CS 11: Basic Programming

Machine Problem #2

Pinoy Ako : Gaano?
Larong susubok ng iyong pagka-Pilipino

Submitted by:

Rhoda Mae Cuevas

Cedric John Ligutan

Ryscheizca Bensh Villarino

Section:

AC-HXY-M3

Submitted on:

December 7, 2018

Contents:

Introduction 2

Game Description 2

How the program works

- Engine 2
- QQQ 8
- Timer 10

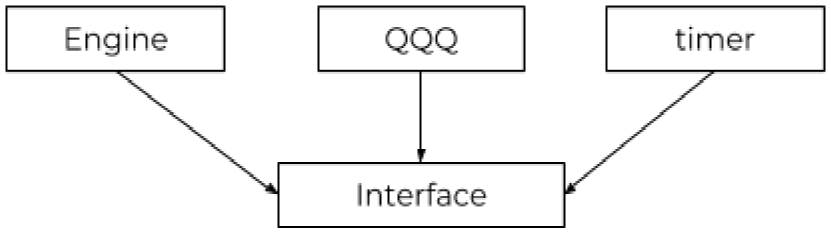
Interface 12

How to play the game 20

Sources 21

1. INTRODUCTION

The game program utilizes four files for the game to be executed. These are the Engine.py, QQQ.py, timer.py, and Interface.py. The Engine.py is the file for the game engine, the QQQ.py is the file for the question formatting, the timer.py is the file for the timers used in the game, and the Interface.py is the file for the user interface and it connects all the previously mentioned files. Each file is a user-defined module which are fed to the Interface.py through the import function in this order:



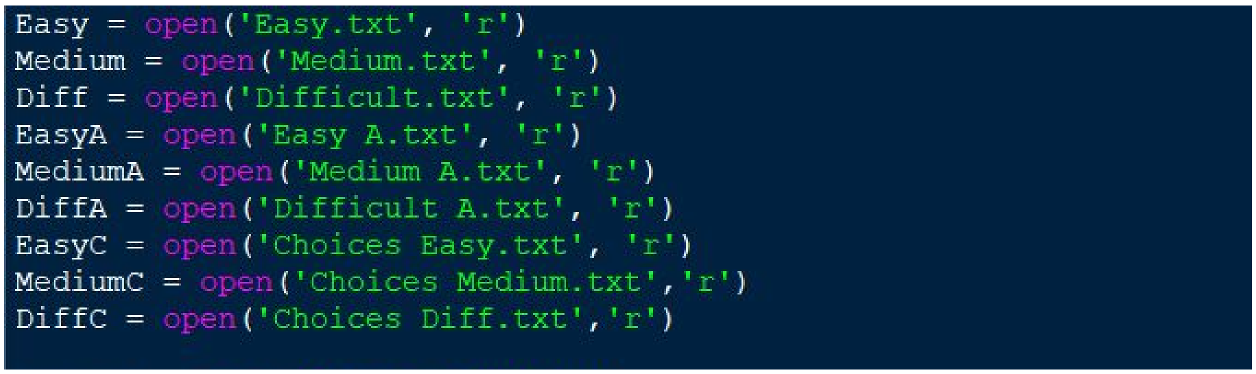
2. GAME DESCRIPTION

There are three levels in the game in which the player may choose from: *Madali* (Easy), *Katamtaman* (Medium), and *Mahirap* (Difficult). Each level utilizes the same game mechanics in which players must answer ten randomized questions within a time limit. The *Madali* level is given a longer time limit of 2 minutes while both the *Katamtaman* and *Mahirap* levels are given a time limit of 1 minute.

3. HOW THE PROGRAM WORKS

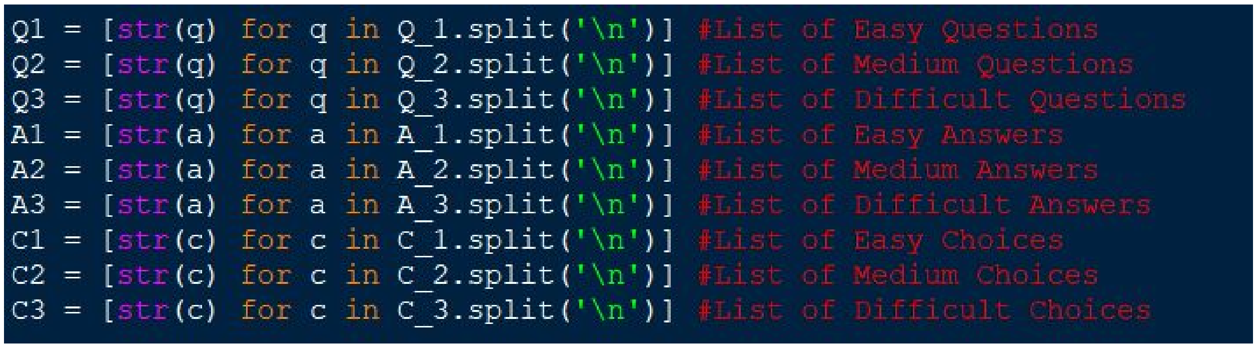
3.1. ENGINE

The engine serves as the base for the mechanics of the game. It starts with opening files that need to be accessed and files that store information. The text in the files to be read are used as the content of the game itself. These files contain the questions, answers, and choices for each level.



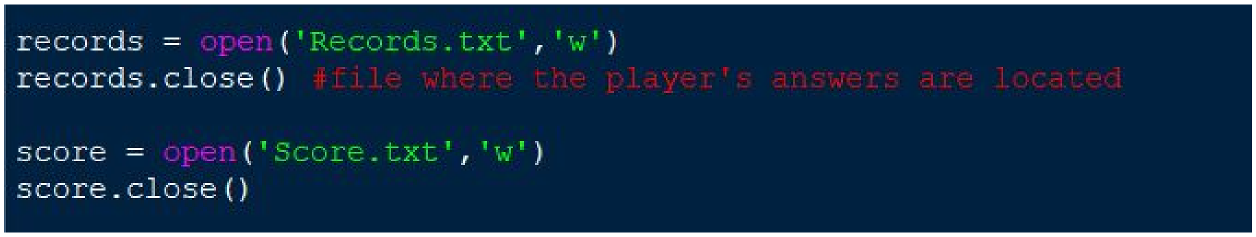
```
Easy = open('Easy.txt', 'r')
Medium = open('Medium.txt', 'r')
Diff = open('Difficult.txt', 'r')
EasyA = open('Easy A.txt', 'r')
MediumA = open('Medium A.txt', 'r')
DiffA = open('Difficult A.txt', 'r')
EasyC = open('Choices Easy.txt', 'r')
MediumC = open('Choices Medium.txt','r')
DiffC = open('Choices Diff.txt','r')
```

The text in the files are appended into different global lists (depending on the level and if the file contains question, answers, or choices).



```
Q1 = [str(q) for q in Q_1.split('\n')] #List of Easy Questions
Q2 = [str(q) for q in Q_2.split('\n')] #List of Medium Questions
Q3 = [str(q) for q in Q_3.split('\n')] #List of Difficult Questions
A1 = [str(a) for a in A_1.split('\n')] #List of Easy Answers
A2 = [str(a) for a in A_2.split('\n')] #List of Medium Answers
A3 = [str(a) for a in A_3.split('\n')] #List of Difficult Answers
C1 = [str(c) for c in C_1.split('\n')] #List of Easy Choices
C2 = [str(c) for c in C_2.split('\n')] #List of Medium Choices
C3 = [str(c) for c in C_3.split('\n')] #List of Difficult Choices
```

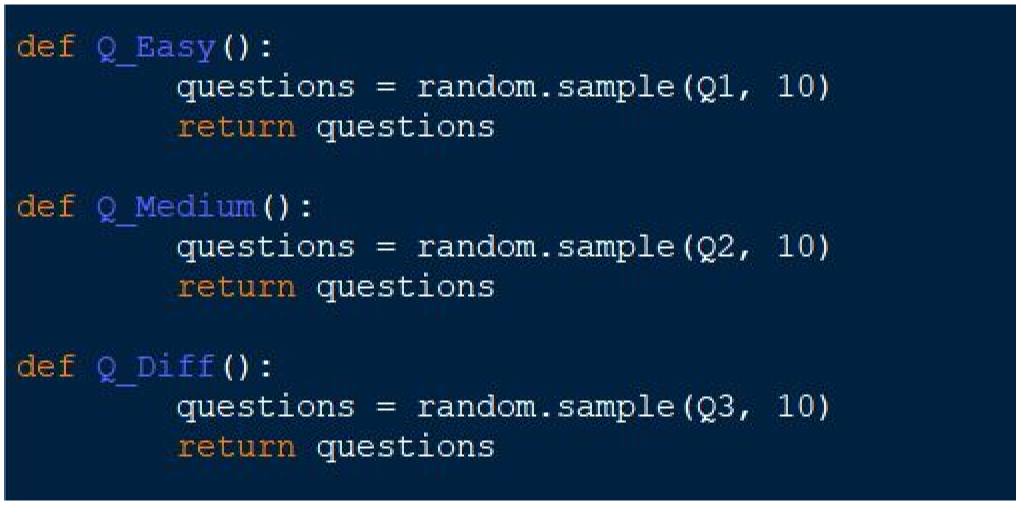
Two files store information as the game is being played. These are the *Records.txt* and the *Score.txt*. The records is where the player's answers are stored which will be needed by other functions in the engine. The score is where the score of the player is stored.



```
records = open('Records.txt','w')
records.close() #file where the player's answers are located

score = open('Score.txt','w')
score.close()
```

<u>Q Functions</u>

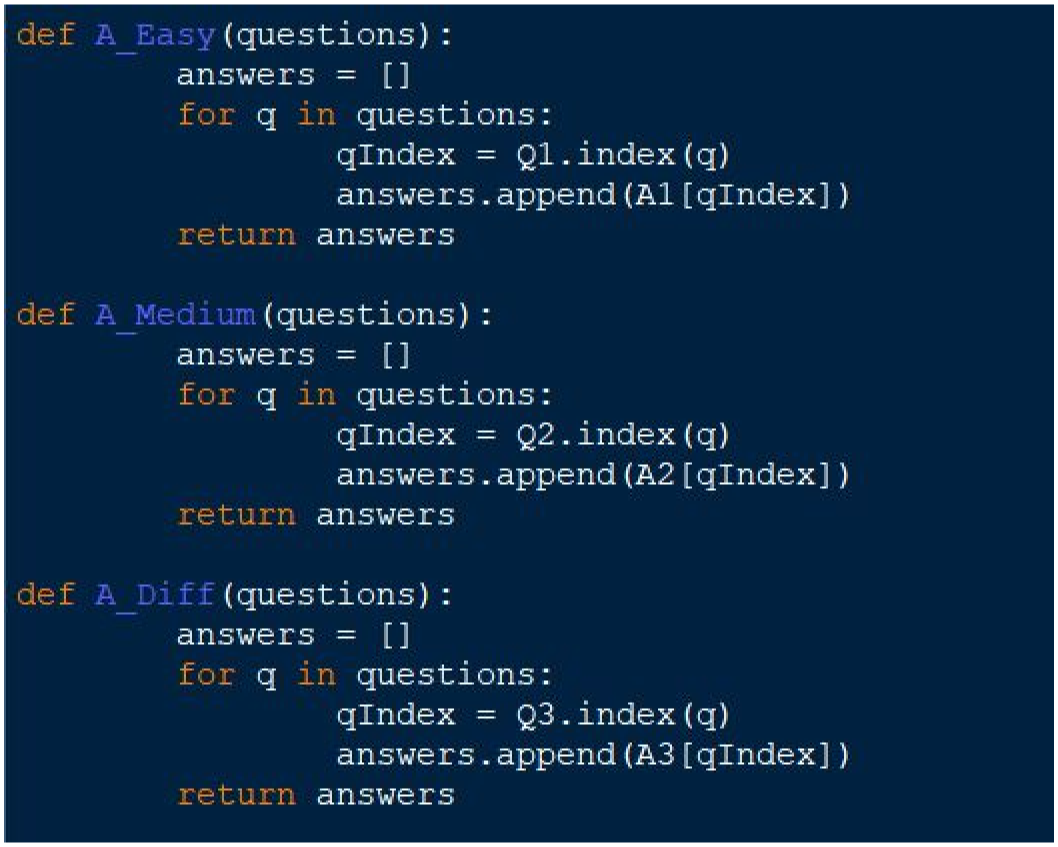
This function returns a list of random question from a list of questions. It uses the previously created global list of questions and the built-in random module. The random.sample module is then used to pick a random item in the list of questions. This prevents the constant repetition of questions when playing the game. A Q function is made for each level.



```
def Q_Easy():
        questions = random.sample(Q1, 10)
        return questions

def Q_Medium():
        questions = random.sample(Q2, 10)
        return questions

def Q_Diff():
        questions = random.sample(Q3, 10)
        return questions
```

A Functions

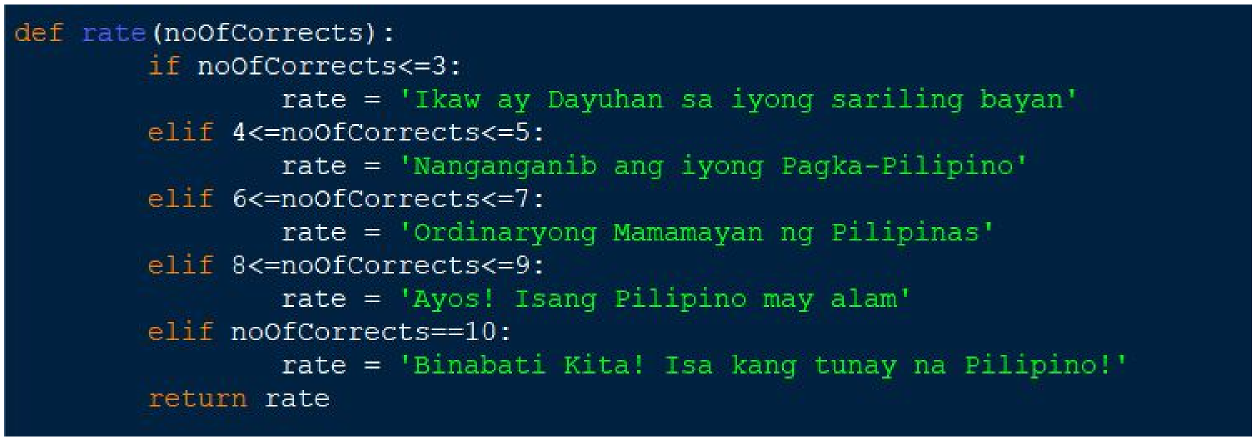
This function returns the corresponding list of answers to a list of question. It takes a list as an argument (named questions). The taken list is then accessed to find the corresponding index from its global list counterpart.. The index is then used to get the corresponding answer of a question (from a global list of answers) which is then appended to the list of answers. Like the Q function, there is an A function for each level of the game.



```
def A_Easy(questions):
        answers = []
        for q in questions:
                qIndex = Q1.index(q)
                answers.append(A1[qIndex])
        return answers

def A_Medium(questions):
        answers = []
        for q in questions:
                qIndex = Q2.index(q)
                answers.append(A2[qIndex])
        return answers

def A_Diff(questions):
        answers = []
        for q in questions:
                qIndex = Q3.index(q)
                answers.append(A3[qIndex])
        return answers
```

Rate Function

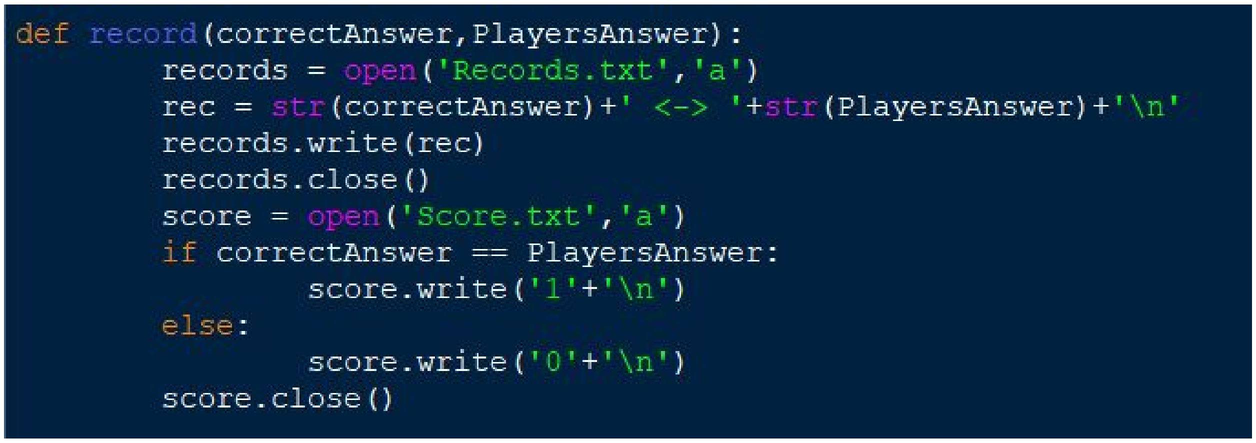
This function returns the rating of the player based on his/her score. It takes an integer value as an argument (named noOfCorrects). Depending on the value of noOfCorrects, a rate is assigned. This will be displayed after a round of the game.



```
def rate(noOfCorrects):
        if noOfCorrects<=3:
                rate = 'Ikaw ay Dayuhan sa iyong sariling bayan'
        elif 4<=noOfCorrects<=5:
                rate = 'Nanganganib ang iyong Pagka-Pilipino'
        elif 6<=noOfCorrects<=7:
                rate = 'Ordinaryong Mamamayan ng Pilipinas'
        elif 8<=noOfCorrects<=9:
                rate = 'Ayos! Isang Pilipino may alam'
        elif noOfCorrects==10:
                rate = 'Binabati Kita! Isa kang tunay na Pilipino!'
        return rate
```

Record Function

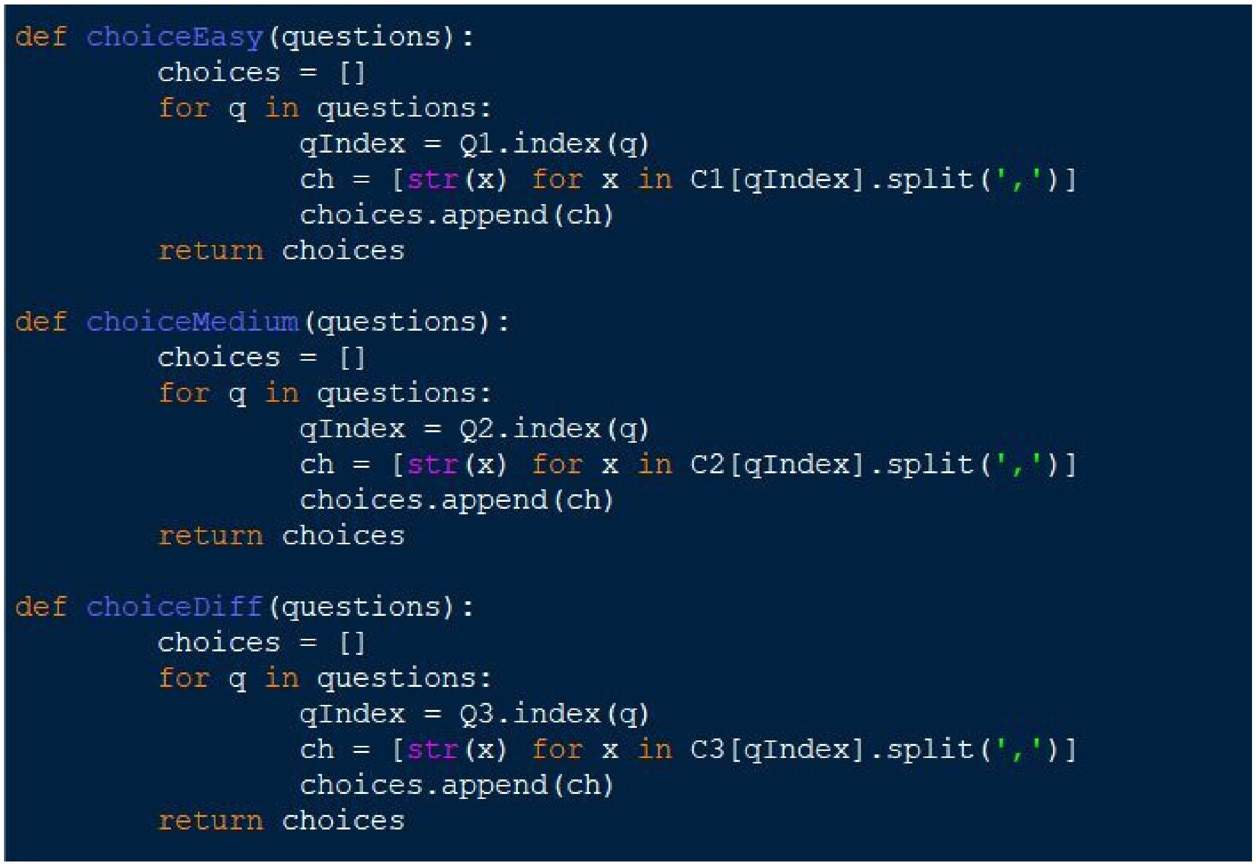
This function appends values into the Records.txt and Score.txt files. It takes two arguments, the correctAnswer and the PlayersAnswer. String addition is then used to store both values in the rec string. The value of rec is then written into the Records.txt file. The correctAnswer and the PlayersAnswer is then compared. If they contain the same value, ‘1’ is written into the Score.txt file, otherwise ‘0’ is written into the Score.txt file.



```
def record(correctAnswer,PlayersAnswer):
        records = open('Records.txt','a')
        rec = str(correctAnswer)+' <-> '+str(PlayersAnswer)+'\n'
        records.write(rec)
        records.close()
        score = open('Score.txt','a')
        if correctAnswer == PlayersAnswer:
                score.write('1'+'\n')
        else:
                score.write('0'+'\n')
        score.close()
```

Choice Function

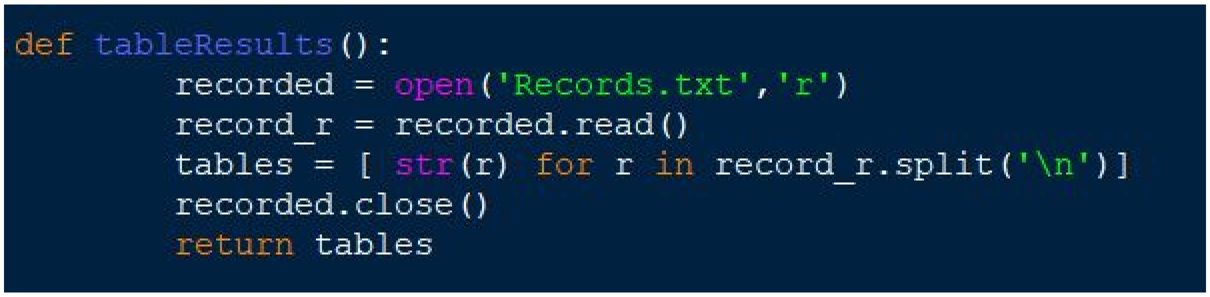
This function returns a corresponding list of choices to a list of questions. Like, the A function, it takes a list as an argument (named questions). The taken list is then accessed to find the corresponding index from its global list counterpart. The index is then used to access another global list of choices and stores the choices in the ch list. The ch list is then appended into the choices list which is then returned. This creates a nested list of choices. Again, a choice function is created for each level of the game.



```
def choiceEasy(questions):
        choices = []
        for q in questions:
                qIndex = Q1.index(q)
                ch = [str(x) for x in C1[qIndex].split(',')]
                choices.append(ch)
        return choices

def choiceMedium(questions):
        choices = []
        for q in questions:
                qIndex = Q2.index(q)
                ch = [str(x) for x in C2[qIndex].split(',')]
                choices.append(ch)
        return choices

def choiceDiff(questions):
        choices = []
        for q in questions:
                qIndex = Q3.index(q)
                ch = [str(x) for x in C3[qIndex].split(',')]
                choices.append(ch)
        return choices
```

Table Results Function

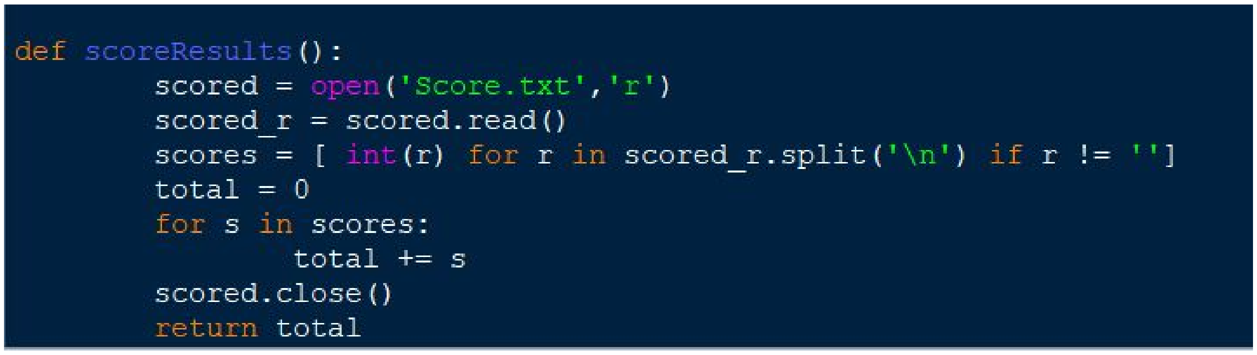
This function returns a list of the correct answers versus the player's answers. It opens and reads the Records.txt file and stores the content in a list (named tables). This is used to view the wrong and correct answers made by the player.



```
def tableResults():
        recorded = open('Records.txt','r')
        record_r = recorded.read()
        tables = [ str(r) for r in record_r.split('\n')]
        recorded.close()
        return tables
```

Score Results Function

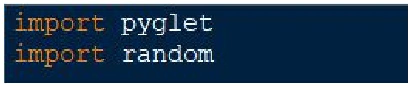
This function returns the total score obtained by the player. It opens and reads the Score.txt file and stores the content, parsed into an integer, in a list (named scores). Each item in the scores list is then accessed and added together and stored in the total variable.



```
def scoreResults():
        scored = open('Score.txt','r')
        scored_r = scored.read()
        scores = [ int(r) for r in scored_r.split('\n') if r != '']
        total = 0
        for s in scores:
                total += s
        scored.close()
        return total
```

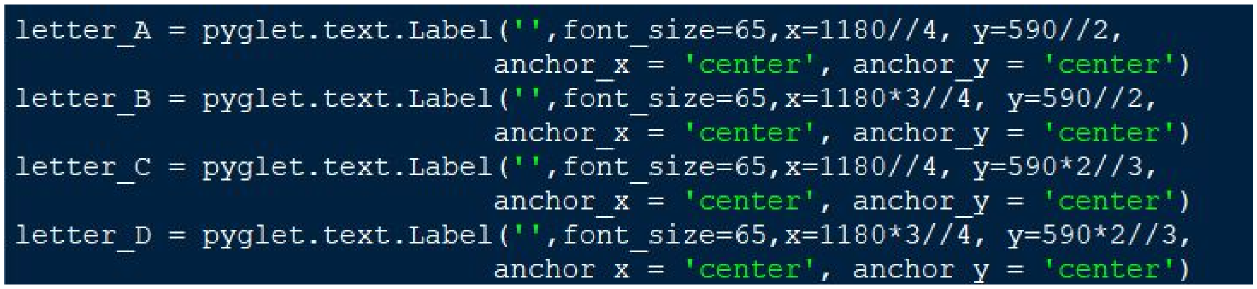
3.2. QQQ

This file is where the questions and choices for each level is formatted. It starts with importing the random module and pyglet, a multimedia library for python.



```python
import pyglet
import random
```

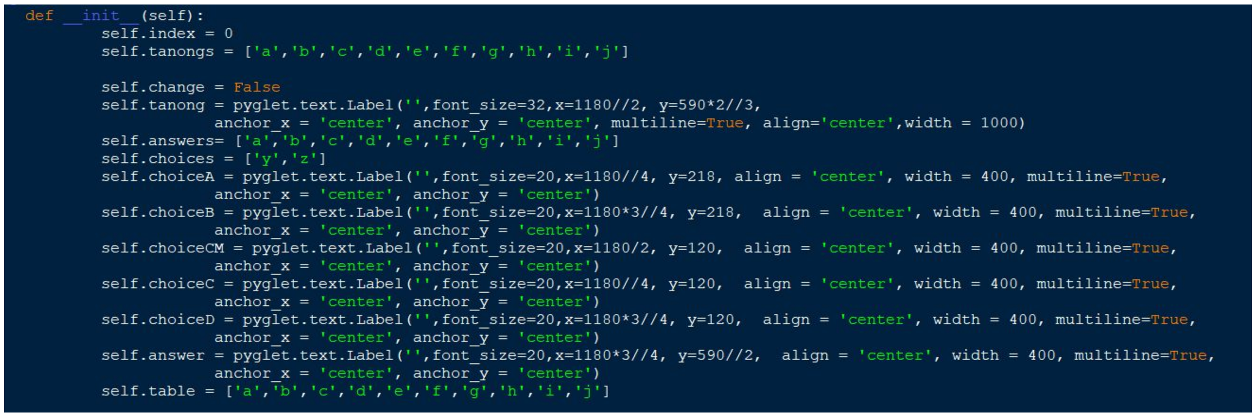
It also formats the individual choices and sets their location on the display window.



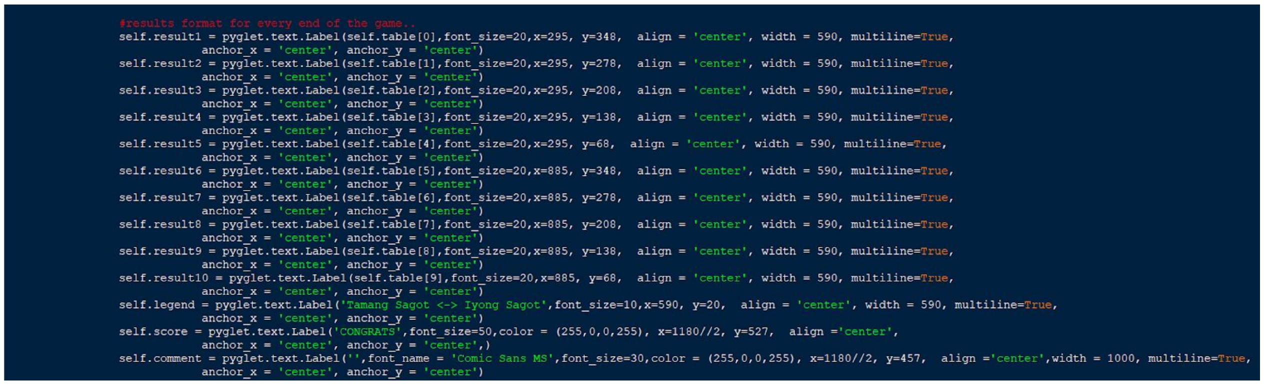
```python
letter_A = pyglet.text.Label('',font_size=65,x=1180//4, y=590//2,
                            anchor_x = 'center', anchor_y = 'center')
letter_B = pyglet.text.Label('',font_size=65,x=1180*3//4, y=590//2,
                            anchor_x = 'center', anchor_y = 'center')
letter_C = pyglet.text.Label('',font_size=65,x=1180//4, y=590*2//3,
                            anchor_x = 'center', anchor_y = 'center')
letter_D = pyglet.text.Label('',font_size=65,x=1180*3//4, y=590*2//3,
                            anchor_x = 'center', anchor_y = 'center')
```

<u>Initializing Function</u>

This function initializes the format of the questions, choices and the results. The format includes font size, its position in the window, and its alignment. This indicates that all questions and answers will have the same format all throughout the game, contrary to the case of the choices and the results. Every choice and every result will have the same format except for its position in the window depending on what choice or result it will be assigned to. At every end of the game, the result (in table form), the legend (for the table), the score, and the rating will appear. Only the score and the comment is attributed with a certain color and font style.



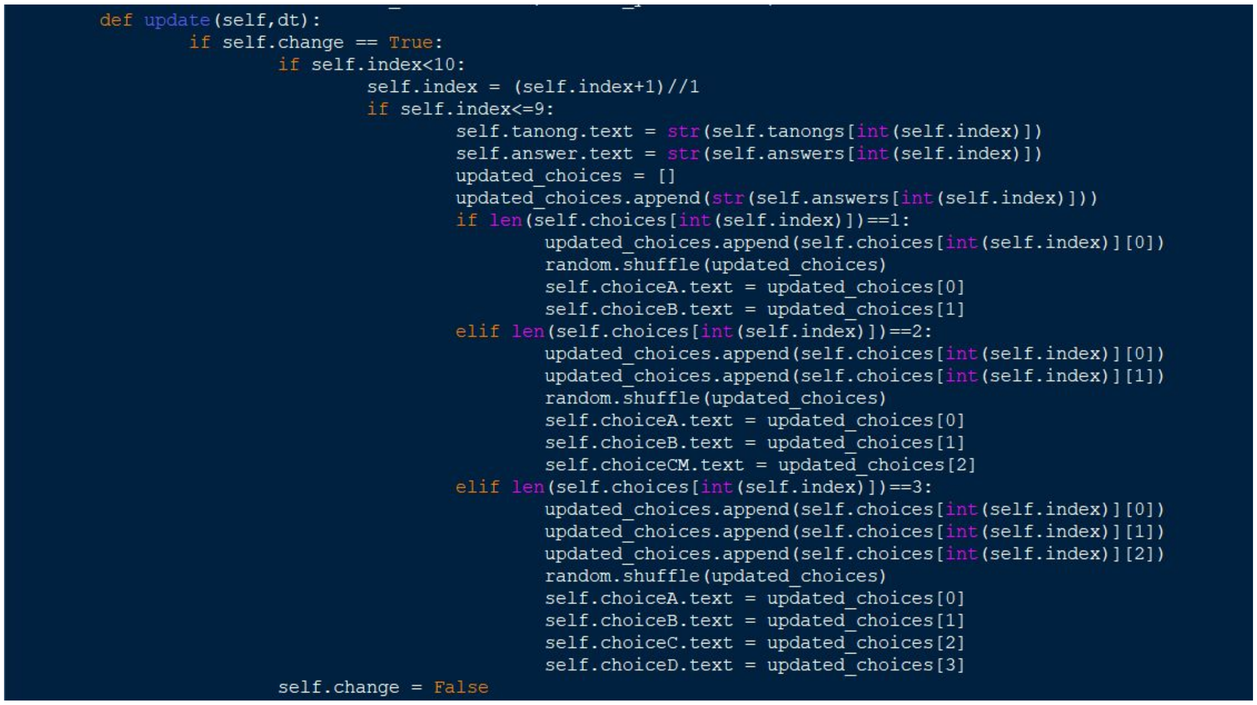
```python
def __init__(self):
        self.index = 0
        self.tanongs = ['a','b','c','d','e','f','g','h','i','j']

        self.change = False
        self.tanong = pyglet.text.Label('',font_size=32,x=1180//2, y=590*2//3,
                    anchor_x = 'center', anchor_y = 'center', multiline=True, align='center',width = 1000)
        self.answers= ['a','b','c','d','e','f','g','h','i','j']
        self.choices = ['y','z']
        self.choiceA = pyglet.text.Label('',font_size=20,x=1180//4, y=218, align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.choiceB = pyglet.text.Label('',font_size=20,x=1180*3//4, y=218,  align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.choiceCM = pyglet.text.Label('',font_size=20,x=1180/2, y=120,  align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.choiceC = pyglet.text.Label('',font_size=20,x=1180//4, y=120,  align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.choiceD = pyglet.text.Label('',font_size=20,x=1180*3//4, y=120,  align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.answer = pyglet.text.Label('',font_size=20,x=1180*3//4, y=590//2,  align = 'center', width = 400, multiline=True,
                    anchor_x = 'center', anchor_y = 'center')
        self.table = ['a','b','c','d','e','f','g','h','i','j']
```



```
#results format for every end of the game..
self.result1 = pyglet.text.Label(self.table[0],font_size=20,x=295, y=348,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result2 = pyglet.text.Label(self.table[1],font_size=20,x=295, y=278,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result3 = pyglet.text.Label(self.table[2],font_size=20,x=295, y=208,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result4 = pyglet.text.Label(self.table[3],font_size=20,x=295, y=138,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result5 = pyglet.text.Label(self.table[4],font_size=20,x=295, y=68,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result6 = pyglet.text.Label(self.table[5],font_size=20,x=885, y=348,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result7 = pyglet.text.Label(self.table[6],font_size=20,x=885, y=278,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result8 = pyglet.text.Label(self.table[7],font_size=20,x=885, y=208,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result9 = pyglet.text.Label(self.table[8],font_size=20,x=885, y=138,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.result10 = pyglet.text.Label(self.table[9],font_size=20,x=885, y=68,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.legend = pyglet.text.Label('Tamang Sagot <-> Iyong Sagot',font_size=10,x=590, y=20,  align = 'center', width = 590, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
self.score = pyglet.text.Label('CONGRATS',font_size=50,color = (255,0,0,255), x=1180//2, y=527,  align ='center',
            anchor_x = 'center', anchor_y = 'center',)
self.comment = pyglet.text.Label('',font_name = 'Comic Sans MS',font_size=30,color = (255,0,0,255), x=1180//2, y=457,  align ='center',width = 1000, multiline=True,
            anchor_x = 'center', anchor_y = 'center')
```

__Update Function__

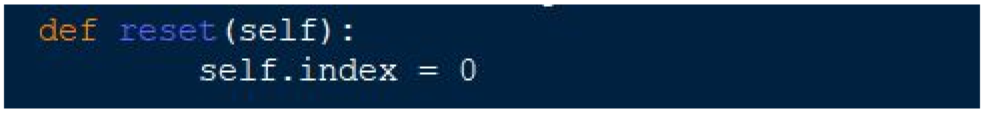
This function updates each choice text as the questions change. Items stored in self.tanong (for questions) and self.answer (corresponding answers) are accessed by self.index. The answer to the question and other choices are then stored into a list called updated_choices. The random.shuffle function then shuffles the items of the list to allow a random arrangement for display. Each item in updated_choices is then individually assigned to A, B, C, or D (the number of choices depend on the length of self.choices).



```
def update(self,dt):
        if self.change == True:
                if self.index<10:
                        self.index = (self.index+1)//1
                        if self.index<=9:
                                self.tanong.text = str(self.tanongs[int(self.index)])
                                self.answer.text = str(self.answers[int(self.index)])
                                updated_choices = []
                                updated_choices.append(str(self.answers[int(self.index)]))
                                if len(self.choices[int(self.index)])==1:
                                        updated_choices.append(self.choices[int(self.index)][0])
                                        random.shuffle(updated_choices)
                                        self.choiceA.text = updated_choices[0]
                                        self.choiceB.text = updated_choices[1]
                                elif len(self.choices[int(self.index)])==2:
                                        updated_choices.append(self.choices[int(self.index)][0])
                                        updated_choices.append(self.choices[int(self.index)][1])
                                        random.shuffle(updated_choices)
                                        self.choiceA.text = updated_choices[0]
                                        self.choiceB.text = updated_choices[1]
                                        self.choiceCM.text = updated_choices[2]
                                elif len(self.choices[int(self.index)])==3:
                                        updated_choices.append(self.choices[int(self.index)][0])
                                        updated_choices.append(self.choices[int(self.index)][1])
                                        updated_choices.append(self.choices[int(self.index)][2])
                                        random.shuffle(updated_choices)
                                        self.choiceA.text = updated_choices[0]
                                        self.choiceB.text = updated_choices[1]
                                        self.choiceC.text = updated_choices[2]
                                        self.choiceD.text = updated_choices[3]
                self.change = False
```

Reset Function

This function resets self.index to 0.



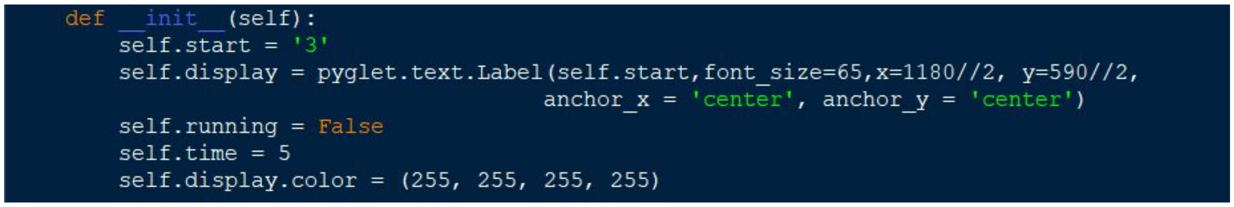
```python
def reset(self):
        self.index = 0
```

3.3. TIMER

The timer times the proceedings of the game. It has two classes, one for counting down before the game starts (class cd) and one for timing the player (class game_timer). Both classes work similarly but with different values. It imports the multimedia library, pyglet. There is also a global variable time_alloted initialized with a value of 1.

Class cd: Initializing Function

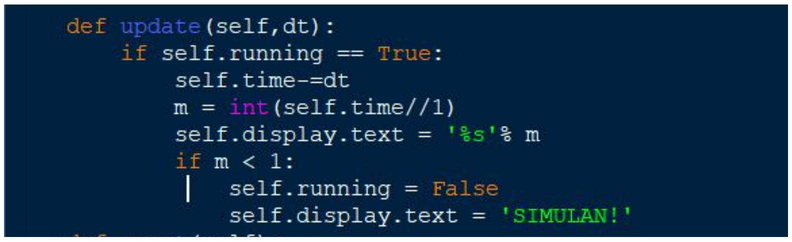
This function initializes the values of self.start (the starting value of the countdown), self.running (whether or not the countdown is starting, and self.time (the total time it takes to countdown) and the format for its display.



```python
    def __init__(self):
        self.start = '3'
        self.display = pyglet.text.Label(self.start,font_size=65,x=1180//2, y=590//2,
                                         anchor_x = 'center', anchor_y = 'center')
        self.running = False
        self.time = 5
        self.display.color = (255, 255, 255, 255)
```

Class cd: Update Function

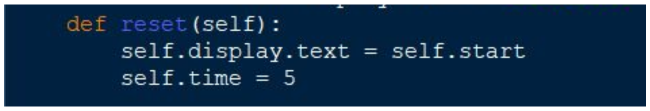
This function continually updates the values of self.time and the displayed text. If the countdown is ongoing, the value of self.time is decremented as the current time changes.



```python
    def update(self,dt):
        if self.running == True:
            self.time-=dt
            m = int(self.time//1)
            self.display.text = '%s'% m
            if m < 1:
                self.running = False
                self.display.text = 'SIMULAN!'
```

Class cd: Reset Function

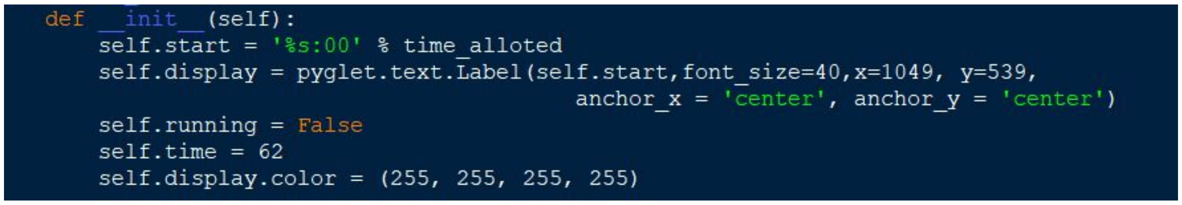
This function resets the values of self.time and the displayed text back to the initialized value.



```
    def reset(self):
        self.display.text = self.start
        self.time = 5
```

Class game_timer: Initializing Function

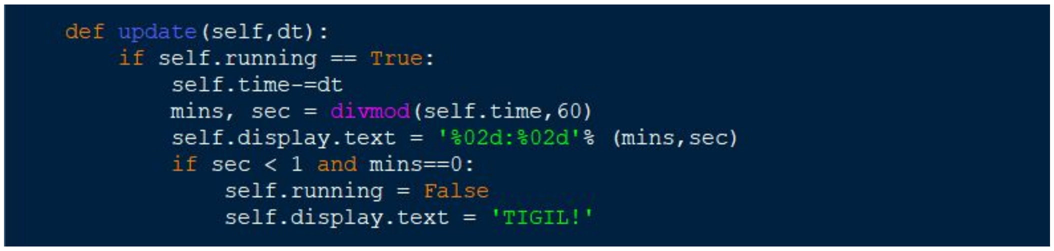
This function initializes the values of self.start, self.time, and the displayed text. self.start is initialized with a formatted value of time_alloted. It also initializes the format for the display.



```
    def __init__(self):
        self.start = '%s:00' % time_alloted
        self.display = pyglet.text.Label(self.start,font_size=40,x=1049, y=539,
                                         anchor_x = 'center', anchor_y = 'center')
        self.running = False
        self.time = 62
        self.display.color = (255, 255, 255, 255)
```

Class game_timer: Update Function

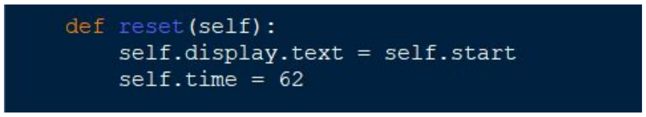
This function continually updates the the values of self.time and the displayed text. If the timer is ongoing, self.time is decremented by the change current time. The displayed text for this function is in minutes and seconds which values depend on the continuously decremented value of self.time.



```
    def update(self,dt):
        if self.running == True:
            self.time-=dt
            mins, sec = divmod(self.time,60)
            self.display.text = '%02d:%02d'% (mins,sec)
            if sec < 1 and mins==0:
                self.running = False
                self.display.text = 'TIGIL!'
```

Class game_timer: Reset Function

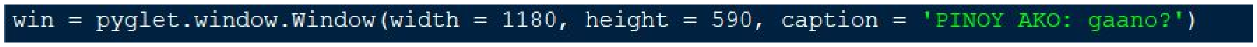
This function resets the values of self.time and the displayed text back to the initialized value.



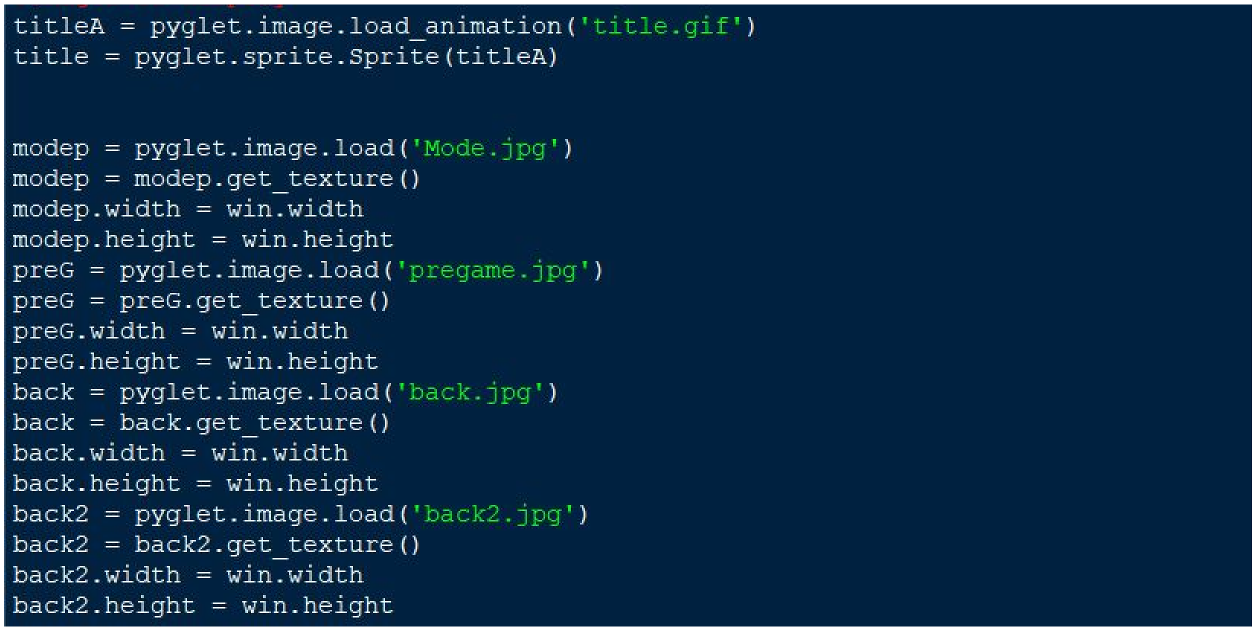
```
    def reset(self):
        self.display.text = self.start
        self.time = 62
```

4. INTERFACE

The interface serves as the overall execution file. It heavily contains pyglet codes for the user interface and utilizes the user-defined engine, QQQ, and timer modules. A window is first created by using pyglet's window.Window function, followed by loading the different images needed for the game's user interface.



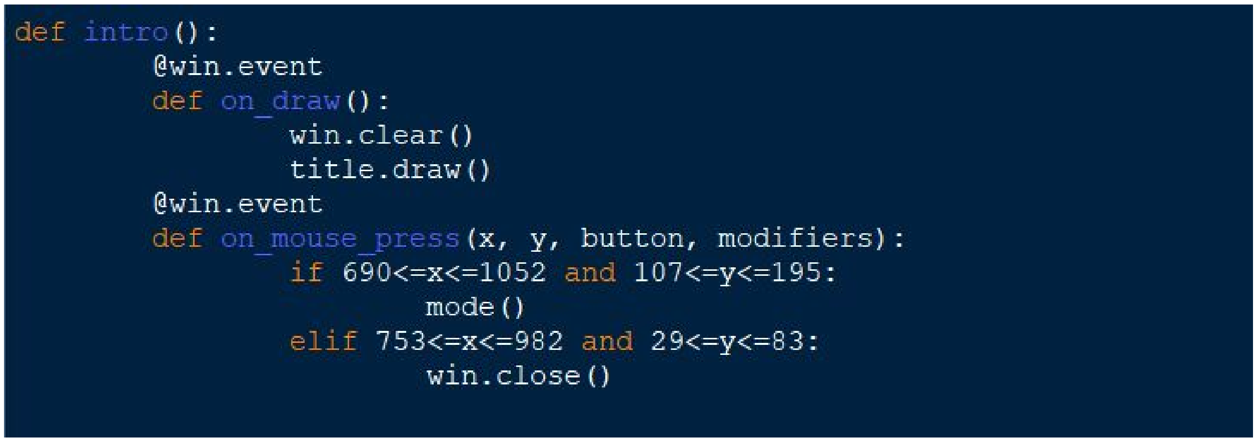
```
win = pyglet.window.Window(width = 1180, height = 590, caption = 'PINOY AKO: gaano?')
```



```
titleA = pyglet.image.load_animation('title.gif')
title = pyglet.sprite.Sprite(titleA)


modep = pyglet.image.load('Mode.jpg')
modep = modep.get_texture()
modep.width = win.width
modep.height = win.height
preG = pyglet.image.load('pregame.jpg')
preG = preG.get_texture()
preG.width = win.width
preG.height = win.height
back = pyglet.image.load('back.jpg')
back = back.get_texture()
back.width = win.width
back.height = win.height
back2 = pyglet.image.load('back2.jpg')
back2 = back2.get_texture()
back2.width = win.width
back2.height = win.height
```

Intro Function

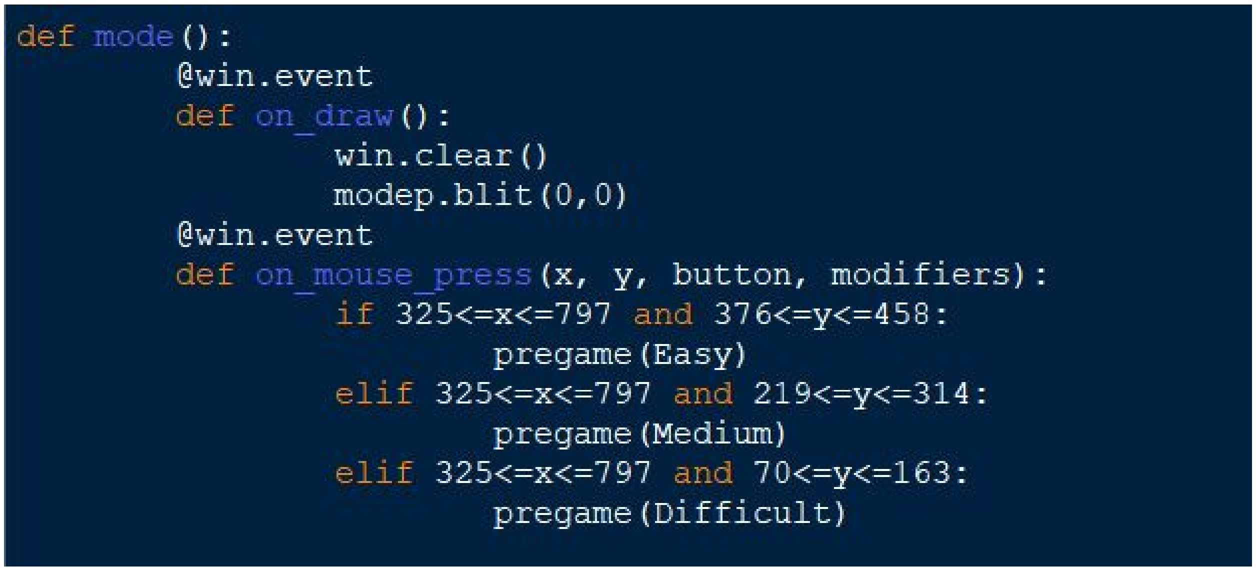
This function displays the intro page of the game. An inner function, on_draw, draws on the window and accesses the title variable that stored the image for the opening graphics. Another inner function, on_mouse_press, identifies which area the mouse is on the window and depending on the range of the location of the mouse click, the window will either be closed or the next funtion, mode will be accessed.



```
def intro():
        @win.event
        def on_draw():
                win.clear()
                title.draw()
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 690<=x<=1052 and 107<=y<=195:
                        mode()
                elif 753<=x<=982 and 29<=y<=83:
                        win.close()
```

Mode Function

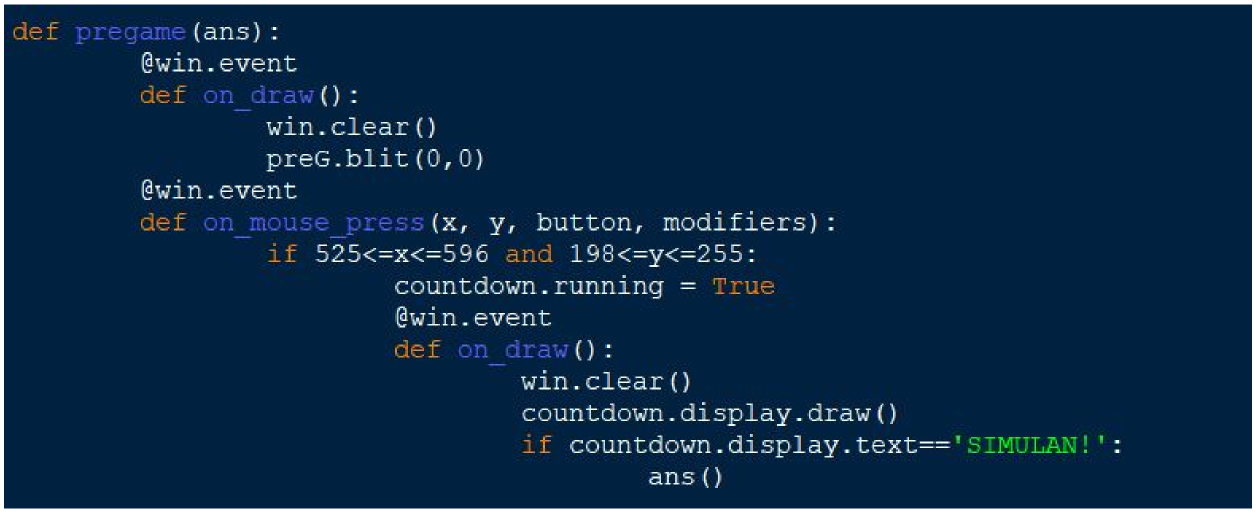
This function displays the level or mode choices of the game. The on_draw function clears the display on the window and replaces it with the image stored in modep. The on_mouse_press function then determines which part of the window is being clicked and that accessed the pregame of the corresponding level.



```
def mode():
        @win.event
        def on_draw():
                win.clear()
                modep.blit(0,0)
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 325<=x<=797 and 376<=y<=458:
                        pregame(Easy)
                elif 325<=x<=797 and 219<=y<=314:
                        pregame(Medium)
                elif 325<=x<=797 and 70<=y<=163:
                        pregame(Difficult)
```

Pregame Function

This function displays the graphics for starting the countdown. It takes a function name as an argument. Like the previous functions, the window is cleared and the image stored in preG is displayed. The on_mouse_press determines where the mouse is clicked, and if within the right range, it will start the countdown and accesses the function used as an argument

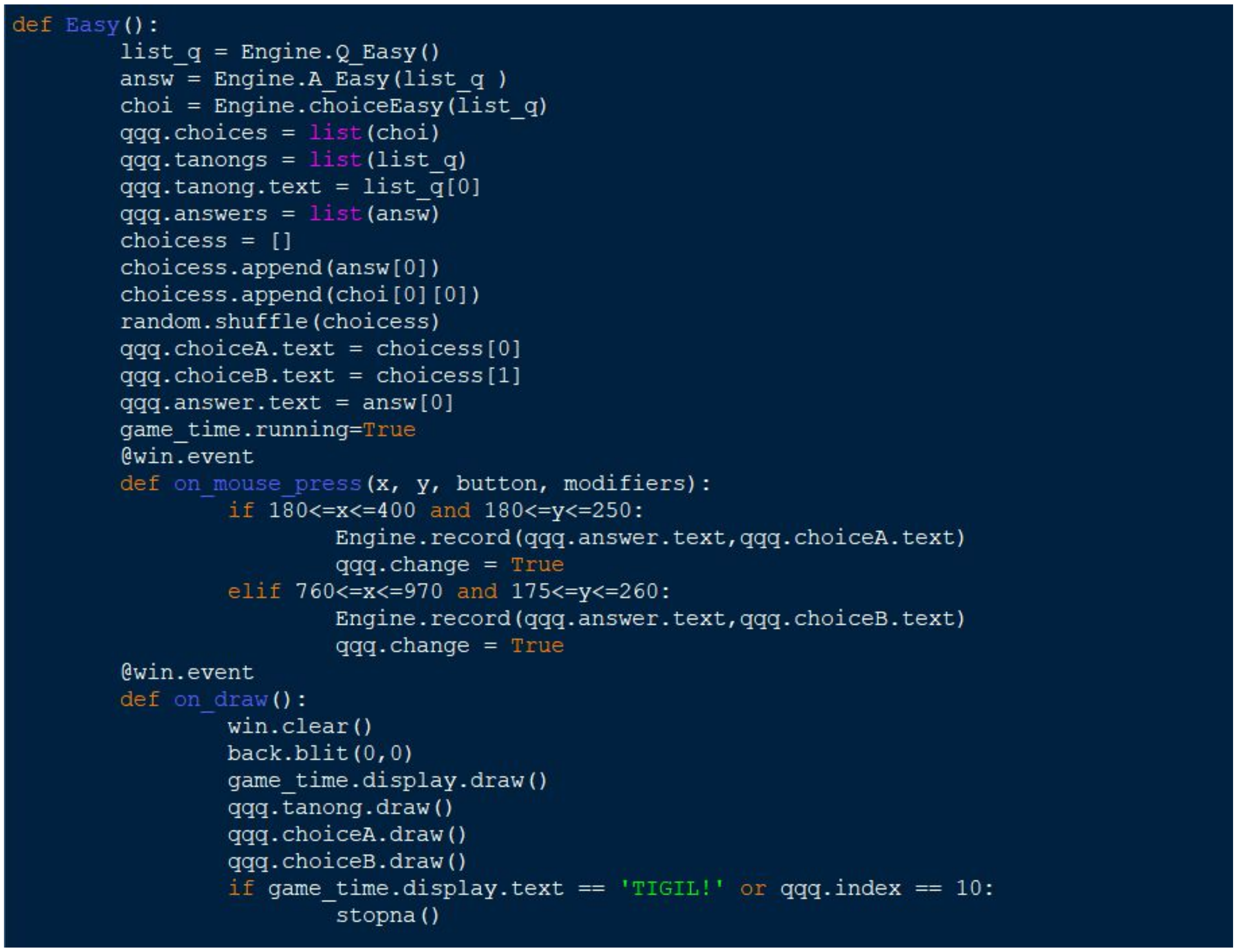


```
def pregame(ans):
        @win.event
        def on_draw():
                win.clear()
                preG.blit(0,0)
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 525<=x<=596 and 198<=y<=255:
                        countdown.running = True
                        @win.event
                        def on_draw():
                                win.clear()
                                countdown.display.draw()
                                if countdown.display.text=='SIMULAN!':
                                        ans()
```

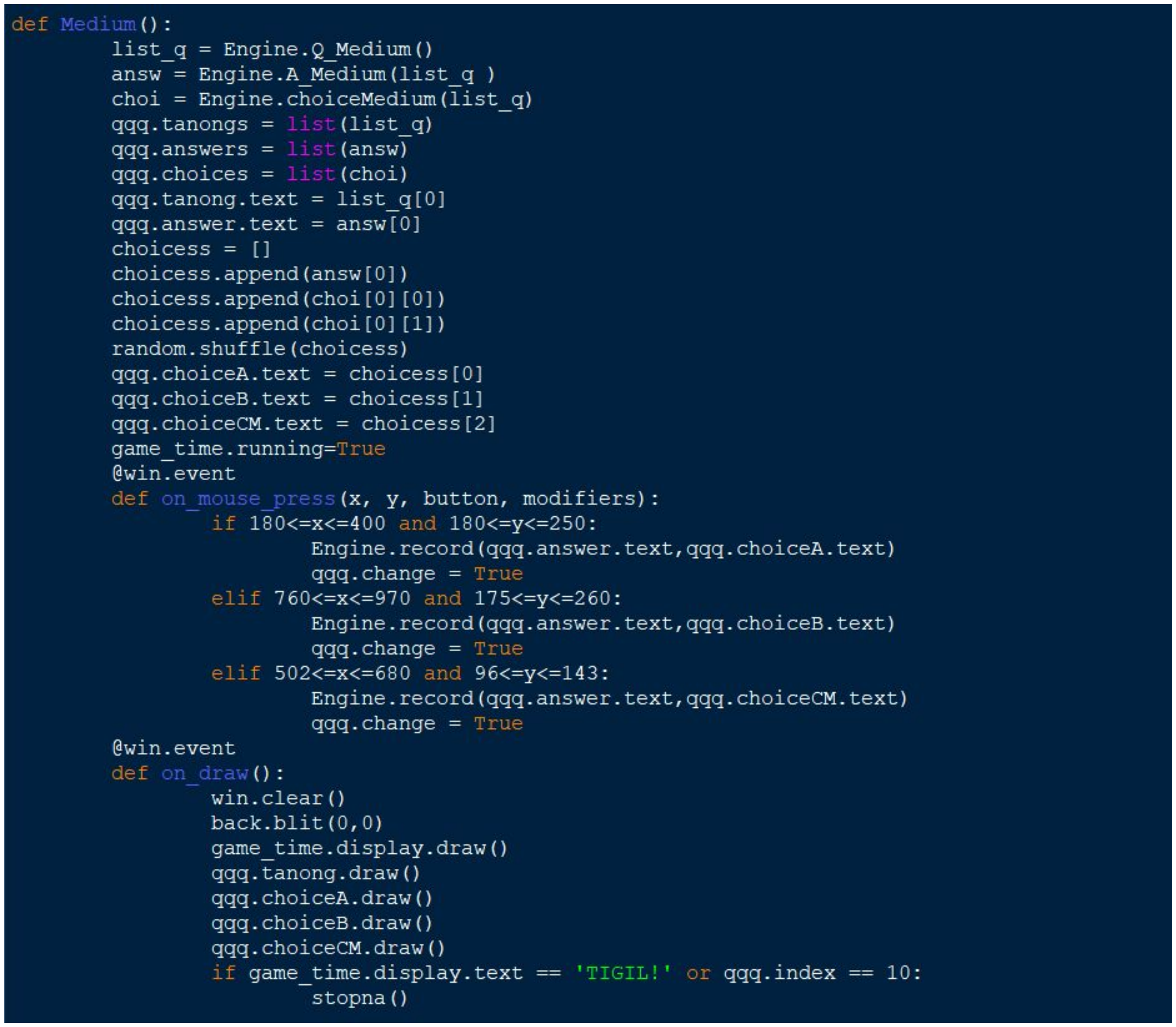
Level/Mode Functions

Each of the levels of the game has a function used to call the corresponding list of questions, choices, and answers made for that level. All of the function hav similar structure but differ in the number of choices.

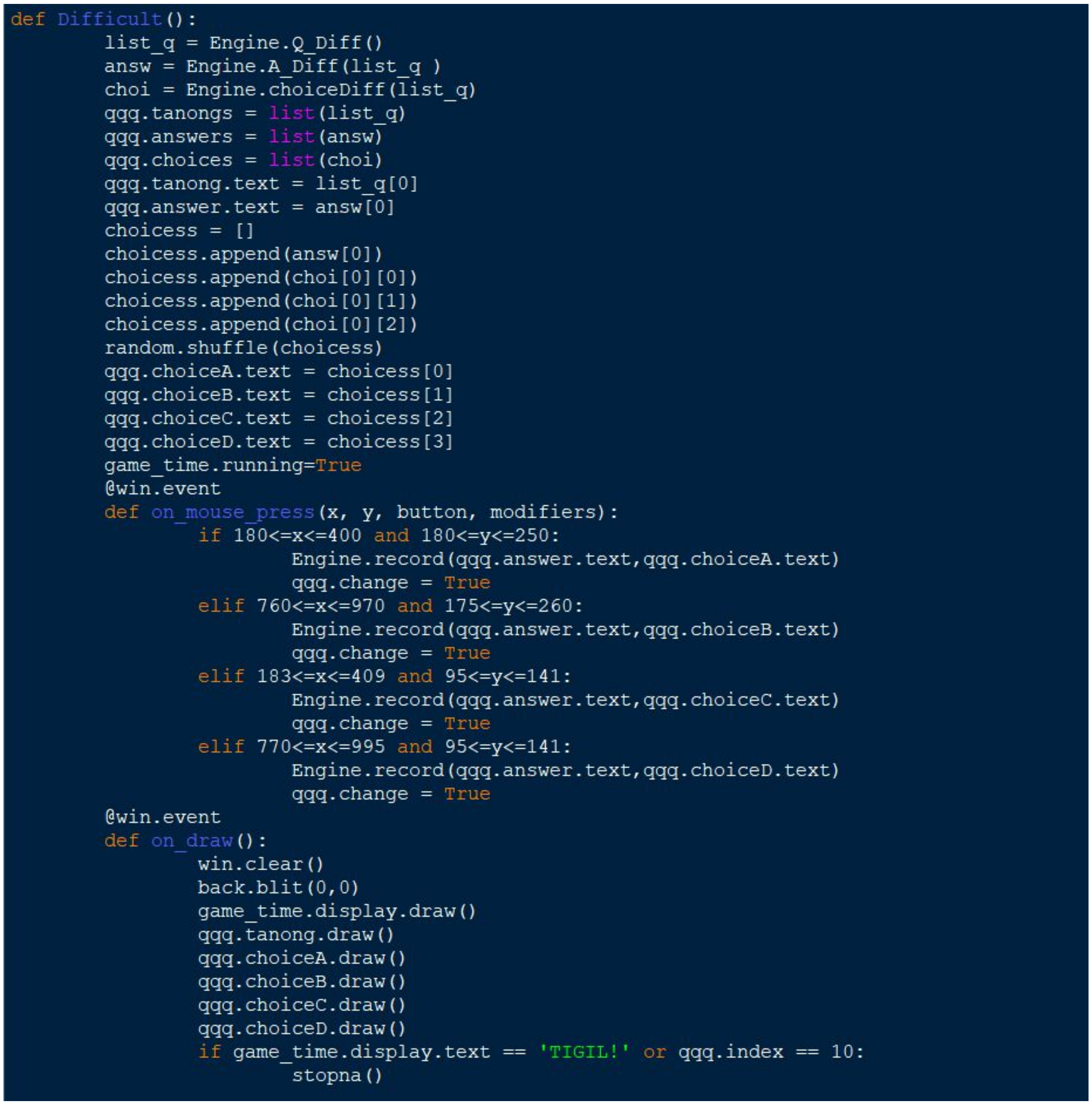
First, lists of random questions, their corresponding answers, and choices are called by using the engine module. The lists are then formatted using the QQQ module. The random formatted question and choices are then displayed and the player’s timer then starts . The on_mouse_press function then determines which area of the window the mouse clicks on to determine which choice the player picked. The on_draw function then clears the window and draws a new set of question and choices. If the timer displays “Tigil” or if 10 questions have been answered, the game is stopped.



```
def Easy():
        list_q = Engine.Q_Easy()
        answ = Engine.A_Easy(list_q )
        choi = Engine.choiceEasy(list_q)
        qqq.choices = list(choi)
        qqq.tanongs = list(list_q)
        qqq.tanong.text = list_q[0]
        qqq.answers = list(answ)
        choicess = []
        choicess.append(answ[0])
        choicess.append(choi[0][0])
        random.shuffle(choicess)
        qqq.choiceA.text = choicess[0]
        qqq.choiceB.text = choicess[1]
        qqq.answer.text = answ[0]
        game_time.running=True
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 180<=x<=400 and 180<=y<=250:
                        Engine.record(qqq.answer.text,qqq.choiceA.text)
                        qqq.change = True
                elif 760<=x<=970 and 175<=y<=260:
                        Engine.record(qqq.answer.text,qqq.choiceB.text)
                        qqq.change = True
        @win.event
        def on_draw():
                win.clear()
                back.blit(0,0)
                game_time.display.draw()
                qqq.tanong.draw()
                qqq.choiceA.draw()
                qqq.choiceB.draw()
                if game_time.display.text == 'TIGIL!' or qqq.index == 10:
                        stopna()
```



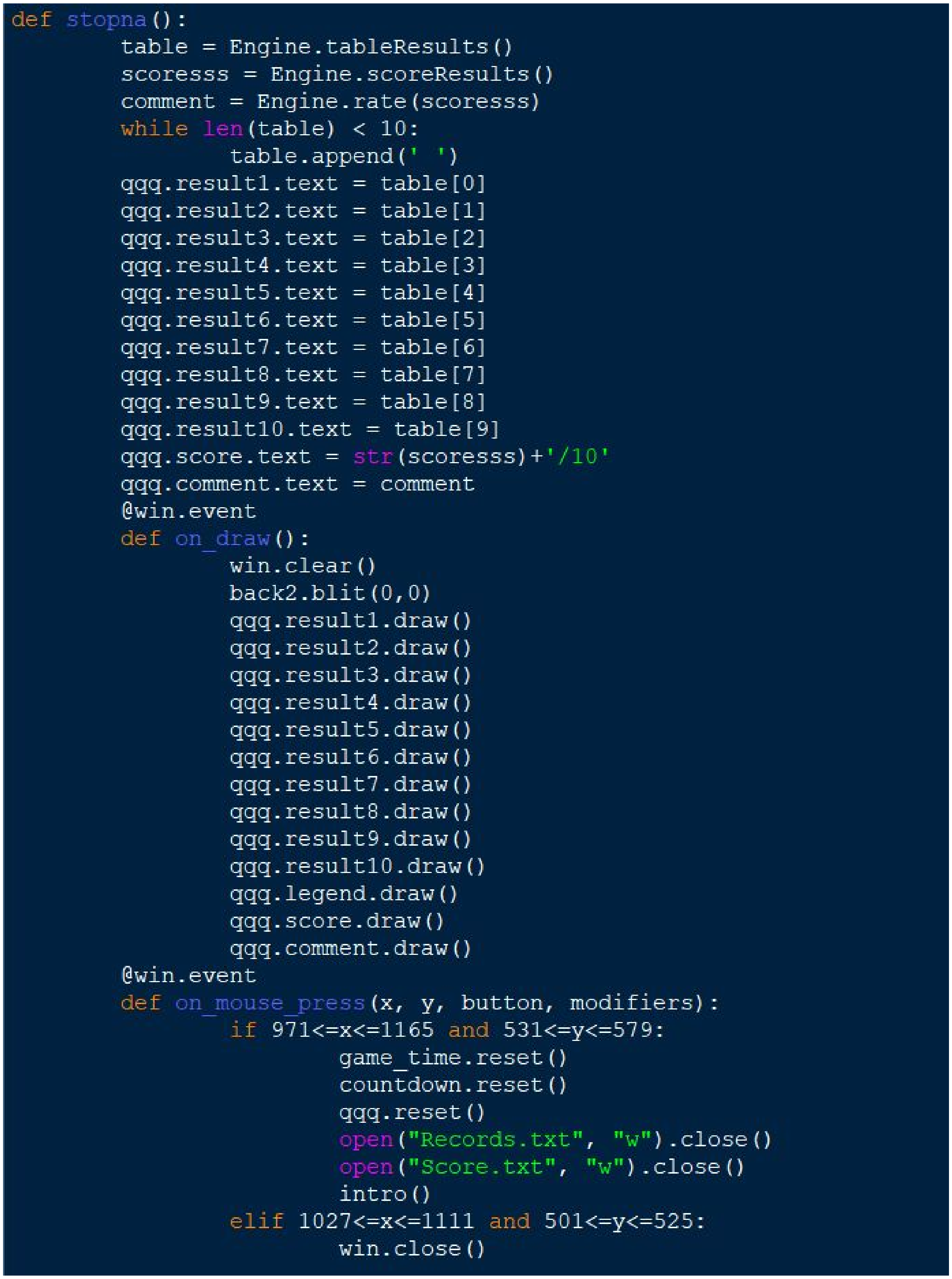
```
def Medium():
        list_q = Engine.Q_Medium()
        answ = Engine.A_Medium(list_q )
        choi = Engine.choiceMedium(list_q)
        qqq.tanongs = list(list_q)
        qqq.answers = list(answ)
        qqq.choices = list(choi)
        qqq.tanong.text = list_q[0]
        qqq.answer.text = answ[0]
        choicess = []
        choicess.append(answ[0])
        choicess.append(choi[0][0])
        choicess.append(choi[0][1])
        random.shuffle(choicess)
        qqq.choiceA.text = choicess[0]
        qqq.choiceB.text = choicess[1]
        qqq.choiceCM.text = choicess[2]
        game_time.running=True
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 180<=x<=400 and 180<=y<=250:
                        Engine.record(qqq.answer.text,qqq.choiceA.text)
                        qqq.change = True
                elif 760<=x<=970 and 175<=y<=260:
                        Engine.record(qqq.answer.text,qqq.choiceB.text)
                        qqq.change = True
                elif 502<=x<=680 and 96<=y<=143:
                        Engine.record(qqq.answer.text,qqq.choiceCM.text)
                        qqq.change = True
        @win.event
        def on_draw():
                win.clear()
                back.blit(0,0)
                game_time.display.draw()
                qqq.tanong.draw()
                qqq.choiceA.draw()
                qqq.choiceB.draw()
                qqq.choiceCM.draw()
                if game_time.display.text == 'TIGIL!' or qqq.index == 10:
                        stopna()
```



```
def Difficult():
        list_q = Engine.Q_Diff()
        answ = Engine.A_Diff(list_q )
        choi = Engine.choiceDiff(list_q)
        qqq.tanongs = list(list_q)
        qqq.answers = list(answ)
        qqq.choices = list(choi)
        qqq.tanong.text = list_q[0]
        qqq.answer.text = answ[0]
        choicess = []
        choicess.append(answ[0])
        choicess.append(choi[0][0])
        choicess.append(choi[0][1])
        choicess.append(choi[0][2])
        random.shuffle(choicess)
        qqq.choiceA.text = choicess[0]
        qqq.choiceB.text = choicess[1]
        qqq.choiceC.text = choicess[2]
        qqq.choiceD.text = choicess[3]
        game_time.running=True
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 180<=x<=400 and 180<=y<=250:
                        Engine.record(qqq.answer.text,qqq.choiceA.text)
                        qqq.change = True
                elif 760<=x<=970 and 175<=y<=260:
                        Engine.record(qqq.answer.text,qqq.choiceB.text)
                        qqq.change = True
                elif 183<=x<=409 and 95<=y<=141:
                        Engine.record(qqq.answer.text,qqq.choiceC.text)
                        qqq.change = True
                elif 770<=x<=995 and 95<=y<=141:
                        Engine.record(qqq.answer.text,qqq.choiceD.text)
                        qqq.change = True
        @win.event
        def on_draw():
                win.clear()
                back.blit(0,0)
                game_time.display.draw()
                qqq.tanong.draw()
                qqq.choiceA.draw()
                qqq.choiceB.draw()
                qqq.choiceC.draw()
                qqq.choiceD.draw()
                if game_time.display.text == 'TIGIL!' or qqq.index == 10:
                        stopna()
```

Stop Na Function

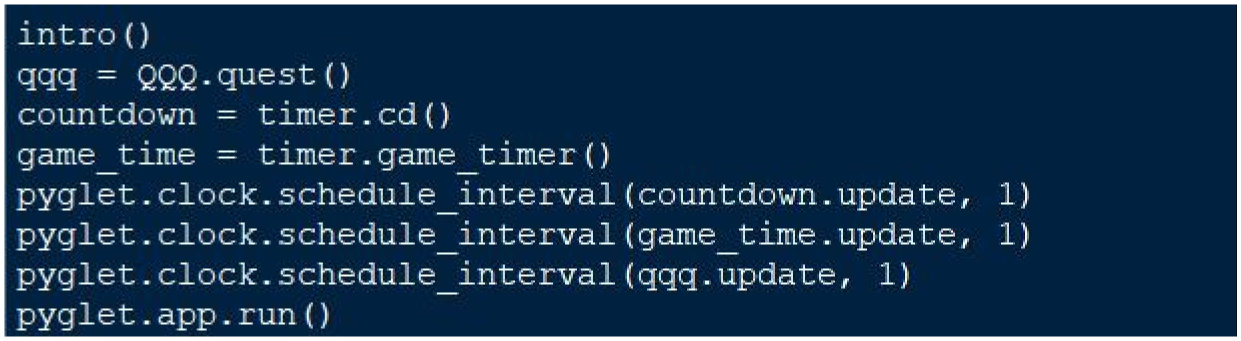
This function is accessed after each of the levels are played. It stops the game and displays the results. The correct answers, the player's answers, the player's score, and the player's rating are accessed by the engine module. Then, using the QQQ module, these details are formatted and displayed using the on_draw function. The on_mouse_press function again determines where on the window the mouse clicks. Depending on the area being clicked, the game may end by the window closing, or the game will reset along with the timer and the countdown.



```
def stopna():
        table = Engine.tableResults()
        scoresss = Engine.scoreResults()
        comment = Engine.rate(scoresss)
        while len(table) < 10:
                table.append(' ')
        qqq.result1.text = table[0]
        qqq.result2.text = table[1]
        qqq.result3.text = table[2]
        qqq.result4.text = table[3]
        qqq.result5.text = table[4]
        qqq.result6.text = table[5]
        qqq.result7.text = table[6]
        qqq.result8.text = table[7]
        qqq.result9.text = table[8]
        qqq.result10.text = table[9]
        qqq.score.text = str(scoresss)+'/10'
        qqq.comment.text = comment
        @win.event
        def on_draw():
                win.clear()
                back2.blit(0,0)
                qqq.result1.draw()
                qqq.result2.draw()
                qqq.result3.draw()
                qqq.result4.draw()
                qqq.result5.draw()
                qqq.result6.draw()
                qqq.result7.draw()
                qqq.result8.draw()
                qqq.result9.draw()
                qqq.result10.draw()
                qqq.legend.draw()
                qqq.score.draw()
                qqq.comment.draw()
        @win.event
        def on_mouse_press(x, y, button, modifiers):
                if 971<=x<=1165 and 531<=y<=579:
                        game_time.reset()
                        countdown.reset()
                        qqq.reset()
                        open("Records.txt", "w").close()
                        open("Score.txt", "w").close()
                        intro()
                elif 1027<=x<=1111 and 501<=y<=525:
                        win.close()
```

Executing Code

This part of the module executes the entire game. It calls the intro function to display the start of the game and after the clock interval for both timer module functions are set, the game is run using pyglet.app.run().



```
intro()
qqq = QQQ.quest()
countdown = timer.cd()
game_time = timer.game_timer()
pyglet.clock.schedule_interval(countdown.update, 1)
pyglet.clock.schedule_interval(game_time.update, 1)
pyglet.clock.schedule_interval(qqq.update, 1)
pyglet.app.run()
```

5. How to play the game (Assumptions)

The player can access the game through opening interface.py in the terminal of their device. It only requires the use of the mouse or the mouse pad. The player can choose the level, and their answer only through clicking.

6. Sources

GIF and images used in the game

http://www.busyok.info/2013/02/flag-boy.html
https://www.vectorstock.com/royalty-free-vector/game-background-2d-application-design-vector-9827371
http://www.3dflagsplus.com/2015/11/philippines-animated-flags-pictures.html
https://yandex.ua/collections/card/5b5099cde3c188008790a582/

Basis for the questions in the game

https://www.proprofs.com/quiz-school/story.php?title=grade-6-quiz-about-history--philippines
https://bayaningfilipino.blogspot.com/2017/10/mga-bayani-ng-pilipinas-at-ang-kanilang.html
http://mgamakasaysayanpooksaatingbansa.blogspot.com/
https://www.academia.edu/28149067/FAMOUS_FILIPINO_ARTISTS_AND_THEIR_ART_WORKS
https://quizlet.com/6618098/makasaysayang-lugar-sa-pilipinas-flash-cards/
https://tl.wikipedia.org/wiki/Kasaysayan_ng_Pilipinas
https://tl.wikipedia.org/wiki/Pilipinas
https://takdangaralin.ph/magagandang-tanawin-sa-pilipinas/
https://www.tagaloglang.com/national-symbols-of-the-philippines/
https://www.kapitbisig.com/philippines/philippine-map-and-national-symbols-pambansang-sagisag-philippine-national-symbols_305.html
https://pambansangsimbolongpilipinas.wordpress.com/2016/12/03/mga-pambansang-sagisag-ng-pilipinas/
https://pinoycollection.com/mga-bugtong/