Binus University International

Simple Doodle Jump in Python

Name : Stephanus Jovan Novarian

NIM : 2201832856

Class: L1CC

Lecturer: Mr. Jude Joseph Lamug Martinez

Task : Program’s Final Project

Computer Science

Introduction to Programming/Program Design Methods

Contents

[1. Introduction 3](#_Toc530669843)

[2. Simple Syntaxes explaination and UML 4](#_Toc530669844)

[3. Class Diagram 5](#_Toc530669845)

[4. Code explaination 6](#_Toc530669846)

[4.1 Main 6](#_Toc530669847)

[4.2 Character/Dino 9](#_Toc530669848)

[4.3 Platform/Platform Manager 13](#_Toc530669849)

[4.4 Scoreboard/Score 16](#_Toc530669850)

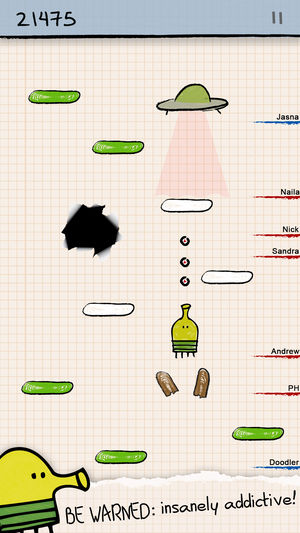
[5. Summary of the Final Project 17](#_Toc530669851)

[6. References 18](#_Toc530669852)

# Introduction

Games have always been a very intresting part for me as a teenager to enjoy, but do they know how to generate a simple game with the power of programming. As I joined Comptuer Science , I learnt a lot of logics that can be implement towards the program. In this Project , the purpose of it was to create one of my favourite game before i get into university which is call the “Doodle Jump” .

So this game is very easy to play, the player just need to control the movement of the character, and jump as high as you can through different platforms. Which will be arrange based on the code. I decided to research from some games which is similar to “Doodle Jump” but more simpler one so I can get some ideas of what should I do and also what are the general syntax and also what are the logics which are based on the programming language that can be coded.



# Simple Syntaxes explaination and UML

There were a lot of methods that was generated plus logics implemented towards the code to run the pygame based on the programming language. The methods that has been used were mainly Class, Dictionaries,Lists and also diffrent loops coded in diffrent function.

From what i learn in the lecture , Class is define as a method in object oriented programming, method define with the self value as to use as initializer or to be used inside a function to work which will make the program works. Function is also a method to generate a logic start with def(), ends with the define functoin name called out. def name(), call out name() to run.

Here is the UML of the pygame :

Character Initialized on the Middle

Character Movement

Setting (Platform Manager)

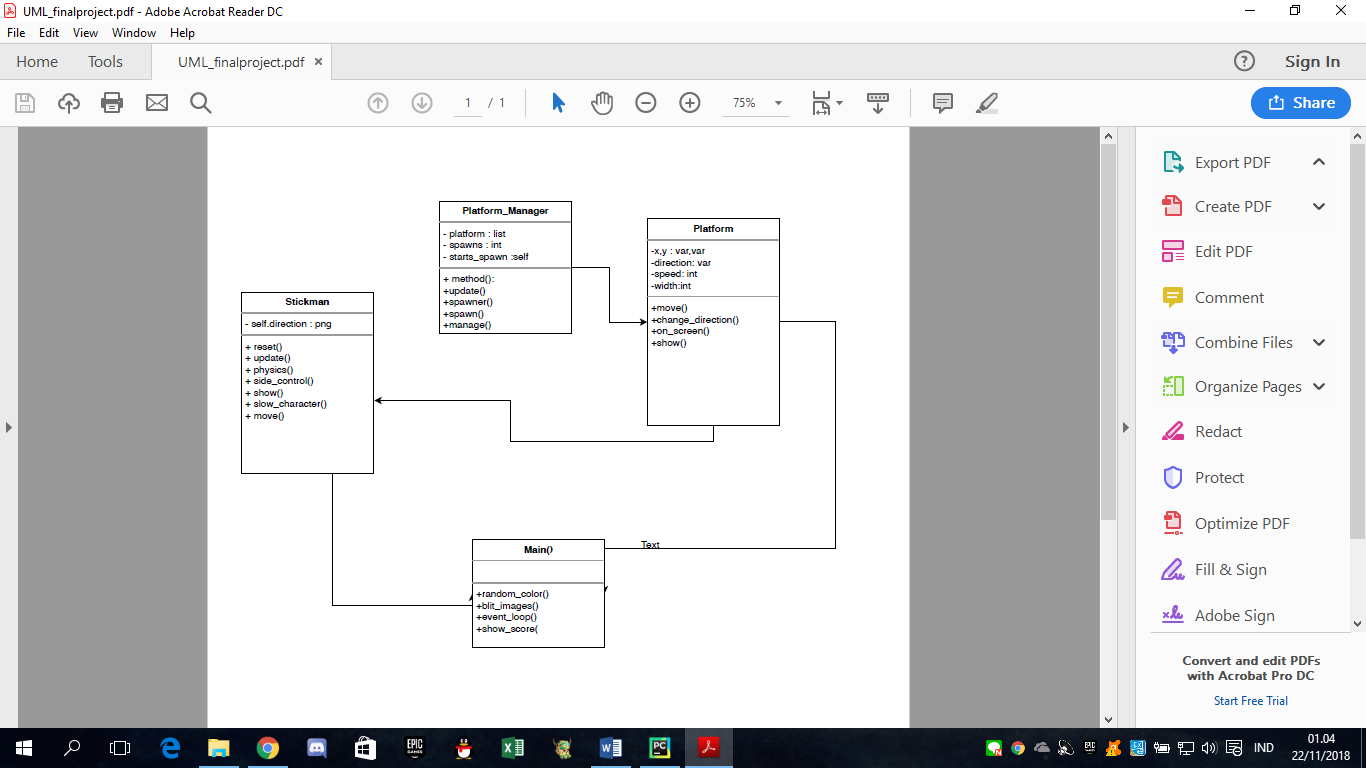
Platform movement on Screen

Show\_score()

Based on Character moving up and collision

# Class Diagram

Here is the class diagram for my final project:



# Code explaination

## 4.1 Main

from pygame import \*  
import pygame.font  
import random  
  
window\_x = 500  
window\_y = 550  
  
# The main and the initial setting for the while True loop that display the screen  
init()  
window = display.set\_mode((window\_x, window\_y))  
bg = pygame.image.load('wallpaper.jpg').convert()  
display.set\_caption('Help Linux Jump!')  
clock = time.Clock()

Obviously to create a pyagme we import pygame first as well as pygame.font for making the score appear in a str condition. (str 🡪 font)

init() for the code of initiation, the variable window set the display mode of horizontal and vertical value of the screen , Help Linux Jump is the title of the pygame, the bg variable is for

making the background of the pygame which will be show on the code windows.blit().

1. info = {  
    'screen\_y': 0,  
    'score': 0,  
    'high\_score': 0  
   }  
     
   # call out the function from the c  
   stick\_man = Stick\_Man()  
   platform\_manager = Platform\_Manager()  
     
   while True:  
    # Place the score for the screen  
     
    event\_loop()  
     
    platform\_blit = platform\_manager.update()  
    stick\_blit = stick\_man.update(platform\_blit)  
    info['screen\_y'] = min(min(0, stick\_blit[1][1] - window\_y \* 0.4), info['screen\_y'])  
    info['score'] = (-stick\_blit[1][1] + 470) / 50  
     
    print(stick\_blit[1][1], info['screen\_y'])

if stick\_blit[1][1] - 470 > info['screen\_y']:  
 info['score'] = 0  
 info['screen\_y'] = 0  
 stick\_man = Stick\_Man()  
 platform\_manager = Platform\_Manager()  
  
 clock.tick(60)  
  
 # Displaying the  
 window.blit(bg,[0,0])  
  
 blit\_images([stick\_blit])  
  
 for x in platform\_blit:  
 i = list(x)  
 i[1] = list(i[1])  
 i[1][1] -= info['screen\_y']  
 draw.rect(window, i[0], i[1])  
  
 info['high\_score'] = max(info['high\_score'], info['score'])  
  
 show\_score(info['score'], 1)  
 show\_score(info['high\_score'], 0)  
  
 display.update()

Info is the type of dictionary which contains the initiaztion values of 3 variables.

While True contains all the loops which will be showed inside the pygame screen , the game starts already with the platform working based on Platform\_manager class, so that is why the code of class function was put on the top below the Info dict as well as the character customization based on the Stickman Class of characters so it enables the character could move as well the rect to blit in the pyagame. the update() put in the while loop so it will work on the pygame screen.

The character initiate at the position based on the loop formula as shown in the coding.

The score and high score of the game was placed as a rect on the left and the right of the pygame screen initiated as 0 since its still standing state. The platform also blit based on the Class that has been set by the program codes

To sum up , all of the code above was to blit everything in the screen while True with the game function worked already as well.

## 4.2 Character/Dino

Here are the codes that is used for generate the characters movement and initiaztion.

class Stick\_Man:  
 # Class for the the Stick\_man , so that it initiate the characters and also the the pictures when the  
 # character wants to move in various movement.  
 def \_\_init\_\_(self):  
 self.crouch = image.load('stickman.png')  
 self.fall = image.load('stickman.png')  
 self.jumping\_right = image.load('stickman.png')  
 self.jumping\_left = transform.flip(self.jumping\_right, True, False)  
 self.stand = image.load('stickman.png')  
  
 self.reset()  
  
  
 def reset(self):  
 # this function is used for reset, so that if the characters dead, its reset into this condition.  
 self.speed\_x = 0  
 self.speed\_y = 0  
 self.max\_speed\_x = 5  
 self.max\_speed\_y = 15  
 self.x\_acceleration = 0.5  
 self.img = self.jumping\_right  
 self.jump\_speed = 15  
  
 scale = 7  
 self.width, self.height = 7 \* scale, 12 \* scale  
 self.scale = scale  
  
 self.x = (window\_x - self.width) / 2  
 self.y = window\_y - self.height  
  
 def update(self, p):  
 # this function is for , after the reset/initial, it updates the movement when clicked ,in movement detection.  
 # it contains function for side\_controls of the wallpaper screen and also movement and show.  
 self.side\_control()  
 self.physics(p)  
 self.move()  
 self.show()  
  
 self.x += self.speed\_x  
 self.y -= self.speed\_y  
  
 return (self.img, (self.x, self.y, self.width, self.height))

As we can see from this codes that was generated , first it initialized the characters picture stickman.png which was from the same file as the python file’s location. The class for the character was created with all the self values in it, I use different images in the crouch and also the jump so that as the character jump , the picture will changed variouly depends on the movement that was set. The self.reset() was made to as the character dead it resets into the state where its standing without any movement generated on the pygame window( the pygame screen), the update.(self,p) is for the update of the character.

Based on the scale as it generates a formula for the character movement updates. Based on the sidecontrol,physics(),move() and show() functions, and from those functions it consists these formulas, codes.

From the combination of the code for the character movement inside the update() function(each function will be explained below) , it returns the image (character) with function with the setted width and height.

def physics(self, p):  
  
 on = False  
  
 for colour, rect in p:  
 x, y, w, h = rect  
  
 # X range  
 if self.x + self.width / 2 > x and self.x - self.width / 2 < x + w:  
 # Y range  
 if self.y + self.height >= y and self.y + self.height <= y + h:  
  
 if self.speed\_y < 0:  
 on = True  
  
 if not on and not self.y >= window\_y - self.height:  
 self.speed\_y -= 0.5  
 elif on:  
 self.speed\_y = self.jump\_speed  
 else:  
 self.y = window\_y - self.height  
 self.speed\_x = 0  
 self.speed\_y = 0  
 if self.x != (window\_x - self.width) / 2:  
 if self.x > (window\_x - self.width) / 2:  
 self.x = max((window\_x - self.width) / 2, self.x - 6)  
 else:  
 self.x = min((window\_x - self.width) / 2, self.x + 6)  
  
 else:  
 keys = key.get\_pressed()  
 if keys[K\_SPACE]:  
 self.speed\_y = self.jump\_speed  
  
def side\_control(self):  
 if self.x + self.width < 0:  
 self.x = window\_x - self.scale  
 if self.x > window\_x:  
 self.x = -self.width  
  
def show(self):  
 if self.speed\_y > 0:  
 if self.speed\_x > 0:  
 self.img = self.jumping\_right  
 elif self.speed\_x < 0:  
 self.img = self.jumping\_left  
 else:  
 self.img = self.fall  
  
def slow\_character(self):  
 if self.speed\_x < 0: self.speed\_x = min(0, self.speed\_x + self.x\_acceleration / 6)  
 if self.speed\_x > 0: self.speed\_x = max(0, self.speed\_x - self.x\_acceleration / 6)  
  
def move(self):  
 keys = key.get\_pressed()  
  
 if not self.y >= window\_y - self.height:  
  
 if keys[K\_LEFT] and keys[K\_RIGHT]:  
 self.slow\_character()  
 elif keys[K\_LEFT]:  
 self.speed\_x -= self.x\_acceleration  
 elif keys[K\_RIGHT]:  
 self.speed\_x += self.x\_acceleration  
 else:  
 self.slow\_character()  
  
 self.speed\_x = max(-self.max\_speed\_x, min(self.max\_speed\_x, self.speed\_x))  
 self.speed\_y = max(-self.max\_speed\_y, min(self.max\_speed\_y, self.speed\_y))

First before the function began to work , on set to False means that the character is not in the state of moving by key\_press.

Basically physics() function contains formula equation to for setting of the character in initial state based on the window screen codes , the code was also used if the character did not reach the first platform, will return the character in the initial position which is the middle. (self.y = 0 )

side\_control() function is for the management if the character x rect value move towards the window.x screen., so that the character will return to the other side of the screen. As the character move towards left of the screen , it continues moving towards from the right of the screen.

show() function is for as the character jumps , fall or moving towards right and left, the image will change , based on the character \_init\_\_ that has been set. but in here I used the same image.

Move() function is for the keydown of the character , as the keyboard button pressed , the character will move towards left or right based on code. To add up the game itself only detect left and right movement after u press space to jump. As the character jumps to a platform , it will automatically jump without stopping.

Slow\_character() contains a one loop which enables the character to slow down when falling after jumping.

P.S: Too add up, in the formula which was intiated in self.speed. set to 0 to make them stay at the initate position or else as we open the game , the character will jump based on the values set, the higher the value the higher the character will jump

## 4.3 Platform/Platform Manager

platform\_spacing = 150  
  
  
class Platform\_Manager:  
 # This class is for the management of the platforms  
 def \_\_init\_\_(self):  
 self.platforms = []  
 self.spawns = 0  
 self.start\_spawn = window\_y  
  
 # the distance of each platforms movement heights  
 scale = 3  
 self.width, self.height = 24 \* scale, 6 \* scale  
  
 def update(self):  
 self.spawner()  
 return self.manage()  
  
 def spawner(self):  
 if window\_y - info['screen\_y'] > self.spawns \* platform\_spacing:  
 self.spawn()  
  
# This function is for appending the platforms as the character jump over the screen(self.spawns += 1)  
 # random movement and also added  
 def spawn(self):  
 y = self.start\_spawn - self.spawns \* platform\_spacing  
 x = random.randint(-self.width, window\_x)  
  
 self.platforms.append(Platform(x, y, random.choice([1, -1])))  
 self.spawns += 1  
  
 def manage(self):  
 u = []  
 b = []  
 for i in self.platforms:  
 i.move()  
 i.change\_direction()  
 b.append(i.show())  
  
 if i.on\_screen():  
 u.append(i)  
  
 self.platforms = u  
 return b

Class : Platform\_Manager , The class which is for the management , movement of the platform that will be coded in the other class which is the Platform Class. To sum up , the code above have the function to append the platform amount through self.y (vertical screen)

Spawner() function is for as the platform spawns above one platform , it makes a distance between spawn and the platform spacing that has been initiate at begining which is 150.

Spawn() function is for appending the rect of the platform based on the code Platform(x,y,randomchoice([-1,1]), random choice here means the position of the platform that are going to append, that is the reason when we restart the game, the platforms will be in random position , so its not the same everytime you want to play over and over.

Manage() function is for the platform movement management and also platoform rect append . initiate as a List and then the function was used inside platform then it appends in the screen.(from the code u.append[i])

class Platform:  
 # This class is for making the platform exist, generate the one platform so that it can be implemented  
 # in the platform manager as it arrange alr from the class above  
 def \_\_init\_\_(self, x, y, direction):  
 self.x = x  
 self.y = y  
 self.direction = direction  
 self.speed = 2  
 self.colour = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))  
 scale = 3  
 self.width, self.height = 24 \* scale, 6 \* scale  
  
 def move(self):  
 self.x += self.speed \* self.direction  
 self.change\_direction()  
  
 def change\_direction(self):  
 if self.x <= 0:  
 self.direction = 1  
 if self.x + self.width >= window\_x:  
 self.direction = -1  
  
 def on\_screen(self):  
 if self.y > info['screen\_y'] + window\_y:  
 return False  
 return True  
  
 def show(self):  
 return ((0, 0, 0), (self.x, self.y, self.width, self.height))

the x and y stands for the horizontal and vertical value/length of the screen that will be initiate for the rect of the platform, speed that will be used for the formula on move() set to 2 and the the width and height of the platform rect was set based on scale value.

Move() function is for the movement of the platform based on the formula of the initiation of speed and the direction, connnected to change\_direction() function which means that the platform movement boundaries could not go through window\_x(pygame screen), as in the game the platforms were only moving on the screen moving either left or right. The On\_screen() function was related to the change\_direction() function.

Show() function consists a syntax which returns the color of the platform rect which is black (0,0,0) with the width and height set in the program.

## 4.4 Scoreboard/Score

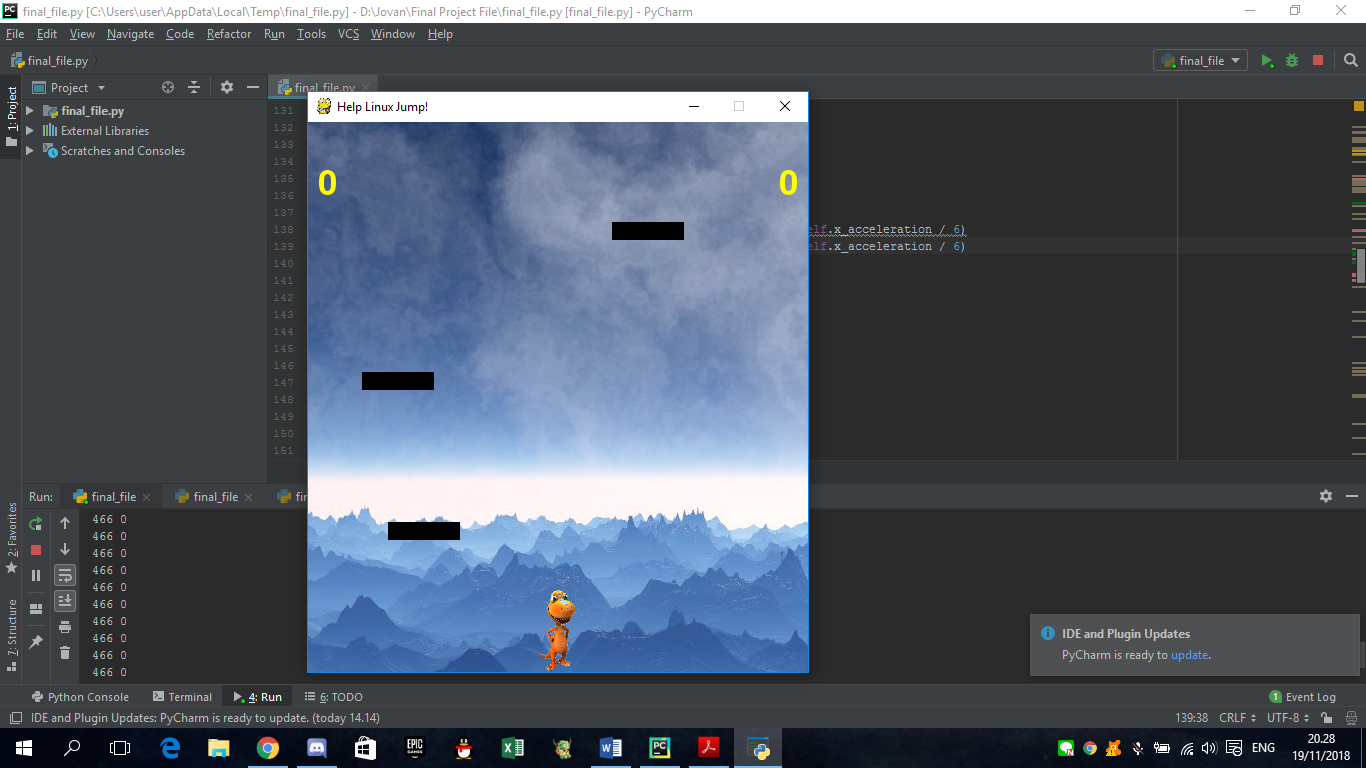
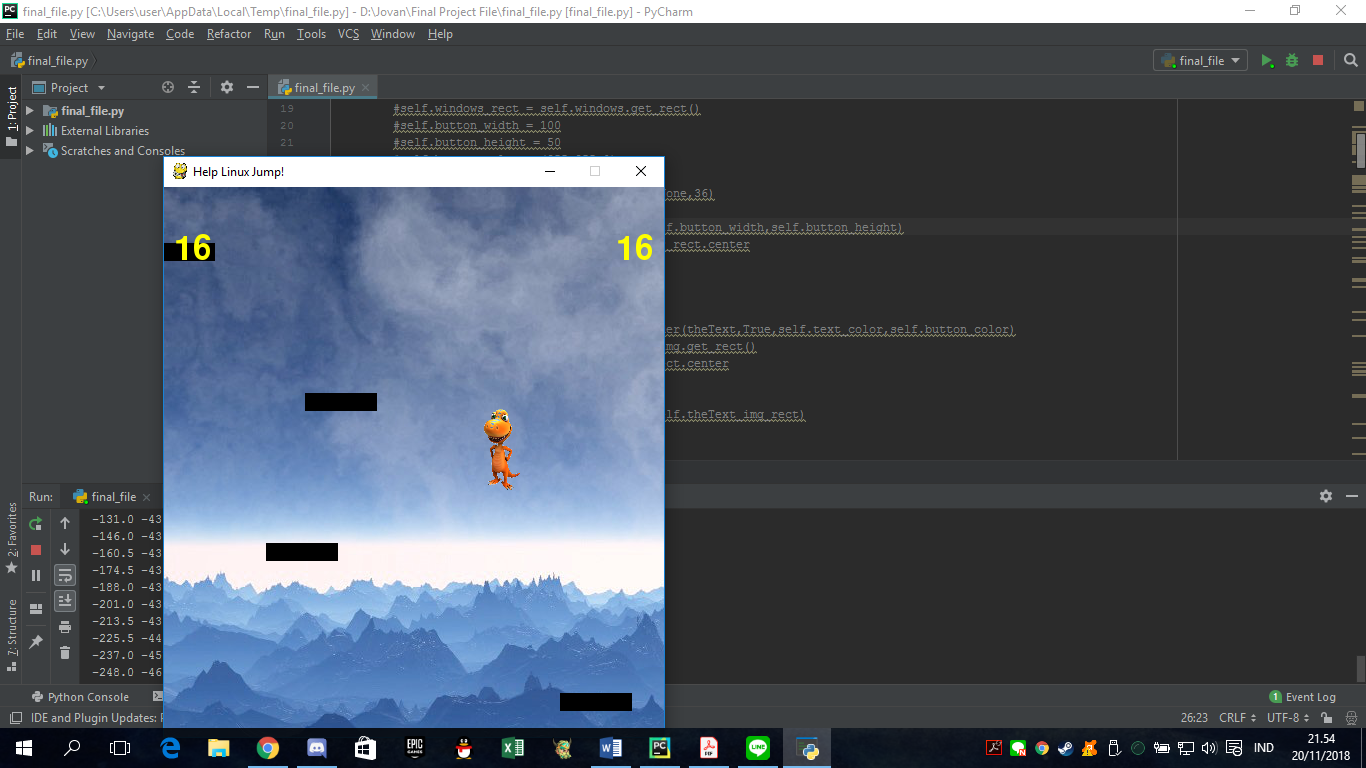
def show\_score(score, pos):  
 # This function is for showing the score , in str with color code as following , initiate as True  
 # on the screen. contains mathematical formula  
 message = f.render(str(round(score)), True, (255, 255, 0))  
 rect = message.get\_rect()  
  
 if pos == 0:  
 x = window\_x - rect.width - 10  
 else:  
 x = 10  
 y = rect.height + 10  
  
 # for showing the messege use the blit syntax  
  
 window.blit(message, (x, y))

The function show\_score(score,pos), is for generating scoreboard and also where to place it on the pygame screen, score set into a string and boolean state of True and also the color. Get\_rect() to get the rect(shape) of the score, pos is stands for position, by using the one loop to generate how later the score works and then the messege value will be show on the screen by window.blit(). As x and y values means the horizontal and vertical variable of the score rect.

# Summary of the Final Project

From this Final Project , I can say that the project was hard initially, but slowly , thanks to some refrences such as the python course book , free and open source websites and as well as question and some answers from Stack Overflow, enables me to understand how it works, makes me finished this project. The hardest part of the project is to understand and make the platform movement manager, since there were a lot of things that was needed to be consider such as the distance of plats and also random movement , where to initiate and also the mathematical formula that was needed for the arrangement.

The second difficulties of the project was the combination of the list and dict, I learn some of the new uses of dictionary inside the platform manager. Not only makes a group of word , numbers inside inside an arrangement.

Overall , I learnt a lot from of new syntaxes and also a lot of new logics from some refrences and online proffesor explaination, which I want to thank with. The Picture below shows the result of the initiation and also the score changes when the character jumps. Which is the best part that shows the pygame works fine.

# References

opensource.com. *Put Platforms in a Python game with Pygame*. 25 July 2018. Tuesday November 2018. <https://opensource.com/article/18/7/put-platforms-python-game>.

Tutorials, Python. *Jump and run in python*. Penyunt. Python Tutorials. 2017. Tuesday November 2018. <https://pythonspot.com/jump-and-run>.

Also I would like to say Thank you to some Phython experts who gave me the idea of making this python Project as well as the character picture taken from a CC used.