Odd Semester (2021)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work****)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |
| **Student Information**: **Surname** | | | | | **Given Names**  **Felix** | | **Student ID Number**  **2101693851** | |
| 1. | | **Anggara** |  | |
|  |  |
| **Course Code** | **: COMP6502** |  |  | | **Course Name** | | **: Introduction to Programming** | |
| **Class** | **: L1BC-BLK** |  |  | | **Name of Lecturer(s)** | | **:** 1. Minaldi Loeis | |
|  |  |  |  | |  | | 2. Jude Martinez | |
| **Major** | **: CS** |  |  | |  | |  | |
| **Title of Assignment**  (if any) | : Bomb Disarming | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |
| **Due Date** | **: 7-11-2017** |  |  | | **Submission Date** | | **: 7 -11-2017** | |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Felix Anggara)

1. Felix Anggara

**Introduction to Programming**

**Final Project**

**Given Name :** Felix **Surname :** Anggara

**Major :** Computer Science **Student ID :** 2101693851

**Batch :** 2021 **Class : L1BC-BLK**

**Course Code :** COMP6502 **Lecturers :** Minaldi Loeis

Jude Martinez

**Project Name :** Bomb Disarming **Built in :** Python

**Project Description :** A game made using Python language. The goal in the game is to disarm the bomb in 30 seconds.

**Purpose :** The purpose of the game is to train concentration of the player through using the unrepeated sound clue.

**Additional Packages :** PyGame, cx\_Freeze

**Statement :**

BINUS University strictly prohibit any form of cheating, including direct or indirect cheating, plagiarism, collusion, bribery, etc. as the violation of this prohibition will result in severe punishments, including loss/drop of marks, fail in related course, and expulsion from the university. By signing this assignment at the bottom of this page, I declare that:

1. All the work included in this project is my own work and doesn’t plagiarize from any of other people work
2. All the work included in this project is my own idea, not stealing other people idea.
3. I promise that all the works won’t resubmit this project for another assignment

Signature:



Felix Anggara

1. **Purpose of the program**

The purpose of this game is to train the player to concentrate, especially in listening. The reason is because many people nowadays don’t really focus on listening others, especially in presentation, lecture, seminar, et cetera. Many of them tend to sleep or watch without knowing the context. With this game, I want to help people to concentrate in listening to something before responding.

1. **Flow Chart Diagram (Chart of How the Program Works)**

Main system running

START

Run Main Function

Main Function

Initialize pygame

Set the screen display to 1200x800

Get the display rectangle

Get home background from home background class

Get in-game background from in-game background class

Set the timer unit to second

Initiate sprite list to store buttons

Get comparing list and ID list from ID class

Next Page (1)

Get start button from start button class

Get exit button from exit button class

While True

Game Start =?

Show menu =?

Close button pressed?

END

Button list is empty?

Click on=?

END

Game Start = True

Show Menu = False

Start button

Exit button

Create buttons and every button has different Y position (cartesian) with others. Assign different sounds and ID to each button randomly (The sound won’t be assigned to more than 1 button)

Loop 8 Times

Play the first sound based from the ID

Activate and reset timer (Limit 30 seconds)

Loop

Done

Yes

No

Previous Page (1)

Game Start True

Show Menu False

Game Start False

Show Menu True

Yes

Get mouse cursor position; Show timer

No

Mouse is clicking or hovering on button

Check every buttons

Play the sound that assigned to the button ID

Match with the hover position

Not match with the hover position

Hovering

Check every buttons

Get the desired ID from comparing List

Clicking

Check clickedbuttons

Not Match

Stop timer; Print “Game Over!” on game screen

Game Start = False

Show Menu = True

Reset IDs, lists, and sounds

Match

Next Page (2)

Reduce timer value by 1 second

Next Page (4)

Next Page (3)

Previous Page (2)

Remove Button from button list

Remove ID and comparing ID

Check if the button list is empty or not

Previous Page (3)

Play the next sound based from ID comparing list

Stop timer; Print “Congratulation!” on game screen

Game Start = False

Show Menu = True

Reset IDs, lists, and sounds

Timer Value > 0

Previous Page (4)

Stop timer; Print “Game Over!” on game screen

Game Start = False

Show Menu = True

Reset IDs, lists, and sounds

Yes

No

1. **Class Diagram**

|  |
| --- |
| Randomizing |
| -idassign: int |
| +Randomizing()  +ordered()  +numbers()  +audioassign()  +modify(): int  +soundassign(idassign: int)  +removing() |

|  |
| --- |
| Button |
| -screen: pygame.display.set\_mode((1200,800))  -ident: int  -looping: int |
| +Button(screen: pygame.display.set\_mode((1200,800)), ident: int, looping: int)  +create() |

|  |
| --- |
| pygame.sprite.Sprite |
|  |
|  |

|  |
| --- |
| Timing |
| -screen\_box: tuple int  -screen: pygame.display.set\_mode((1200,800))  -stat: Status()  -buttons: sprite group  -numbering: Randomizing() |
| +Timing(screen\_box: tuple int, screen: pygame.display.set\_mode((1200,800))  +resettimes()  +decrement(stat:Status(), buttons: sprite group, numbering: Randomizing())  +blittimer() |

|  |
| --- |
| Status |
|  |
| +Status() |

|  |
| --- |
| Music |
| -checker: string |
| +Music(checker: string) |

|  |
| --- |
| Bggame |
| -screen\_box: tuple int  -screen: pygame.display.set\_mode((1200,800)) |
| +Bggame(screen\_box: tuple int, screen: pygame.display.set\_mode((1200,800)))  +blitbg() |

|  |
| --- |
| Bghome |
| -screen\_box: tuple int  -screen: pygame.display.set\_mode((1200,800)) |
| +Bghome(screen\_box: tuple int, screen: pygame.display.set\_mode((1200,800)))  +blitbg() |

|  |
| --- |
| Trigger |
| -screen: pygame.display.set\_mode((1200,800)) |
| +Trigger(screen: pygame.display.set\_mode((1200,800)))  +blitit() |

|  |
| --- |
| Exiting |
| -screen: pygame.display.set\_mode((1200,800)) |
| +Exiting(screen: pygame.display.set\_mode((1200,800)))  +blitit() |

1. **Flow Chart Diagram (Converting Program to Executable)**

Start

Set the TCL and TK libraries

Input the name of the file that want to be converted to .exe

Input the name for the installer

Input the used additional packages

Input the files that want to be included in the installer and the .exe directory

Run cx\_Freeze setup

END

1. **Custom Class Description**
2. Randomizing

Randomizing class controls the button ID assigning and sound assigning to every single button IDs. It also controls the ID reassigning in every new game. Every new game session, it serves as the indicator of which ID that is expected.

1. Button

Button class is used to describe the button that will be generated, including storing the ID that is assigned to the button. It also has responsibility to show the button to screen. It will load the button image from directory, and print it on screen.

1. Timing

Timing class controls the timer and the timer clock animation. It’s the one that responsible for limiting the game duration before it’s considered as game over.

1. Status

Status class is basically a switch whether the game is started or not. It controls the game trigger and menu trigger.

1. Music

Music class is responsible for playing the sound based from the ID that the sound is assigned to.

1. Bggame

Bggame class contains any description of background that will be shown during the game. It will load background image and print it on screen.

1. Bghome

Bghome class contains any description of background that will be shown in menu screen. It will load background image and print it on screen.

1. Trigger

Trigger class is used to generate start button. It contains any description about the button. It will load the button image and print it on screen.

1. Exiting

Exiting class is used to generate exit button. It contains any description about the button. It will load the button image and print it on screen.

1. **Functions Outside the Class**
2. rungame()

rungame() is the main function which triggers the whole program. It calls other functions and classes to run all of them. It also contains the button list for storing the buttons when the game is started.

1. checkev()

checkev() is the function to check the input from the hardware. The name of the function is actually the acronym of check event. It will check whether the input is clicking, hovering, or clicking the close button from outside of the game screen.

1. button\_click()

button\_click() is the function that is integrated with checkev(). It will check which button that is clicked, both when in menu screen and when the game is started. It also will check if the button that is clicked is the right one or the wrong one, and it will check whether the button list is empty or not.

1. **Source Code Dump**
2. **interface.py**

*#Import modules and custom classes***import** pygame  
**from** pygame.sprite **import** Group  
**import** sys  
**from** Button\_generate **import** \*  
**from** randomizer\_audio **import** \*  
**import** mechanism **as** me  
**from** stats **import** Status  
**from** Timer **import** \*  
**from** BG **import**\*  
*#Creating function for running the game***def** rungame():  
 *#Initiate pygame* pygame.init()  
 *#Set the screen width* width=1200  
 *#Set the screen height* height=800  
 *#Set screen resolution* screen=pygame.display.set\_mode((width,height))  
 *#Get the reactangle area from screen (width, height, coordinates inside the screen box)* screen\_box=screen.get\_rect()  
 *#Set the title to "Bomb Disarming"* pygame.display.set\_caption(**"Bomb Disarming"**)  
 *#Import the background classes* bakgrogame=Bggame(screen\_box)*#Assigned to bakgrogame* bakgrohome=Bghome(screen\_box)*#Assigned to bakgrohome  
 #Scan the time tick in second (1000ms)* pygame.time.set\_timer(pygame.USEREVENT, 1000)  
 *#Generate font called 'font' and set 'font' style to Consolas with the size of 30* font = pygame.font.SysFont(**'Consolas'**, 30)  
 *#Import timer class called 'Timing'* ticking=Timing(screen\_box,screen)  
 *#Import randomizer class called 'Randomizing'* numbering=Randomizing()  
 *#Create sprite list for storing buttons* buttons=Group()  
 *#Import 'Status' class* stat=Status()  
 *#Import trigger game button class called 'Trigger'* trig=Trigger(screen)  
 *#Import quit game button class called 'Exiting'* quiting=Exiting(screen)  
 *#Start the loop* **while True**:  
 *#fill the screen with white base background* screen.fill((255, 255, 255))  
 *#Check the hardware event* me.checkev(numbering,buttons,stat,trig,quiting,ticking)  
 *#Update button list* buttons.update()  
 *#If the game is started* **if** stat.gamestart:  
 *#Show ingame background called 'bakgrogame'* bakgrogame.blitbg(screen)  
 *#Show the stopwatch-like timer* ticking.blittimer()  
 *#Scan if button list empty* **if** len(buttons) == 0:  
 *#Move mouse cursor to (0, 0)* pygame.mouse.set\_pos(0, 0)  
 *#Reset timer* ticking.resettimes()  
 *#Generate 8 Buttons* **for** a **in** range(1, 9):  
 *#Generate button ID number* numberid = numbering.modify()  
 *#Generate button* switch = Button(screen, numberid, a)  
 *#Assign music to button with specific ID* numbering.soundassigning(switch.ident)  
 *#Add button to the list* buttons.add(switch)  
 *#Play the sound started from the button with the ID of 1* Music(numbering.assignsound[numbering.comparing[0]])  
 *#If the game isn't started or is already finished* **if** stat.showup:  
 *#Show menu background called 'backgrohome'* bakgrohome.blitbg(screen)  
 *#Show trigger game button* trig.blitit()  
 *#Show quit game button* quiting.blitit()  
 *#Show buttons from button list* **for** btn **in** buttons:  
 btn.create()  
 *#Show numberical timer, Game Over, and Congrats* screen.blit(font.render(ticking.text, **True**, (255, 0, 0), (255, 255, 255)),(screen\_box.centerx, screen\_box.height - 100))  
 *#Update screen using flip* pygame.display.flip()  
*#Run the function*rungame()

1. **Button\_generate.py**

*#Import modules***import** pygame  
**from** pygame.sprite **import** Sprite  
*#Button class***class** Button(Sprite):  
 *#Describe the button* **def** \_\_init\_\_(self,screen,ident,looping):  
 *#Using inheritance to inherit Sprite* super(Button,self).\_\_init\_\_()  
 *#Load the screen* self.scrn=screen  
 *#Get the screen rectangle* self.screen=self.scrn.get\_rect()  
 *#Load the button image* self.image=pygame.image.load(**'Cable.bmp'**)  
 *#Get image rectangle* self.rect=self.image.get\_rect()  
 *#Create the the first position coordinate minus 50 by dividing the height of the screen into eight* self.coordfirst=(self.screen.height / 8)  
 *#Set thge multiplier number for coordinate position* self.multiplier=50  
 *#Create the multiplier for the position* self.multiplying=(self.multiplier\*looping)  
 *#Generate the Y-coordinate* self.Y\_coordinate=self.coordfirst+self.multiplying  
 *#Set image coordinates* self.rect.centery = self.Y\_coordinate  
 self.rect.centerx = self.screen.centerx  
 *#Set the button ID* self.ident=ident  
 *#Method to create the button on screen* **def** create(self):  
 *#Print the button* self.scrn.blit(self.image,self.rect)  
*#Start button class***class** Trigger:  
 *#Describe the button* **def** \_\_init\_\_(self,screen):  
 *#Load the screen* self.scrn=screen  
 *#Get the creen rectangle* self.screen = self.scrn.get\_rect()  
 *#Load the start button image* self.image = pygame.image.load(**'Button1.png'**)  
 *#Get the image rectangle* self.rect = self.image.get\_rect()  
 *#Set the position of the button started from the center of screen* self.coordY=-50  
 *#Set the button coordinates* self.rect.centery = self.screen.centery+self.coordY  
 self.rect.centerx = self.screen.centerx  
 *#Method to print the button* **def** blitit(self):  
 *#Print the button to screen* self.scrn.blit(self.image,self.rect)  
*#Exit button class***class** Exiting:  
 *#Describe the button* **def** \_\_init\_\_(self,screen):  
 *#Load the screen* self.scrn=screen  
 *#Get the screen rectangle* self.screen = self.scrn.get\_rect()  
 *#Load the exit button image* self.image = pygame.image.load(**'Button2.png'**)  
 *#Get the image rectangle* self.rect = self.image.get\_rect()  
 *#Set the position of the button started from the center of screen* self.coordY=50  
 *#Set the button coordinates* self.rect.centery = self.screen.centery+self.coordY  
 self.rect.centerx = self.screen.centerx  
 *#Method to print the button* **def** blitit(self):  
 *#Print the button to screen* self.scrn.blit(self.image,self.rect)

1. **BG.py**

*#Importing Pygame***import** pygame  
*#InGame Background Class***class** Bggame:  
 *#Describe the background* **def** \_\_init\_\_(self,screen\_box):  
 *#Import the screen rectangle* self.screen\_box=screen\_box  
 *#Load the image* self.\_\_background1 = pygame.image.load(**'bombardo.bmp'**)  
 *#Get the background rectangle* self.\_\_background1rect = self.\_\_background1.get\_rect()  
 *#Set the center of the background* self.\_\_background1rect.center = self.screen\_box.center  
 *#Method to apply background to screen* **def** blitbg(self,screen):  
 *#Print background to screen* screen.blit(self.\_\_background1,self.\_\_background1rect)  
*#Menu background class***class** Bghome:  
 *#Describe the background* **def** \_\_init\_\_(self,screen\_box):  
 *#Import the screen rectangle* self.screen\_box=screen\_box  
 *#Load the image* self.\_\_background2 = pygame.image.load(**'homescr.bmp'**)  
 *#Get the background rectangle* self.\_\_background2rect = self.\_\_background2.get\_rect()  
 *#Set the center of the background* self.\_\_background2rect.center = self.screen\_box.center  
 *#Method to apply background to screen* **def** blitbg(self,screen):  
 *#Print background to screen* screen.blit(self.\_\_background2,self.\_\_background2rect)

1. **mechanism.py**

*#Import modules***import** pygame  
**import** sys  
**import** time  
*#Import custom class, music***from** randomizer\_audio **import** Music  
*#Checking event function***def** checkev(numbering,buttons,stat,menustart,menuexit,ticking):  
 *#Check occured event* **for** event **in** pygame.event.get():  
 *#Get cursor positions* cursorx, cursory = pygame.mouse.get\_pos()  
 *#If the event is quit, then It will close the program* **if** event.type==pygame.QUIT:  
 *#Exit program* sys.exit()  
 *#If mouseclick, then It will trigger checking the click positions* **if** event.type==pygame.MOUSEBUTTONDOWN:  
 *#Check click position* button\_click(numbering,buttons,cursorx,cursory,

stat,menustart,menuexit,ticking)  
 *#If cursor above the button, then, It will play sound based from the alphabet inside the ID dictionary* **if** event.type==pygame.MOUSEMOTION:  
 *#Check on every buttons* **for** clicking **in** buttons:  
 *#Check collide point* **if** clicking.rect.collidepoint(cursorx, cursory):  
 *#Get the alphabet from ID dictionary* soundclue=numbering.assignsound[clicking.ident]  
 *#Play the sound* Music(soundclue)  
 *#This will be triggered by USEREVENT event* **if** event.type == pygame.USEREVENT:  
 *#Activate when game is running* **if** stat.gamestart:  
 *#Check if there are still buttons inside the list* **if** len(buttons)>0:  
 *#Decrease time using function* ticking.decrement(stat,buttons,numbering)  
*#Button click Function***def** button\_click(numbering,checking,cursorx,cursory,stat,menustart,menuexit,ticking):  
 *#Check when it's started* **if** stat.gamestart:  
 *#Check whether the checking is empty or not (button list)* **if** len(checking) != 0:  
 *#Check every button(clicking) in list(checking)* **for** clicking **in** checking:  
 *#Check collision between cursor click and clicking* **if** clicking.rect.collidepoint(cursorx, cursory):  
 *#Check whether the ID of clicking is match with the comparing list or not* **if** clicking.ident == numbering.comparing[0]:  
 *#Removing the comparing value that is match with clicking ID* numbering.removing()  
 *#Remove clicking from checking* checking.remove(clicking)  
 *#Update checking* checking.update()  
 *#Check the comparing list whether is empty or not* **if** len(numbering.comparing) > 0:  
 *#Play the next sound based from the ID* Music(numbering.assignsound[numbering.comparing[0]])  
 **else**:  
 *#Statement* ticking.text=**"CONGRATULATION!"** *#Stop the game* stat.gamestart = **False** *#Show the menu* stat.showup = **True** *#Emptying checking* checking.empty()  
 *#Reset the values of the lists* numbering.ordered()  
 numbering.numbers()  
 numbering.audioassign()  
 **else**:  
 *#Statement* ticking.text=**"GAME OVER!"** *#Stop the game* stat.gamestart=**False** *#Show the menu* stat.showup=**True** *#Empty the list* checking.empty()  
 *#Reset the values of the lists* numbering.ordered()  
 numbering.numbers()  
 numbering.audioassign()  
 **else**:  
 *#Check whether the menu is triggered or not* **if** stat.showup:  
 *#Check if the mouseclick is on the start button* **if** menustart.rect.collidepoint(cursorx, cursory):  
 *#Start the game* stat.gamestart=**True** *#Hide the menu* stat.showup=**False** *#Check if the mouseclick in on the exit button* **if** menuexit.rect.collidepoint(cursorx, cursory):  
 *#Close the program* sys.exit()

1. **randomizer\_audio.py**

*#Import modules***import** pygame  
**import** random  
*#Randomizer Class To assign ID and sound randomly***class** Randomizing:  
 *#Initialize values from other methods* **def** \_\_init\_\_(self):  
 self.ordered()  
 self.numbers()  
 self.audioassign()  
 *#Method of comparison list to compare IDs and to reset comparison list* **def** ordered(self):  
 self.comparing = [1, 2, 3, 4, 5, 6, 7, 8]  
 *#Method of ID list consisted of IDs that will be assigned to Buttons and reset ID list* **def** numbers(self):  
 self.list=[1,2,3,4,5,6,7,8]  
 *#Method of assigning alphabet to the IDs (the alphabet represents sounds that will be played (music class for furter configuration)* **def** audioassign(self):  
 *#Dictionary for storing the alphabets to the IDs* self.assignsound={1:**''**,2:**''**,3:**''**,4:**''**,5:**''**,6:**''**,7:**''**,8:**''**}  
 *#Alphabets list* self.\_\_alphabet=[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**,**'g'**,**'h'**]  
 *#Method to modify the ID list and assigning the IDs* **def** modify(self):  
 *#Choose the IDs randomly* assigninglist=random.choice(self.list)  
 *#Remove the ID trace inside the list to prevent one ID to be assigned to more than one button* self.list.remove(assigninglist)  
 *#Outputting the ID from the method to outside the class* **return** assigninglist  
 *#Method for modify alphabet list and assigning the alphabets* **def** soundassigning(self,idassign):  
 *#Choose the alphabet randomly* self.assigning=random.choice(self.\_\_alphabet)  
 *#Assign the alphabet to the specific ID dictionary* self.assignsound[idassign]=self.assigning  
 *#Remove the alphabet to prevent one aphabet to be assigned to more than one button* self.\_\_alphabet.remove(self.assigning)  
 *#Method to delete the first number of comparison list to prevent scanning for the ID that no longer exist (if button with ID 1 has already been removed, it will scan for button with ID 2)* **def** removing(self):  
 *#Delete command* **del** self.comparing[0]  
*#Music class for identify the sound that must to be played***class** Music:  
 *#Initialize 'Music' class* **def** \_\_init\_\_(self,checker):  
 *#Check the alphabet inside the ID dictionary, and play the specific sounds  
 #Load music, then play once  
 #pygame.mixer.music.load("") to load the sound  
 #pygame.mixer.music.play(1) to play te sound without loop* **if** checker==**'a'**:  
 pygame.mixer.music.load(**"chewy1.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'b'**:  
 pygame.mixer.music.load(**"alarm1.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'c'**:  
 pygame.mixer.music.load(**"bloop\_x.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'d'**:  
 pygame.mixer.music.load(**"buzzer\_x.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'e'**:  
 pygame.mixer.music.load(**"cartoon001.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'f'**:  
 pygame.mixer.music.load(**"coin2.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'g'**:  
 pygame.mixer.music.load(**"cymbals.wav"**)  
 pygame.mixer.music.play(1)  
 **elif** checker==**'h'**:  
 pygame.mixer.music.load(**"disconnect\_x.wav"**)  
 pygame.mixer.music.play(1)

1. **stats.py**

*#This is 'Status' class to indicate whether the game is started or not***class** Status:  
 *#It only has 1 method, the initializer* **def** \_\_init\_\_(self):  
 self.gamestart = **False** *#This function is used for indicate whether the game is started or not* self.showup = **True** *#This function is used for trigger the menu screen or not*

1. **Timer.py**

*#Importing modules***import** time  
**import** pygame  
*#Timer class***class** Timing():  
 *#Initialize the timer* **def** \_\_init\_\_(self,screen\_box,screen):  
 *#Load the screen* self.screen=screen  
 *#Get the screen rectangle* self.screen\_box=screen\_box  
 *#Time limit setting* self.timelimit=30  
 *#Reset the timer or get the timer components* self.resettimes()  
 *#Initial Statement* self.text=**"Disarm The Bomb"** *#Method to reset the timer or get the timer components* **def** resettimes(self):  
 *#Set the counter time limit* self.\_\_counter=self.timelimit  
 *#Set initial number string for the counter (that will be shown on screen)* self.text =**'30'** *#Load timer arrow image* self.timearrow = pygame.image.load(**'timer arrow.png'**)  
 *# Copy the timer arrow image* self.timearrowcopy=self.timearrow  
 *#Load the timer frame image* self.timeframe = pygame.image.load(**'timer frame.png'**)  
 *# Get timer arrow image rectangle* self.timearrow\_rect = self.timearrow.get\_rect()  
 *#Set the time arrow coordinate* self.timearrow\_rect.centerx=self.screen\_box.width-100  
 self.timearrow\_rect.centery=self.screen\_box.centery-100  
 *#Get the copy of time arrow image rectangle* self.timearrowcopy\_rect=self.timearrow\_rect  
 *#Get the timer frame image rectangle* self.timeframe\_rect = self.timeframe.get\_rect()  
 *#Set the timer frame image coordinate* self.timeframe\_rect.centerx = self.screen\_box.width - 100  
 self.timeframe\_rect.centery = self.screen\_box.centery - 100  
 *#Method to decrease time* **def** decrement(self,stat,buttons,numbering):  
 *#Check whether the counter is already zero or not* **if** self.\_\_counter > 0:  
 *#Reduce the counter value* self.\_\_counter -= 1  
 *#Change the number string to counter current value in the form of string* self.text = str(self.\_\_counter)  
 *#Move the timer arrow clockwise* self.timearrowcopy=pygame.transform.rotate(self.timearrowcopy,(-360/self.timelimit))  
 self.timearrowcopy\_rect=self.timearrowcopy.get\_rect(center=self.timearrow\_rect.center)  
 **else**:  
 *#Statement* self.text = **"GAME OVER!"** *#Stop the game* stat.gamestart = **False** *#Show the menu* stat.showup = **True** *#Empty the button list* buttons.empty()  
 *#Reset the values of the lists* numbering.ordered()  
 numbering.numbers()  
 numbering.audioassign()  
 *#Method to print the timer to screen* **def** blittimer(self):  
 *#Print the timer frame to screen* self.screen.blit(self.timeframe,self.timeframe\_rect)  
 *#Print the timer arrow to screen* self.screen.blit(self.timearrowcopy,self.timearrowcopy\_rect)

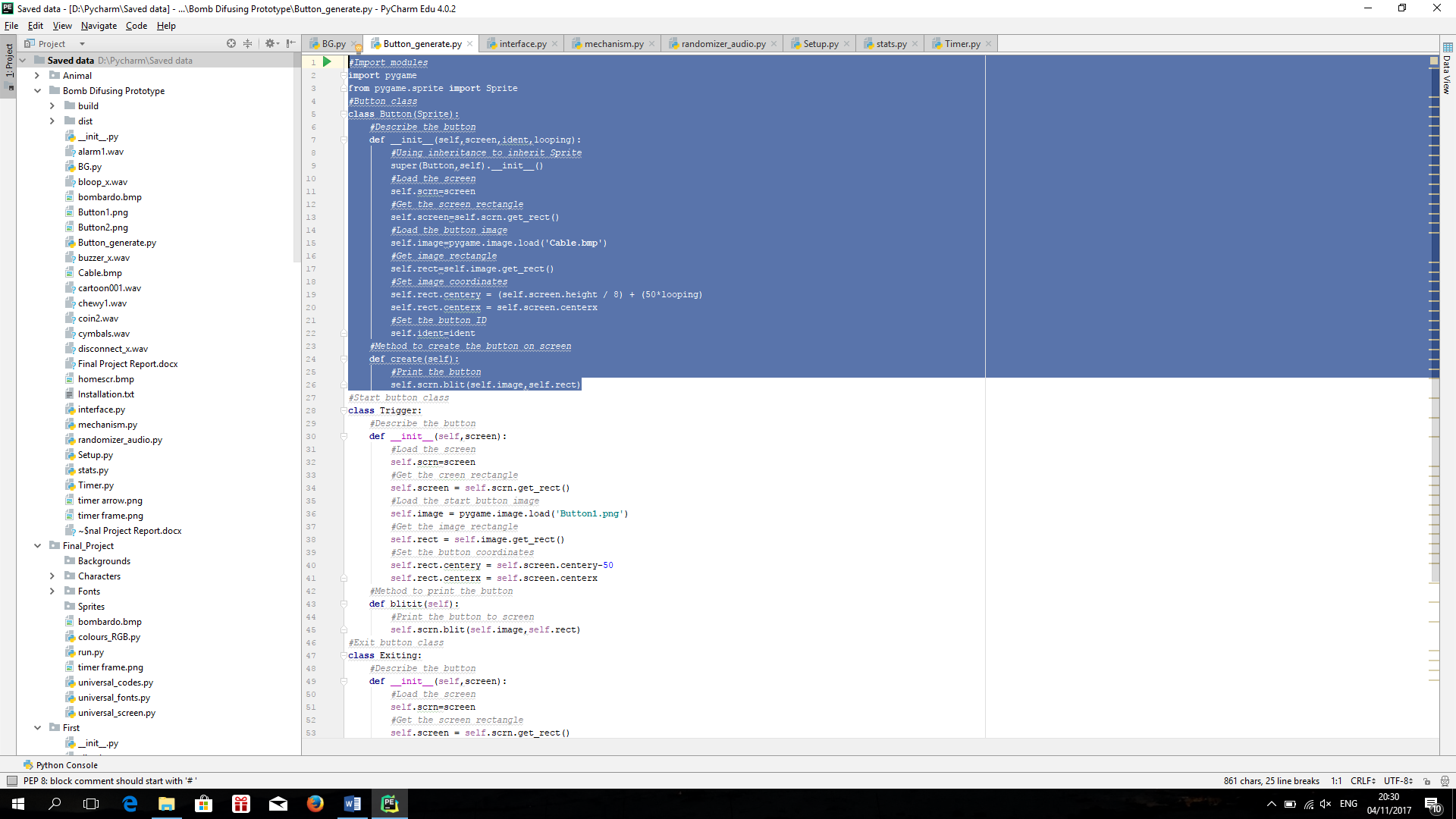
1. **Setup.py**

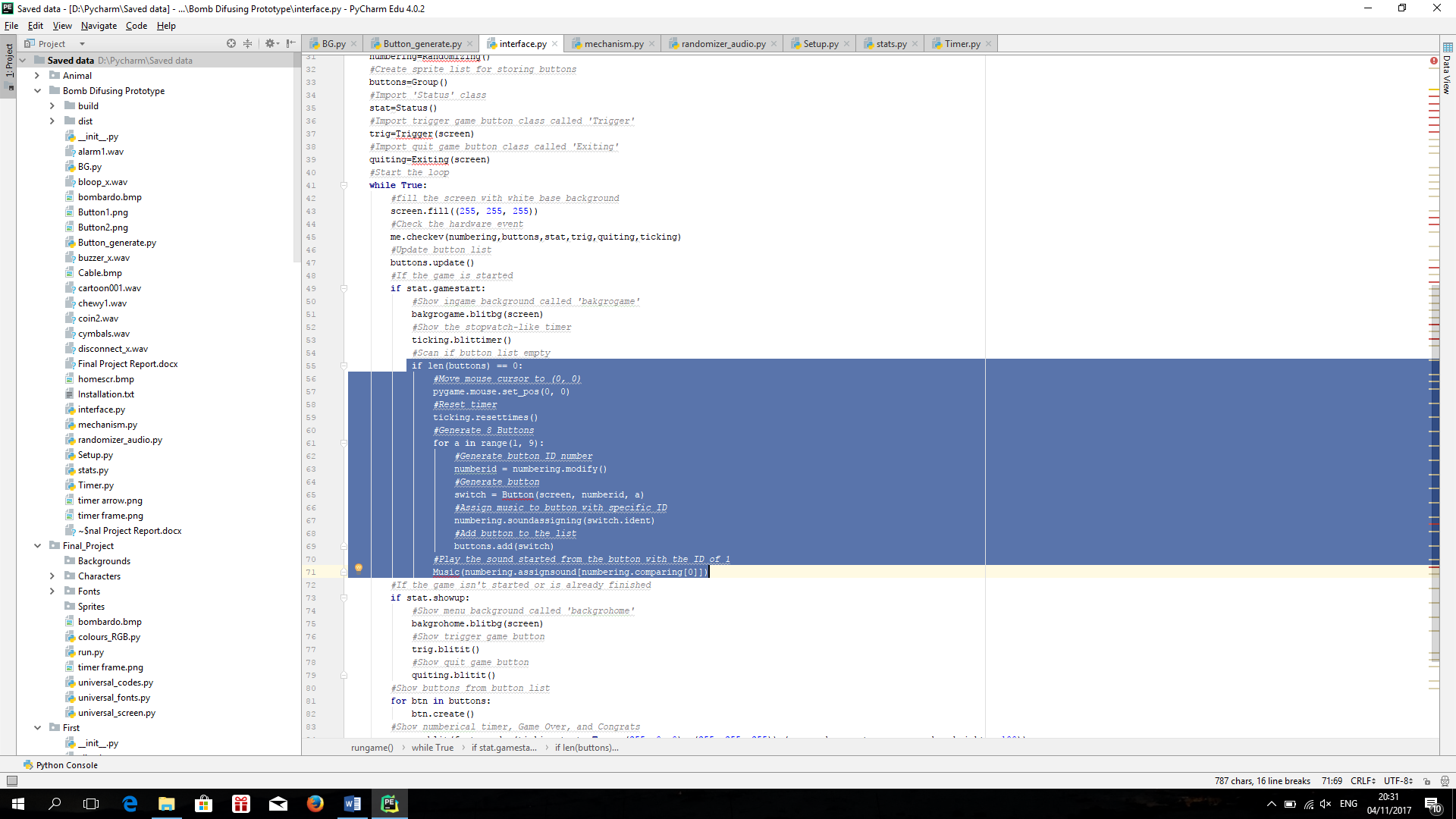
*#Using cx\_Freeze module***import** cx\_Freeze  
*#Importing os module***import** os  
*#Get the TCL and TK directories*os.environ[**'TCL\_LIBRARY'**] = **r'C:\Users\DarkHelix\AppData\Local\Programs\Python\Python36\tcl\tcl8.6'**os.environ[**'TK\_LIBRARY'**] = **r'C:\Users\DarkHelix\AppData\Local\Programs\Python\Python36\tcl\tk8.6'***#Variable to select which file that you want to have Its executable file*executing=[cx\_Freeze.Executable(**"interface.py"**)]  
*#Run cx\_Freeze and create needed files to run apps*cx\_Freeze.setup(  
 *#Name of installer* name=**"Bomb Disarming"**,  
 *#Describe the included files and packages* options={**"build\_exe"**:{**"packages"**:[**"pygame"**],  
 **"include\_files"**:[**"alarm1.wav"**,**"BG.py"**,**"bloop\_x.wav"**,**"bombardo.bmp"**,  
 **"Button1.png"**,**"Button2.png"**,**"Button\_generate.py"**,  
 **"buzzer\_x.wav"**,**"Cable.bmp"**,**"cartoon001.wav"**,**"chewy1.wav"**,  
 **"coin2.wav"**,**"cymbals.wav"**,**"disconnect\_x.wav"**,**"homescr.bmp"**,  
 **"mechanism.py"**,**"randomizer\_audio.py"**,**"stats.py"**,**"Timer.py"**,  
 **"timer arrow.png"**,**"timer frame.png"**]}},  
 *#Select the files that will have its executable file* executables = executing,  
 *#Describe the version of program* version=**'1.0.0'**)

1. **Progress**

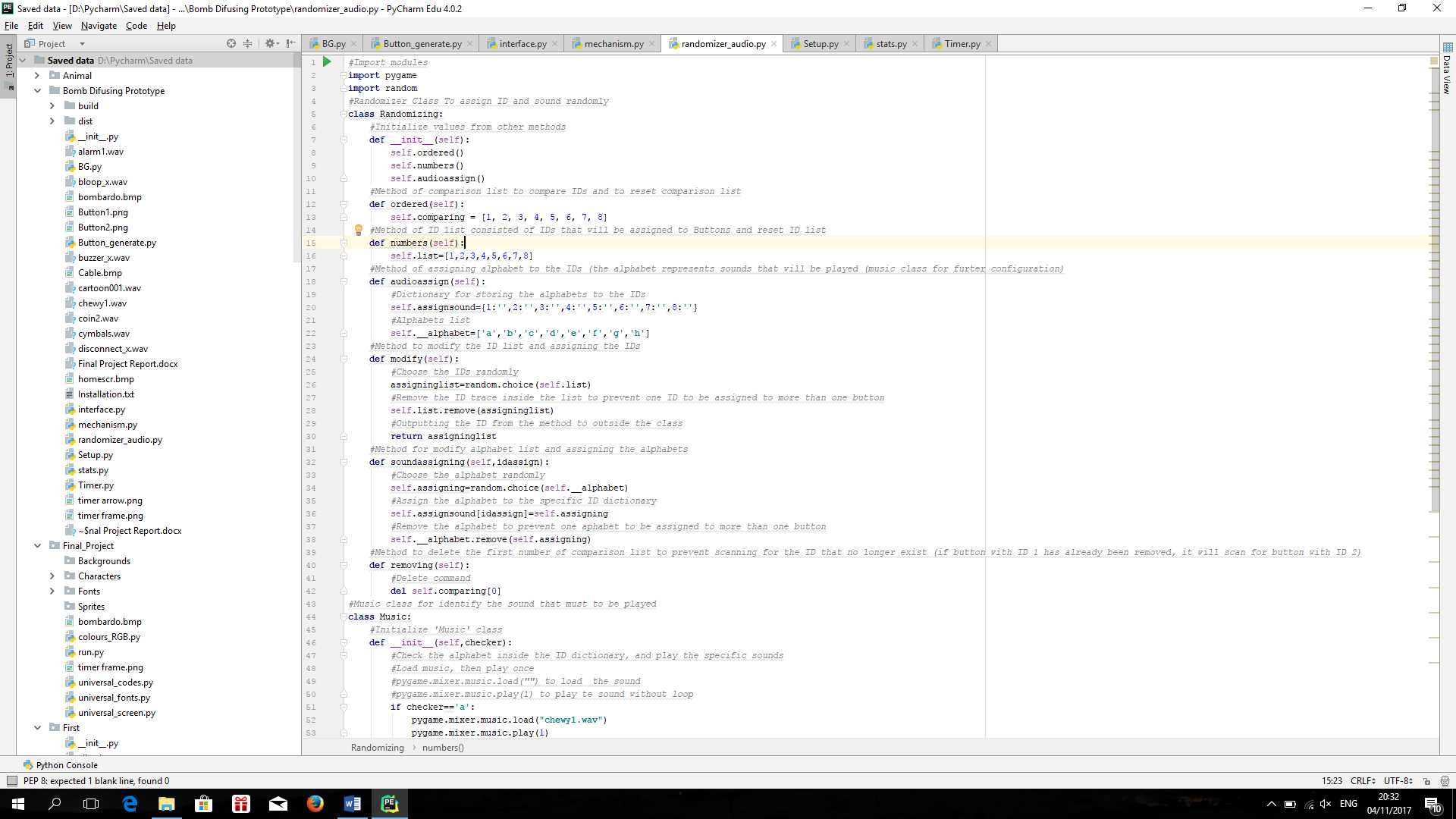
**Disclaimer :** The screenshot was taken after program was complete

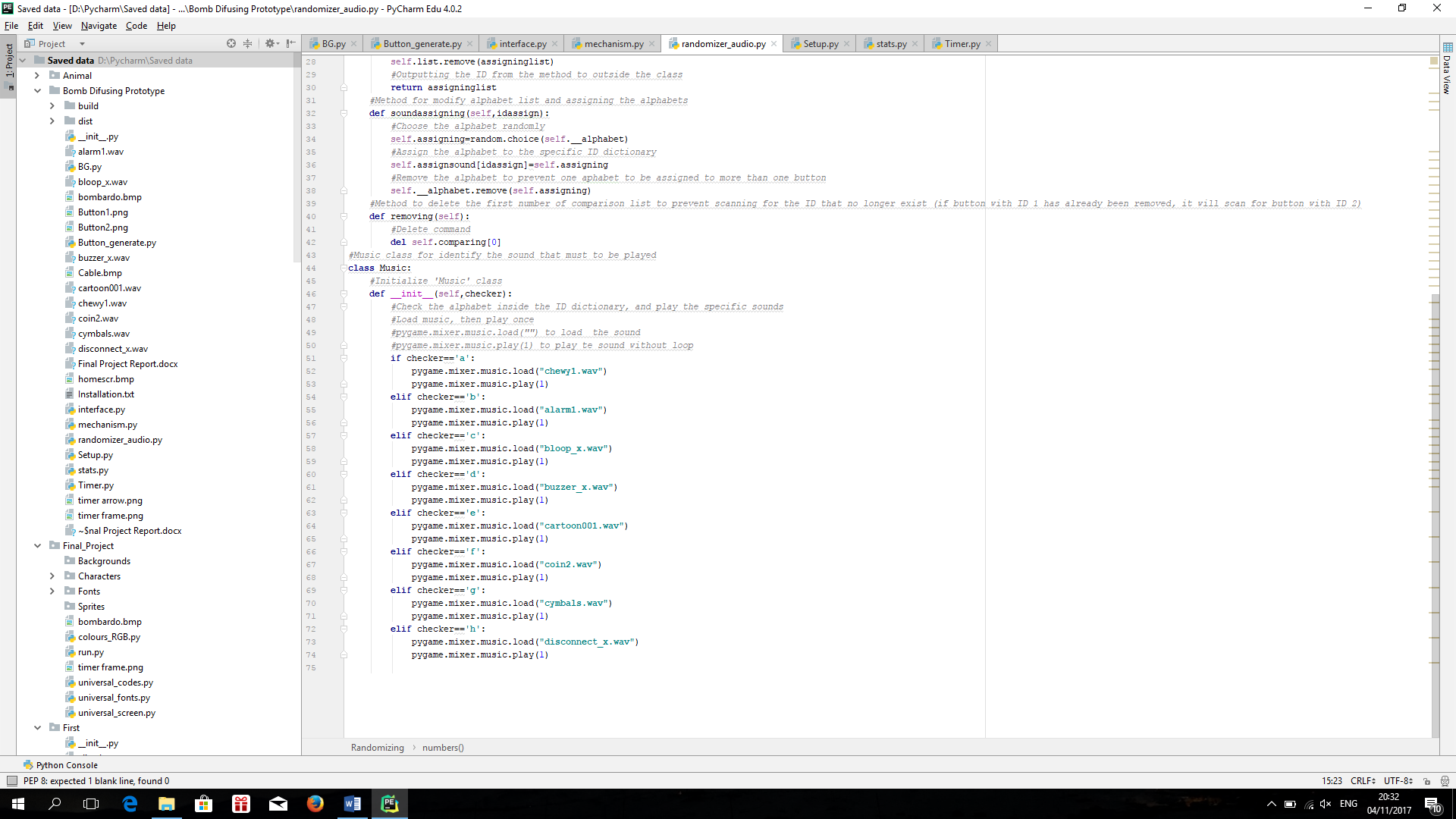
At First, I wanted to create a dodging game similar to Temple Run. I had prepared some of the game mechanics for that. But then, on Tuesday 24th October 2017, I had reconsidered myself to create another game instead of completing the initial plan. I decided to create something that is not as same as the others. When I saw my friend created a mystery horror game using PyGame buttons, I decided to create the game using the PyGame buttons. After a long process of thoughts, I decided to create bomb disarming game. In my first try, I’m trying to create the code to generate the buttons. The problem is when I want to generate buttons through the looping. I tried to generate buttons in different position by changing the value of the Y-coordinate from the outside of the class, but it didn’t actually work. So, I use the value from for looping to manipulate the button generation Y-coordinate by inserting the value inside the class.



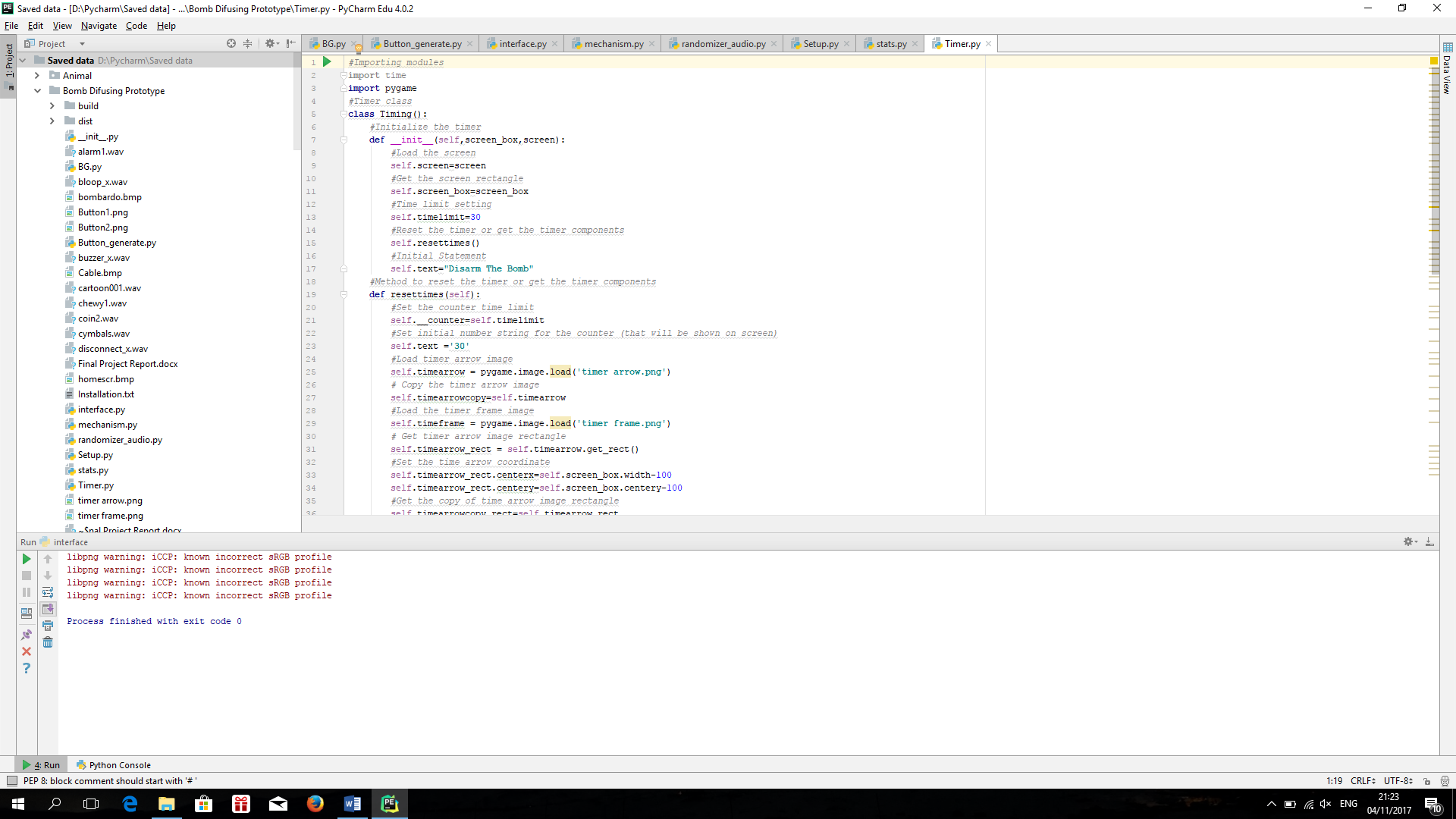


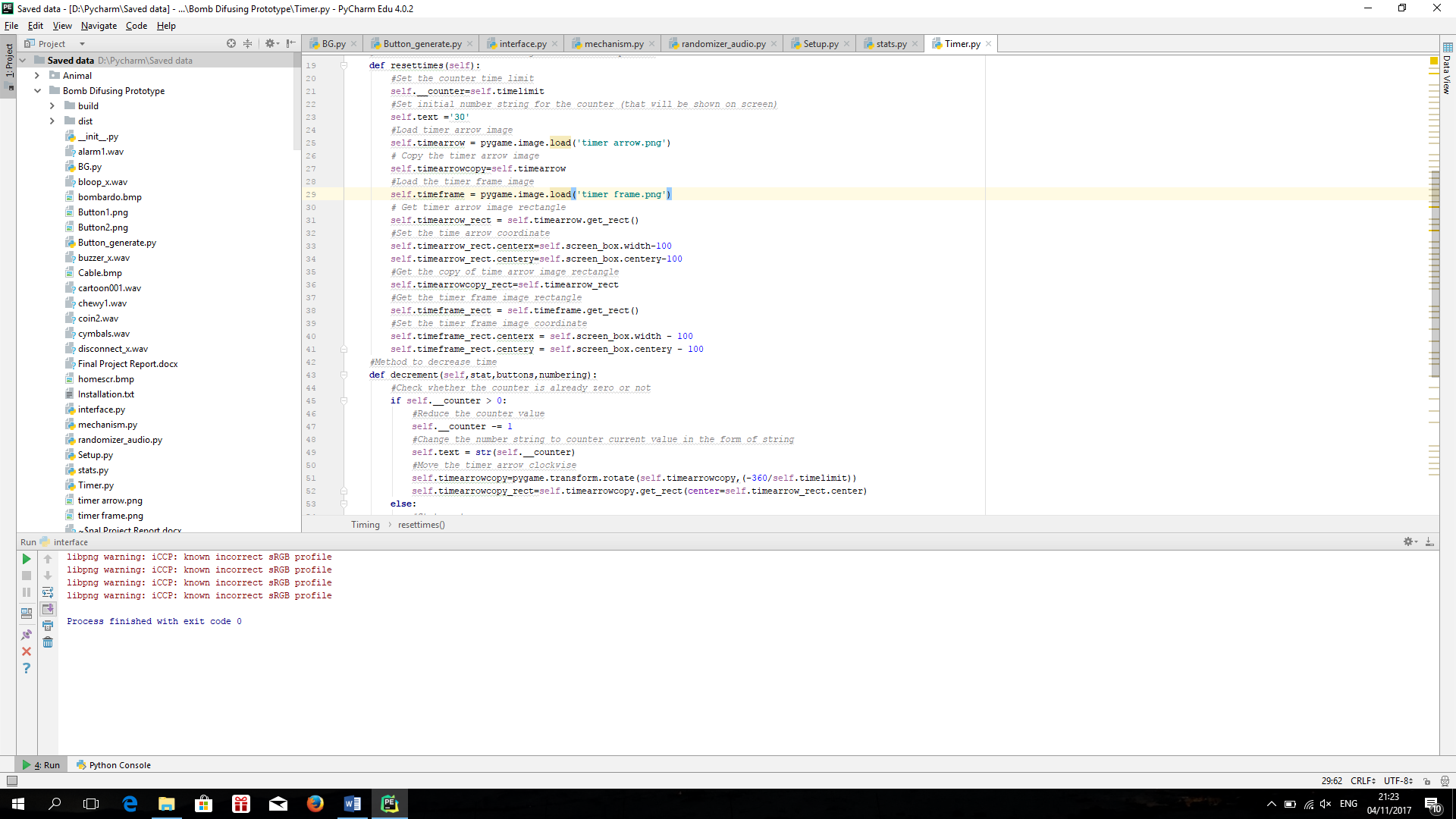
The next problem was to assign every button into designated IDs randomly. So, I decided to create randomizer class. Inside the class, I created the list of ID and another list to compare them. I also create a dictionary to assign the sound to every button based from Its ID. I also want to randomize the sound. So, I created another list contained alphabet that will be used to assign sound randomly. I also create a method to delete the ID of the button that is no longer existed, plus I put the lists inside a method so that when the player wants to restart the game, the game will automatically reset the lists. For the sound effect, I created another class to detect which sound that needed to be played. I used the music instead of sound because when I used sound, when I hover the cursor into other buttons, the other sound was played when the previous sound was yet to stopped, so it caused a huge interference.

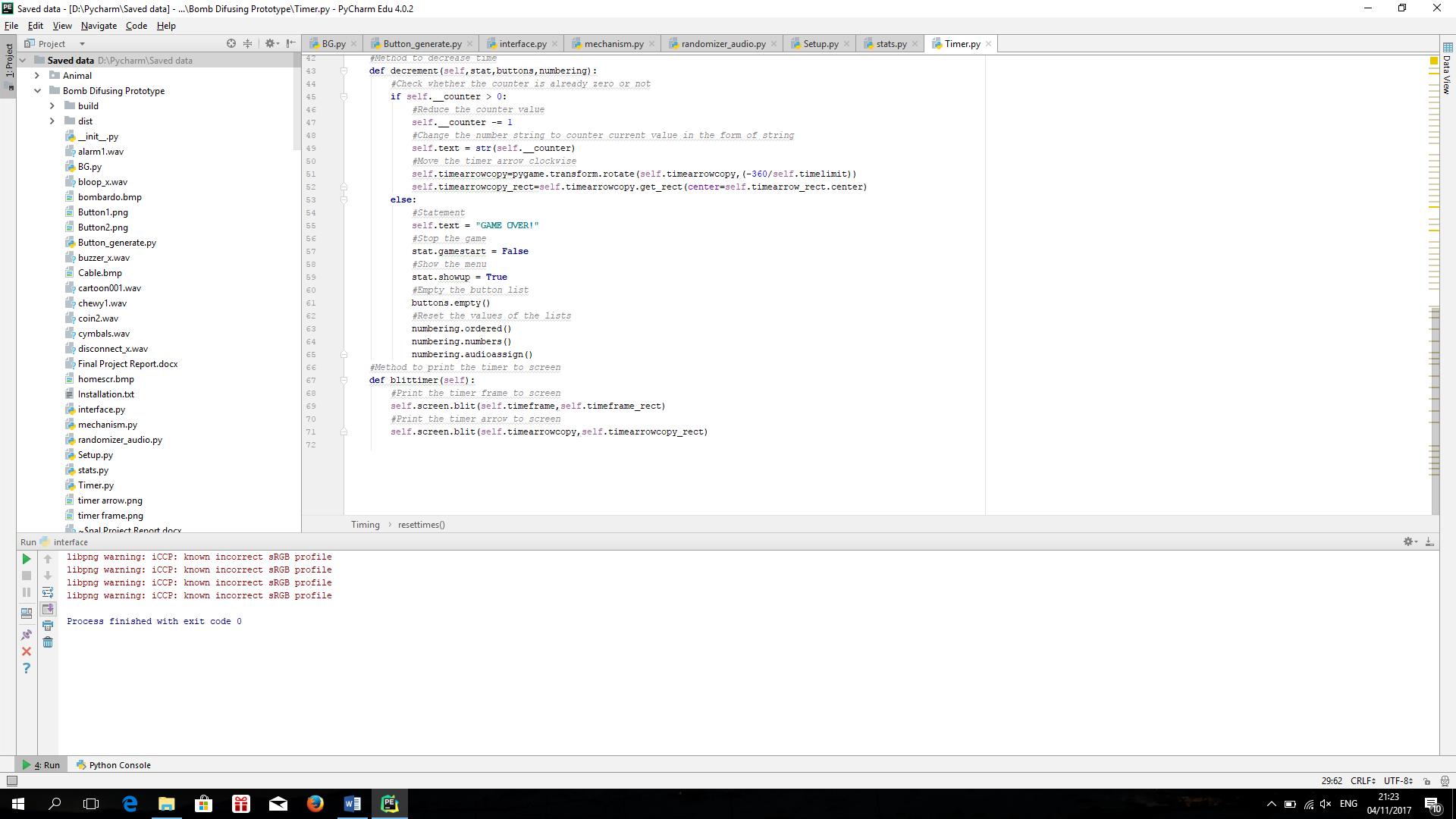




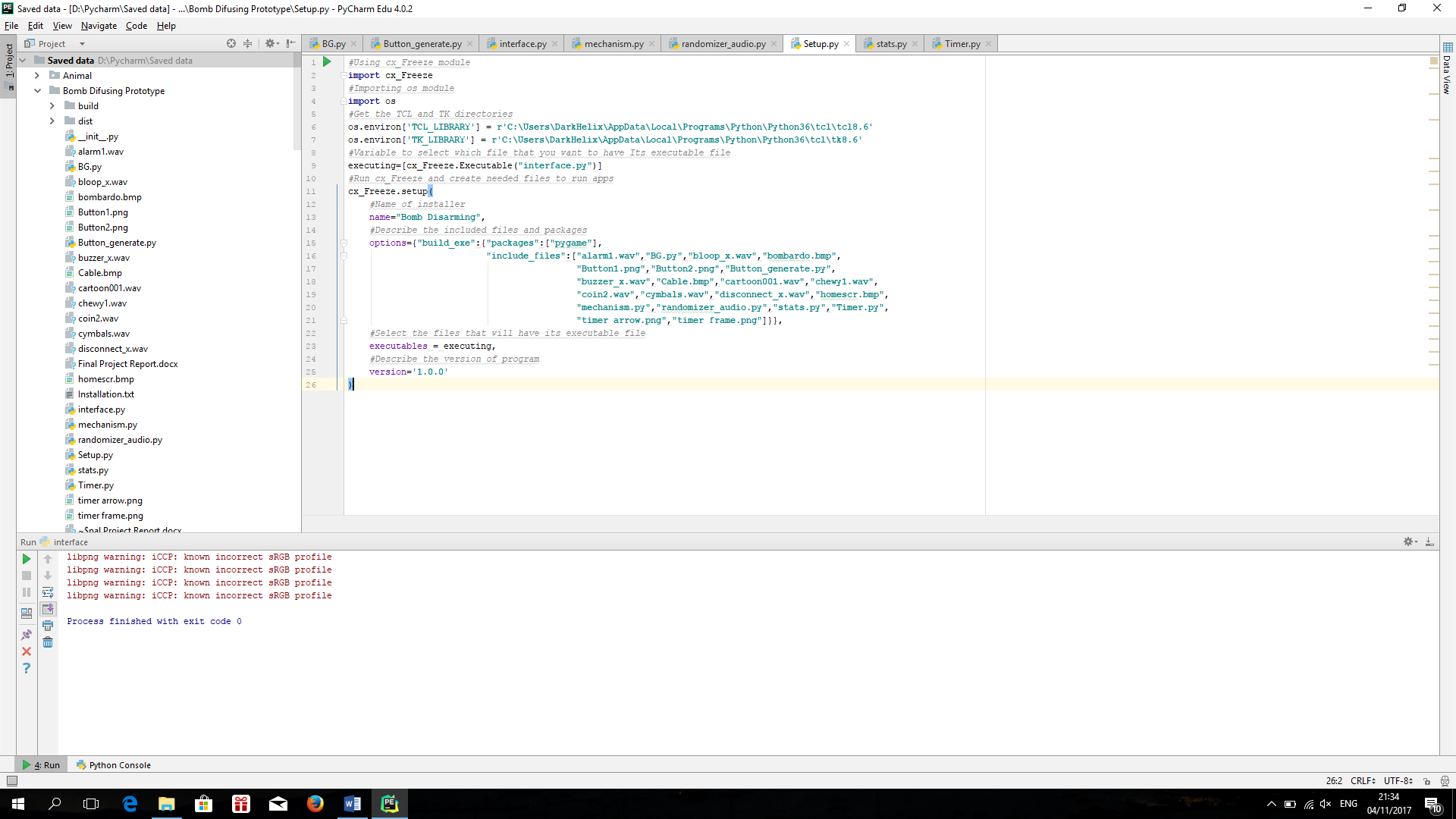
Next thing to do, the timer. I was doing that on Friday, 27th October 2017. I was having a trouble to create the proper timer. So, I searched so many sources, and finally came out with a solution:







The work Is pretty much done, but I do wonder if I can instead of just like that, I converted the thing into .exe. When I was thinking about It, on Monday 30th October 2017, my friend gave me a suggestion to watch a YouTube channel named sentdex. I watched how he converted the python files into executable. The thing is, it is similar with what I found on the internet. The name of the package is cx\_Freeze. I tried the program with several trials and errors. After failed to use them multiple times, I finally managed to use them properly.



The final program is like this:

