Assignment Cover Letter

(Individual Work)

BINUS UNIVERSITY INTERNATIONAL

Student Information:	**Surname**	**Given Names**	**Student ID Number**
1.	**Tandra**	**Vincentius**	**2301894804**

Course Code : **COMP6502**

Course Name : **Introduction to programming**

Class : **L1AC**

Name of Lecturer(s) : **Ida Bagus Kerthyayana Manuaba**

Major : **CS**

Title of Assignment (if any) : Shoot the Target

Type of Assignment : **Final Project**

Submission Pattern

Due Date : **14/01/2020**

Submission Date : **14/01/2020**

"Shoot the Target"

Name: Vincentius Gabriel Tandra

ID : 2301894804

I. Description

The function of this program:

This program is an incremental game also known as an idle game which I named Shoot the Target. This game was made with the intention of making a simple game that can be played to reduce boredom, it does not take much work to play it and so is not a game that serves to be a distraction or one that is too physically or emotionally echausting The players of the game will have a target at the center of the screen and money which will go up whenever it is clicked. On the left side of the screen is a store menu which can be used to buy items with the money obtained from clicking the target. These items each work passively and give money every second in the background and have a default price which goes up with every purchase. This game has no storyline or ending and so can be played whenever, even if you get bored of clicking, you can let the game's passive income make money for you and stay satisfied watching the numbers get bigger and bigger. The reason I created this game is to make a game that is very simple and playable when bored that doesn't require much work and so does not distract somebody when played unlike other games which require much of a player's concentration when played.

II.a. Design/Plan

The game is made up of a number of files, these are the main file (project.py) , the different classes, which used are in different files and accessed by importing them and their respective variables from each other and a text file that contains all of the data. Since, the game uses text and numbers which change constantly, this text file is needed so it works dynamically as it does.

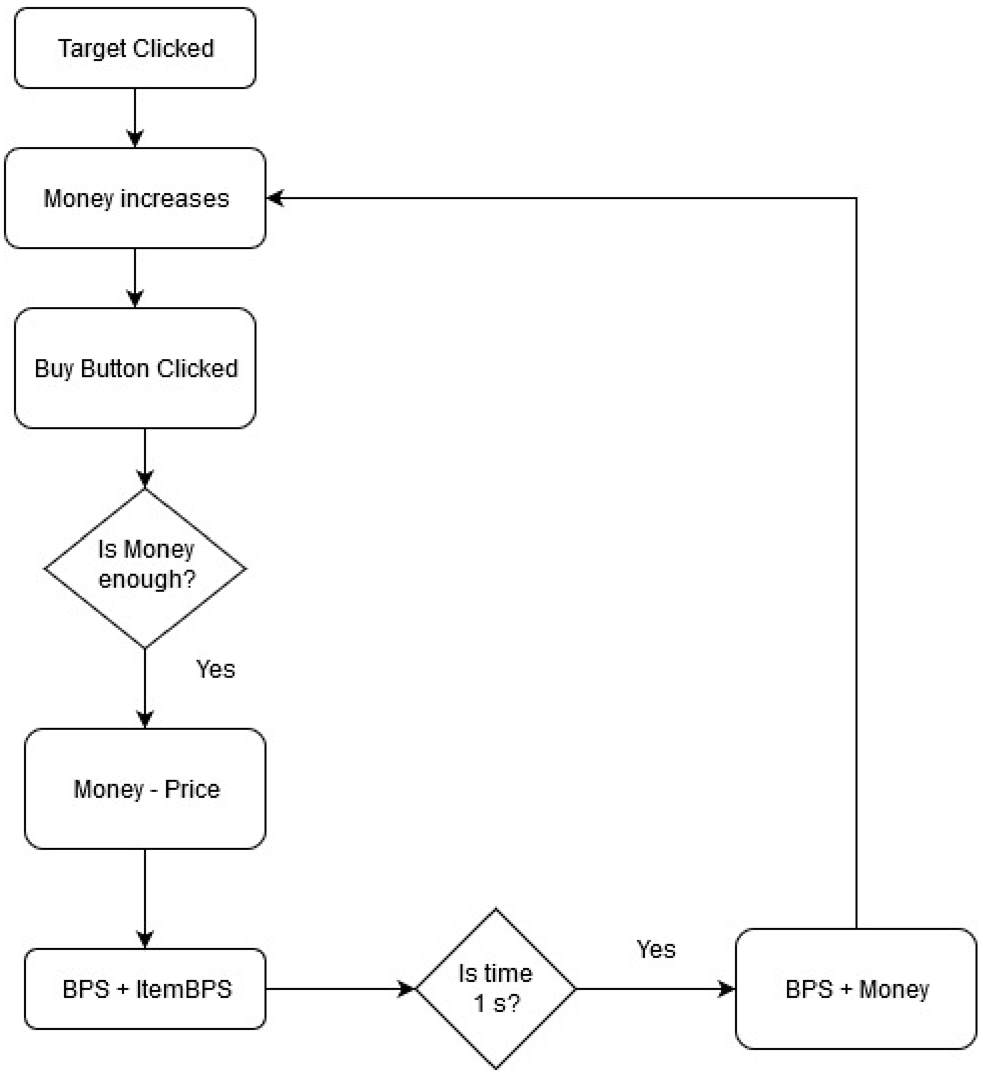
First a few things have to be done, the game window is made, a caption and the display icon and image is loaded, the audio files are also loaded.

3 classes are used, the **Target** class, **Menu** class and the **StoreMenu** class, each are used to define the target, money, the store menu as a whole and a class used to define instances of the store items and is used to draw in the elements of the in game store menu. These classes are then split into 3 files so they can be more easily read. Incremental games use a feature known as passive income and so this passive income which is defined as **BPS**(bullets per second) is given a default value of zero and increases when an item from the store menu is bought, this income gives money every second and so a time counter is established that adds the BPS amount every time a time counter reaches 1000 milliseconds or 1 second, this process then repeats for the duration that the game. More detail about the use of each class is specified in the section below

In the main game loop, a math equation is used to confine the clicking of the target to the boundaries of the image, so that players cannot click outside of the target to gain money, two functions, used to draw the target image and to continuously update the store’s values are called.

In the event handler where all the magic happens, if the game is closed, it saves all of its data by rewriting the current values into the text file. There is a button click event where money increases by 1 when clicked, an audio sound is also played. After that, more button click events are made where each one corresponds to when a store menu item is purchased. When clicked, the menu item’s BPS is added to the default BPS and a sound is played. Additionally, the price of that item goes up whenever it is bought and continues to do so with every consecutive purchase.

Flow Diagram:



II.b. Explanation of Each Function inside each Class

Target.py

Menu.py

StoreMenu,py

Class Target:

Init (self,x,y,targetValue) function :

Here the **x** and **y** positions of the clickable target are defined and the **targetValue** property of the target which is the amount of money you get per click when the target is clicked once. Additionally, a new variable known as **shotCount** is added and set to its default value which is 0, it is the number that displays the amount of money on the screen for the entirety of the time the game is running—e.g., if a player clicks once, the **shotCount** increases to 1 and when a store item is bought, this **shotCount** value goes up by the amount per second one item is worth. Lastly, when an instance of this class is created, two functions which will be explained further below are called. These functions are named **shotCounter** and **draw** respectively **shotCount(self) function :**

This function opens a text file containing all of the data used for the game, and reads all of it as a variable known as **text**, every value of data is written in separate lines and so each on is split by a new line. After that, each piece of data which is written in a format containing a string, a colon not separated by a space and empty strings is replaced so only the integer value of each piece of data is left and then each piece of data is assigned a variable so it can be used in different parts of the game

draw(self) function :

Here the target object, background color is added and additional text is drawn into the code by blitting each one, the **shotCount** value is also formatted to separate by comma after reaching 4 digits, --e.g., 999 becomes 1,000 and not 1000 so that larger numbers are easier to read.

Class Menu:

Before the class is made, two functions are created here, one to create text and another to create rectangles using pygame to make the process much easier and faster. The purpose of this class is to define each item's properties and how each will be drawn into the window

Init () function :

Here, all of the properties of each store item is made. These include their **name**, **description**, **stock price**, rectangular coordinates and their buy button outline. Additionally, each piece of text's x and y coordinates are defined within this function.

Draw(self) function:

Within this function, all of the rectangles and text is drawn in, from the inner rectangles and their outlines and then each piece of text.

Class StoreMenu:

Init(self) function:

Here, each store item is initialized as an instance of the **Menu** class and is given their properties. This is done so they can use the functions within the **Menu** class and since each item shares the same property but with different values, it is easier to define them as class instances.

Update(self) function:

Here the **draw()** function from the **Menu** class is called for each instance created in the **init()** function and then each of the updating text values is reassigned the variables contained within the target class which represent the values in the text file. This is done so that the text can update when the events in the game happen. There was an issue using this function to update the values and this was necessary for the game to work properly. More about this issue will be explained in deatil below.

III.a. Lessons that Have Been Learned

1. ***Learning the ins and outs of pygame***

   After making this project, I have learned the basics of using the pygame library such as creating the display, creating shapes, inserting text, audio and making use of the events. It was fun to make because coding in general is not a very visual experience and learning how the logic and numbers translate into making a game was a very refreshing learning experience

2. ***One use of the math library***

   At first, i learned that pygame can store coordinates in rectangles using the pygame.Rect method. However, it cannot be used to store circular coordinates which the target used, will be a circle. After looking it up, I learned that there's a way to calculate the boundaries of a circular object using a mathematical formula. The mouse click event is then confined to the circular object's circumference so that the target can't be clicked anywhere else but on the target itself. I find this useful because it might come in handy in the future.

3. ***One use of file handling***

   I wasn't very aware of the use of file handling or in this case text files besides for the exercises that we do in class but without it, I wouldn't be able to make dynamic text for the game that also saves when the program is quit. By being able to apply this programming concept in making this game, I better learned how to use it and how I might be able to use it in the future.

4. ***Further implementation of classes***

   After the demo and evaluation of my game, I was criticised for the use of redundant variables and repeating code within the **StoreMenu** and so was advised to use a class to represent each store item within the. It was far too much work to add new items and it would be far more efficient in general to use a class. After remaking most of that code into a class, my other classes needed to change to adapt to this change as well. After all problems were fixed, the end product was far more organized and easier to understand. Previously, about 60 lines of code was needed to add a new item to the menu but now all that needs to be done is to create and instance of a class.

III.b. Problem that Have Been Overcome

When I first started on this project, without very much reference I was only able to create a circle which did something, I found it difficult at first to conceptualize the code beyond that point. I needed to add a way to be able to display dynamic text, change the circle into an image to suit the game and more. The dynamic text was one problem I had no idea how to fix and so I took reference from a YouTube video making a similar type of game. I learnt that using text files allowed this and also to be able to save the game data so it wouldn't restart whenever closed, I also was unable to decide on a window size for the game and liked the size used in the video. The design aspect of the game was also somewhat difficult to come up with since I don't consider myself the most creative person in that department and so I got my brother to make the target image and took reference from other popular incremental games such as Cookie Clicker and the YouTube video for things like the store menu to make things more interesting.

After evaluation of my code, I started work on implementing classes to replace my redundant and repeating code. It was quite difficult to do but once I understood that each repeating part uses the same code with different values, I was able to begin making a new class for each item. After this implementation, an issue within my code appeared. The text within the game's store menu did not update. At first, I thought of quick fixes because I had encountered this issue in the past and it involved the text having to be rendered within the update function of the **StoreMenu** class instead of being initialized there so that it would be drawn constantly within the main loop of the game. However, at the time I was using simple variables to do this and now that two classes were being used instead, this rendering of text was done within a function of another class. I realized the problem was caused within the instance creation which was inside the init() function inside the **Menu** class. It was using the values within the text file for properties such as their stock. However, when that stock increased its number stays the same within the game despite the value in the text file already updating creating an odd visual glitch where the number on screen does not fit the actual number being used. After a lot of time was spent trying to fix this problem, I realized that the **update()** function within the **StoreMenu** class which calls a **draw()** function from the **Menu** class uses the values first defined when the class instance is created and continues to use that value despite the text already updating.

For example, if the stock of the slingshot item is currently 0 and its price is 15 and the button to buy a slingshot is clicked, the event happens as it does and the text file updates anot now that stock is 1 and its price is 40 . However, the text on screen does not change, this is due to the draw() function using the 15 value defined within the instance which is static and continues to do so. If the program is closed and reopened, it will then use the current value of stock and price (1 and 40) but will also only stay at that value. I figured out how to fix this simply by updating the properties within the instances by re-assigning them to the values in the text file so whenever the **update()** function is called, each property's value is changed to the proper current value and the proper number is drawn on screen.

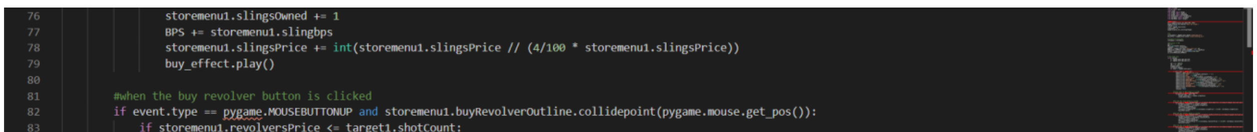
Resources :

- https://www.youtube.com/watch?v=9Wk4gTHucYU
- https://stackoverflow.com (used for equations, fixing errors and other pygame issues)
- https://orteil.dashnet.org/cookieclicker/ (inspiration for game, design and price adjustment)
- https://www.dafont.com/press-start.font (for font used for ingame text)

- Timotius Ariel Tandra(target image, color choice)

V. Evidence of code

Video demonstration of code is contained in the github link as code.mp4



```
                storemenu1.slingsOwned += 1
                BPS += storemenu1.slingbps
                storemenu1.slingsPrice += int(storemenu1.slingsPrice // (4/100 * storemenu1.slingsPrice))
                buy_effect.play()

        #when the buy revolver button is clicked
        if event.type == pygame.MOUSEBUTTONUP and storemenu1.buyRevolverOutline.collidepoint(pygame.mouse.get_pos()):
            if storemenu1.revolversPrice <= target1.shotCount:
```