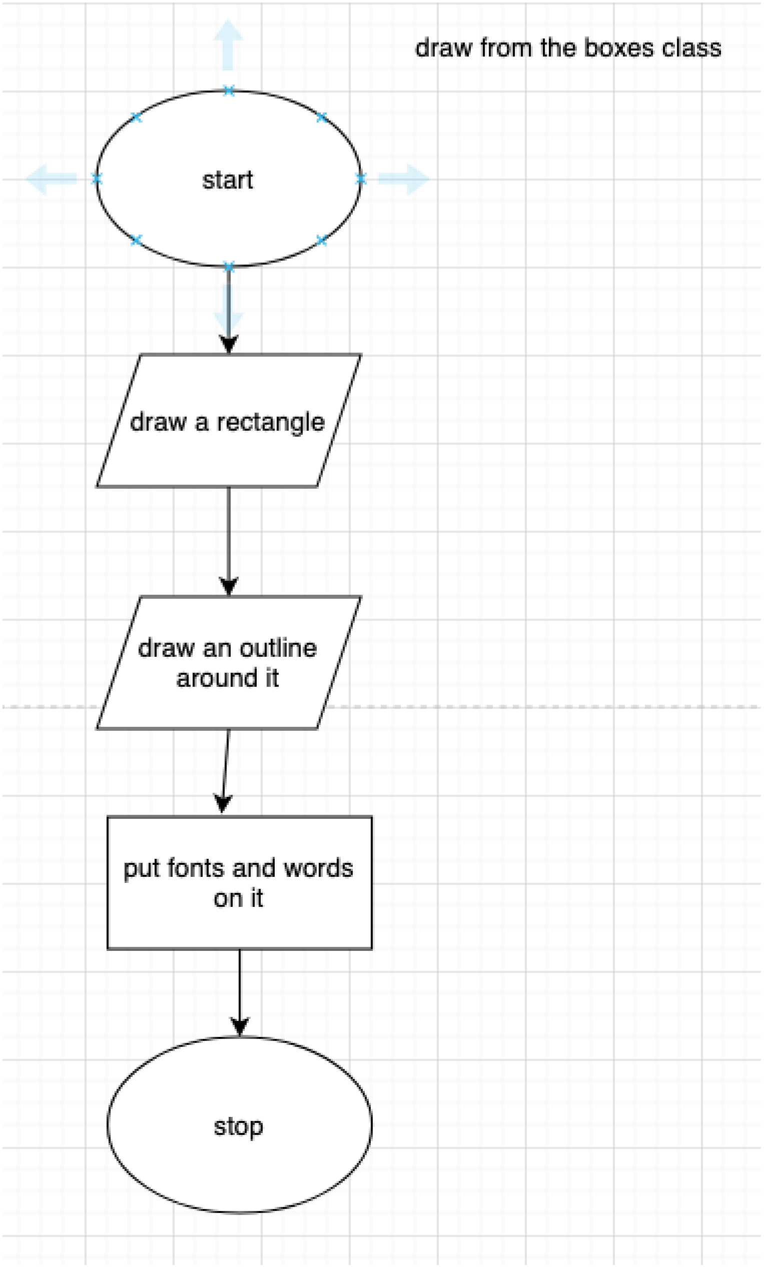


pouring subprogram
in the water class

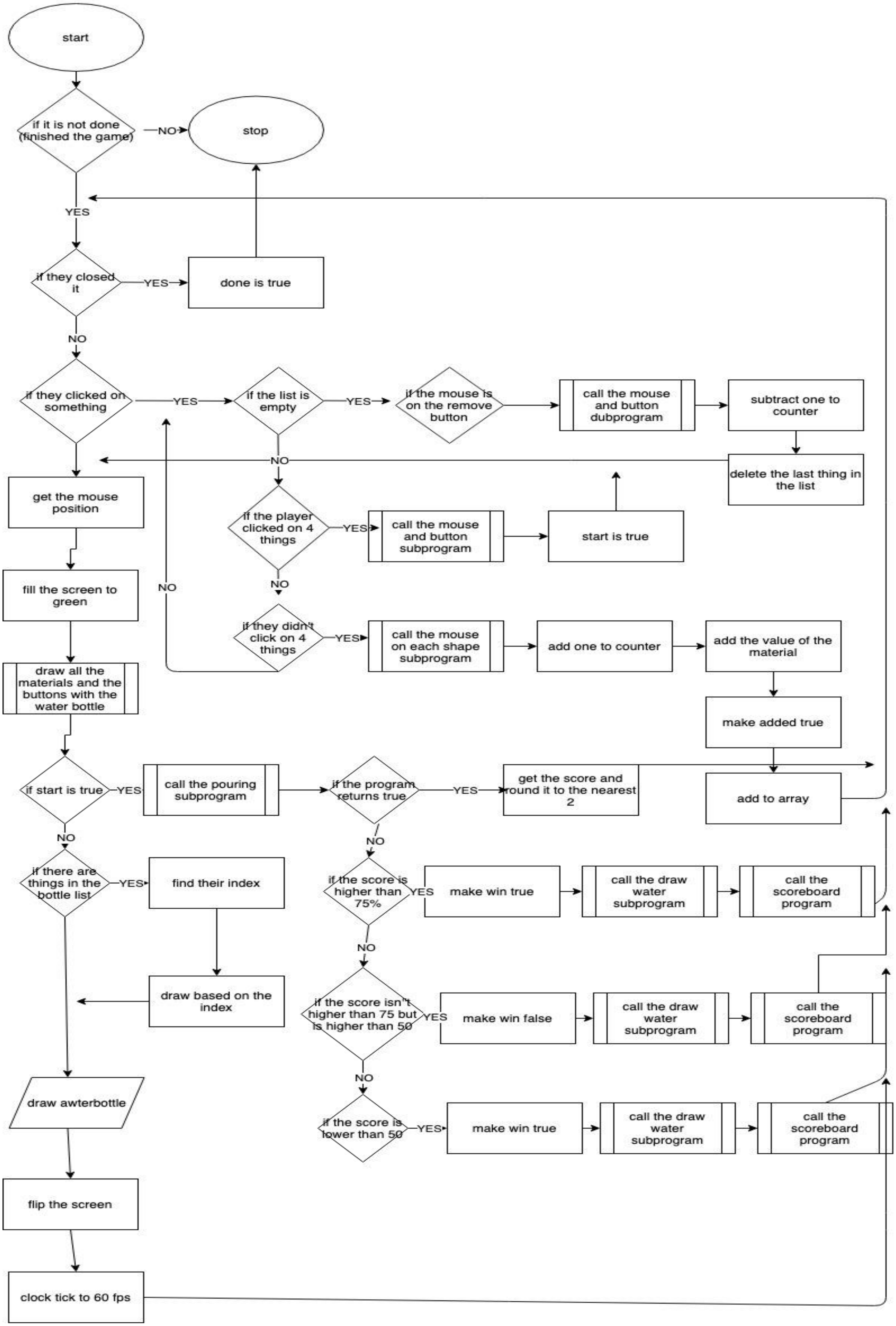
start
clear the screen
draw a shape
move the shape from top to bottom
if it hits the bottle cap
YES
keep it where it is, return it as true
No
return as false
stop


displaySCores subprogram, from the class scoreboard

start
if it is won, from main, (with the parameters)
YES
draw a scoreboard saying that you won the game
NO
draw a scoreboard saying that you lose the game
stop


draw from the boxes class
start
draw a rectangle
draw an outline around it
put fonts and words on it
stop

main program


start
if it is not done (finished the game)
NO
stop
YES
if they closed it
YES
done is true
NO
if they clicked on something
YES
if the list is empty
YES
if the mouse is on the remove button
call the mouse and button dubprogram
subtract one to counter
NO
delete the last thing in the list
get the mouse position
if the player clicked on 4 things
YES
call the mouse and button subprogram
start is true
NO
NO
fill the screen to green
if they didn't click on 4 things
YES
call the mouse on each shape subprogram
add one to counter
add the value of the material
draw all the materials and the buttons with the water bottle
make added true
if start is true
YES
call the pouring subprogram
if the program returns true
YES
get the score and round it to the nearest 2
add to array
NO
NO
if there are things in the bottle list
YES
find their index
if the score is higher than 75%
YES
make win true
call the draw water subprogram
call the scoreboard program
NO
draw based on the index
if the score isn"t higher than 75 but is higher than 50
YES
make win false
call the draw water subprogram
call the scoreboard program
NO
draw awterbottle
if the score is lower than 50
YES
make win true
call the draw water subprogram
call the scoreboard program
flip the screen
clock tick to 60 fps

Implementation Section

The Clean Water Filter Simulator is a prized simulator for people playing it and also for the coder making the game. The Clean Water Filter Simulator is a simulator where the player chooses 4 materials from 12 options to create the best filter they could make. This simulator was the perfect fit for the Culminating Performance Task, as it is believed to be the perfect fit for all the requirements. For example, the use of a list, the most important piece of code that is needed in order to create this simulator, as it not only stores all the materials in the bottle but it makes it easier to manipulate and draw out. Moreover, the use of objects and classes was a huge help as it was very easy to sort everything out and organize it, as well as shortening the code written. Selection was important and implemented as there was a different feature in each box when clicked on. Repetition was needed in order to go through the list easily instead of writing long pieces of code. Subprograms were needed as it made it a lot easier to trigger events like clear the screen and draw certain types of shapes.

However, there were a lot of things that needed to be done in order to complete this seemingly small simulator. There were many times where the past material needed to be brought up to have a brief refresher on how to use graphics and lists. Moreover, there are also times where code had to be searched up as there was no other efficient way to make the font centralized. Finally and most importantly, the use of buttons was something that is very enjoyable to learn as it was very difficult and needed a lot of logic in order to be coded correctly.

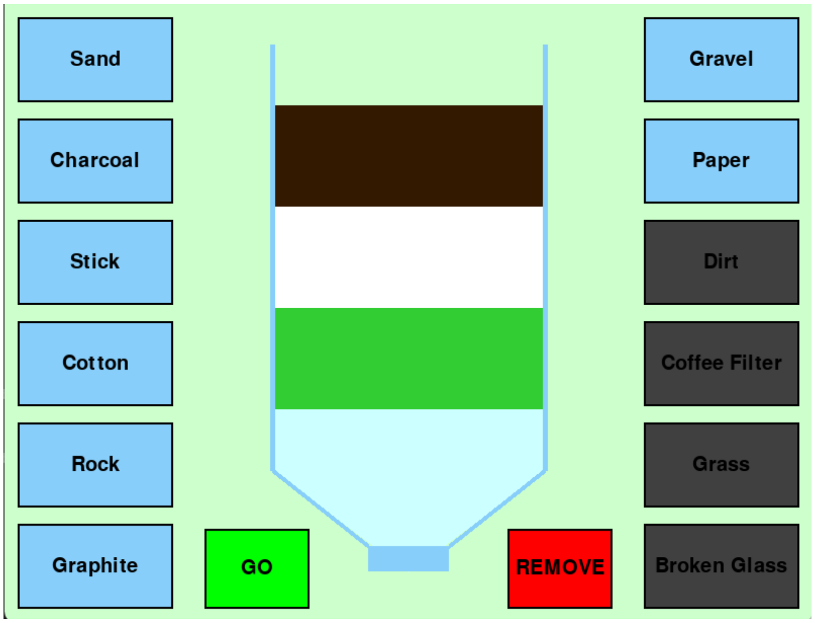


Figure 1, the usage of lists and buttons as the materials need to be stored in the list, and the player has to click on those buttons to put them in the list.

Testing Section

Testing was a very important part of the creation of the Clean Water Simulator as there were a lot of things that needed to be tested because it did not work the way it was intended.

There were 3 important methods to test out code:

1. Debugger
2. Putting print statements to see if it runs
3. Ask someone to do the two things above

In this program, all of those were tested in the simulation.

The debugger was used to solve so many problems that occurred. One example was when the 'Remove' function was implemented, where the total score would be up in the hundreds instead of 40.

```
for eachScore in range(len(inBottle)):
   totalScore += inBottle[eachScore].totalScore
```

This code messed up the whole program to the point where there were thoughts of removing the whole 'Remove' function out completely. When using old reliable, the debugger, the total score was being multiplied by 3 every single time something was clicked. This was fixed when this code was implemented.

```
totalScore -= inBottle[-1].totalScore
```

Alongside this code for each of the objects.

```
totalScore += (object name).totalScore
```

The use of a 4th eye was important as my professor, Mr. Reid, has helped me debug a problem with the 'GO' square lighting up. What was intended was when there were 4 clicks on the materials, the GO button would light up to green and become function from being gray. The problem was that the GO button would light up as long as there was something clicked inside the program, as seen below.

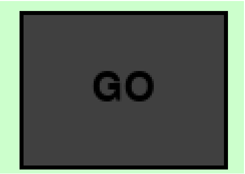


Figure 2, This is when it is not lit up and the user did not choose 4 materials.

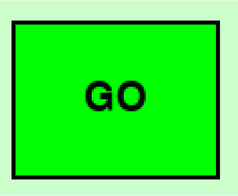
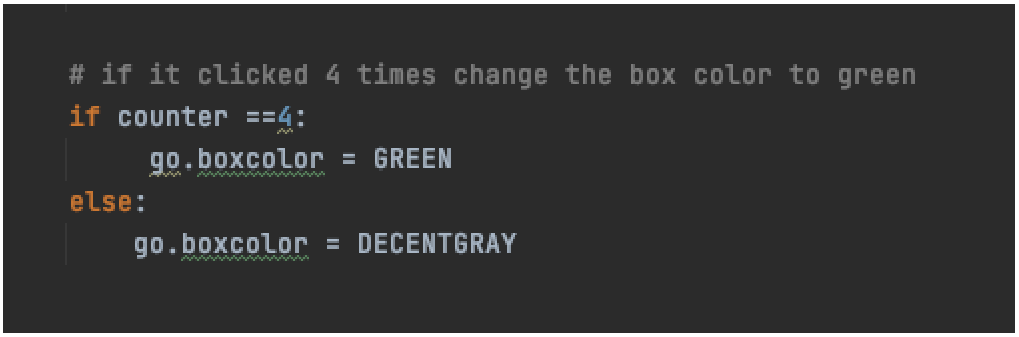


Figure 2.5, This is when the user clicks 4 materials but would happen if they choose 3 things and click something random on the screen

In order to fix this, my professor and I went on a debugging spree using his preferred IDLE, ‘PYZO’, where he debugged the whole program and found out the problem was that it was implemented way too early in the program and was fixed just by putting it to the very end of the if selections.



```
# if it clicked 4 times change the box color to green
if counter ==4:
     go.boxcolor = GREEN
else:
    go.boxcolor = DECENTGRAY
```

Figure 3, This was the correct, award-winning code that made the problem go away with the help of 4 eyes and a debugger.

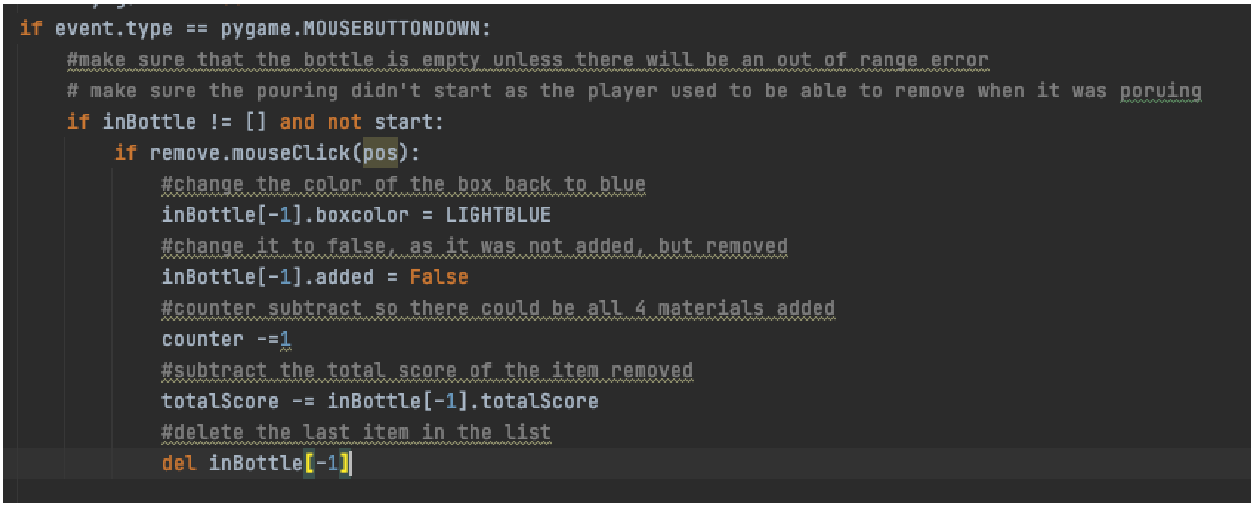
Analysis Section

Clean Water Filter Simulator is one of the best projects to work on for someone who is beginner or intermediate at coding. It makes the coder develop a strong sense of essential computer science skills such as logical thinking (getting the mouse position and practicing sequence) and a strong sense of how to use code (making and manipulating arrays).

This project went well and everything that was put in the proposal was added into it including extra features, such as the remove button. Everything went very well, the code works, and there are really no bugs inside the code.

Things that are notable (proud of)

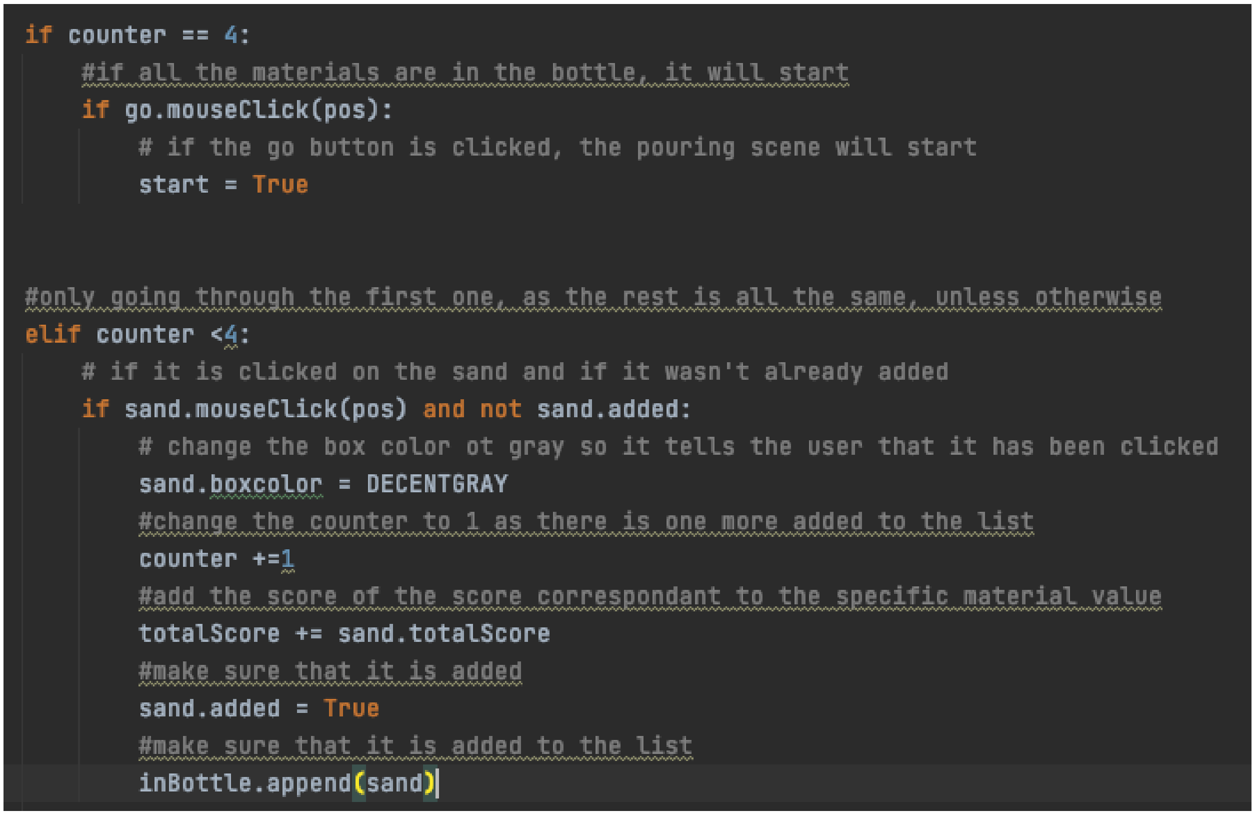
My most proud work in this project is how the remove button was used and how flawless it is. With the variables used to stop bugs in the remove button such as making sure that there was an empty list and if it did not already start.



```
if event.type == pygame.MOUSEBUTTONDOWN:
    #make sure that the bottle is empty unless there will be an out of range error
    # make sure the pouring didn't start as the player used to be able to remove when it was poruing
    if inBottle != [] and not start:
        if remove.mouseClick(pos):
            #change the color of the box back to blue
            inBottle[-1].boxcolor = LIGHTBLUE
            #change it to false, as it was not added, but removed
            inBottle[-1].added = False
            #counter subtract so there could be all 4 materials added
            counter -=1
            #subtract the total score of the item removed
            totalScore -= inBottle[-1].totalScore
            #delete the last item in the list
            del inBottle[-1]
```

Figure 4, the incredible use of arrays and the errorproof selection such as making sure the list is not empty and the pouring did not already start, make it a true piece of art.

Another code that is notable is how well the buttons and boxes were implemented into the simulator, as a beginner coder, it brings joy to the eye when seeing the code run flawlessly without error and does what it is supposed to do.



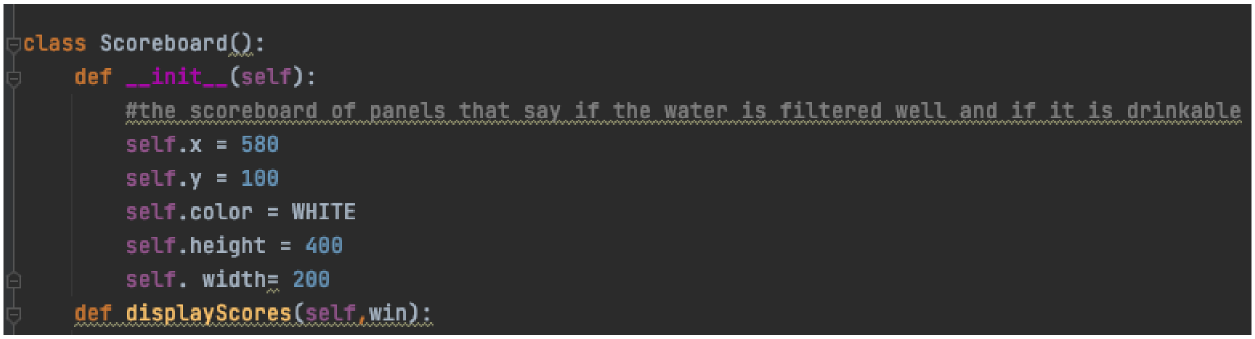
```
if counter == 4:
    #if all the materials are in the bottle, it will start
    if go.mouseClick(pos):
        # if the go button is clicked, the pouring scene will start
        start = True


#only going through the first one, as the rest is all the same, unless otherwise
elif counter <4:
    # if it is clicked on the sand and if it wasn't already added
    if sand.mouseClick(pos) and not sand.added:
        # change the box color ot gray so it tells the user that it has been clicked
        sand.boxcolor = DECENTGRAY
        #change the counter to 1 as there is one more added to the list
        counter +=1
        #add the score of the score correspondant to the specific material value
        totalScore += sand.totalScore
        #make sure that it is added
        sand.added = True
        #make sure that it is added to the list
        inBottle.append(sand)
```

Figure 4.5, Shows the code of how foolproof it is to make an error, with making sure that the counter is less than 4, adding one to the counter inside the sand object, making sure that it was not added before, and the appending of the sand, truly proves that this code is just majestical.

Things to work on

There is no such thing as perfect code in a project that is half a thousand lines of code for a small simulator. There are many things that could have been worked on if it was not for the tight time struggle and deadline of 2 weeks.

The first thing was how badly organized the classes and objects are (ironic isn't it?) There was one thing that was annoying and that was the fact that the scoreboard class was created when it could have been inherited from the Boxes() class as it is still boxes.



```
class Scoreboard():
    def __init__(self):
        #the scoreboard of panels that say if the water is filtered well and if it is drinkable
        self.x = 580
        self.y = 100
        self.color = WHITE
        self.height = 400
        self. width= 200
    def displayScores(self,win):
```

Figure 5, shows that the attributes are the exact same as the Boxes() class that would have been neater as they are still both boxes.

Another thing that could have been worked on was the quality of life. *What does this mean*? This means that there could have been features added or modified to help the player out. Something that was missed completely was the introductions and telling what the player should do, as right now, they are just left there playing around not knowing what the objective is. Another thing that could have been worked on but did not because of the tight time period was a try again button so it would be easier for the teacher and/or player to try again with a different combination.

References

In this project, there were not a lot of things that were taken from a third party other than learning from their material/source, however, it is still cited alongside some other third-party websites that have been used for increased efficiency.

How to centralize text and fonts in pygame
https://stackoverflow.com/questions/23982907/pygame-how-to-center-text
How to create buttons in pygame and outlines
https://www.youtube.com/watch?v=4_9twnEduFA

Learning Sources:

Mr. Reid's ICS3U Class
Mr. Reid's Demos
Mr. Reid's labs
Program arcade games
http://programarcadegames.com/index.php
Simpson College Computer Science

Waterloo CEMC website
https://cscircles.cemc.uwaterloo.ca/

Invent your own computer games with Python- Al Sweigart, 4th edition Book