**Teacher’s Reference**

Pygame is a library in python which is used to design simple graphical interactive games.



The basic idea of this game is that it will rain down coins one by one and whenever the player clicks on a coin it will increase the score by 1 point and it will disappear from the screen as player has collected it. And a new coin will start to fall from a random place along x-axis at the top of the window. If the player miss the chance to collect it, then the coin will pass on through the bottom of the window and again a new coin will start falling and score will not increase. This goes on infinitely until player quits the game by closing the window.

**pip install pygame**

First, we need to install the pygame library & briefly explain to the students what pygame library is and why it is used. Though there are other libraries also like ‘arcade’ which can be used to design games in python

**1. import pygame**

Import the pygame library just like we import math and random.

**2. import random**

We are using this library for getting random number in some range. E.g. random number in range 1 to 500

**3. pygame.font.init()**

This library helps us to define font type and size of the text that we are putting inside pygame window

**4. pygame.init()**

This initialises the pygame. In other words it tells pygame to start and launch an empty pygame program. So, as a result of it an empty pygame program starts in the background but it will not show anything yet as we have not created a window yet.

**5. score = 0**

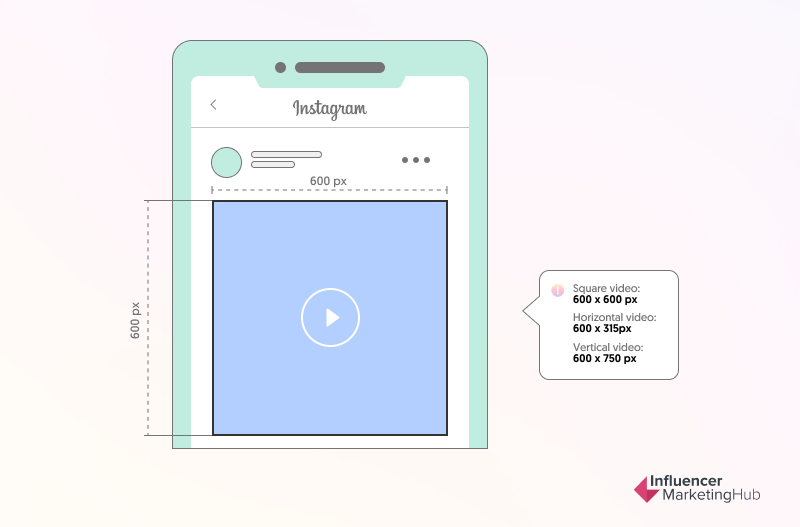
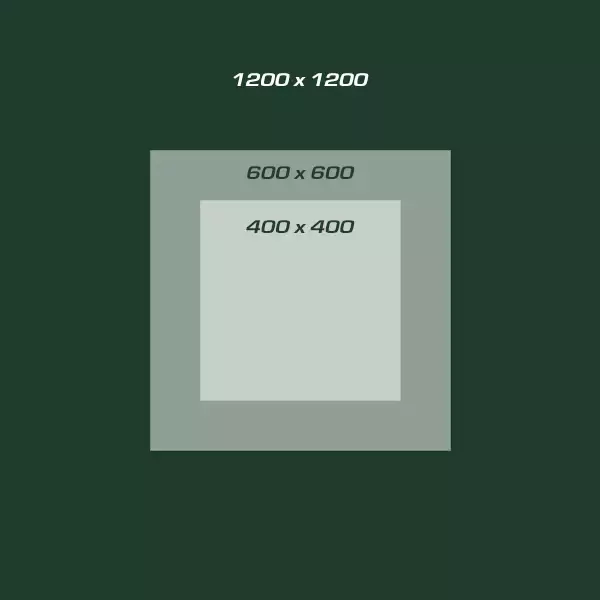
This is an integer variable which will store the score of player. Its value will increase by one when each time the player will successfully click on the coin(means he has collected that coin).

**6. width = 600 # width of the surface of pygame**

This integer variable will define the number of pixels along X-axis of our game i.e. how much wide it will be on the screen of our computer.

**7. height = 600 # height of the surface of pygame**

This integer variable will define the number of pixels along Y-axis of our game i.e. how much height it will have on the screen of our computer.



This image will give us the idea of height and width defined in pixels.

8**. coin\_width = 30**

So whenever we insert any image into pygame window, this image will have height and width of certain pixels (e.g. 250x300 pixels, etc.). There may be a chance that the image we are inserting into the game window will be larger than the game window itself. For e.g. in this game coin image is of the size 2048x2048 pixels which is way larger than the game window itself which is 600x600 pixels. So this coin will appear very big and will go even out of the game window. So to avoid this problem we need to resize the coin height and width. And this ‘coin\_width’ variable will resize our coin to be 30 pixels in width inside the game window.

**9. coin\_height = 30**

This variable will resize the coin’s height to 30 pixels

**10. window = pygame.display.set\_mode((width, height))**

pygame.display.set\_mode() is a method which will actually create a window of pygame of a given size defined by the user. Now the user will be able to create an empty window of pygame of size 600\*600. But this window will still be empty as we have not put any text or image onto it. It will be like an empty canvas of size 600x600 in which we can put or draw anything by coding it.

An important concept which needs to be conveyed to students is that this method takes the size of window in form of a tuple. Like this ‘(width, height)’ in which first one will be the width always. This way of defining size in pygame should be kept in mind as in future also we will be defining sizes of objects in this way.

**11. pygame.display.set\_caption("Clicker Game")**

It will set the title of the window of pygame. In this case “Clicker Game”.

**12. fps = 60 # frames per second**

Frames per second will tell pygame that how many times it should paint the canvas with the objects/text/images that we have put into the pygame window.

**13.** **bg = pygame.image.load(r'./GitHub/Python/ClickerGame/bg1.png')**

This line will insert the image into Python.

An important point here is that ‘r’ before the image address means that the string will be treated as raw string i.e. a character following a backslash is included in the string without change (so that python will not read it as /t means tab space or /n as new line). And user can add different types of images like jpg, jpeg, tiff, png, gif, etc.

Another point here to note is that we need to give full address of the image in load method. A good practice is that we should keep all file of a project in one single folder.

**14. background = pygame.transform.scale(bg, (width, height))**

Now, this is the way we resize any image according to our needs in pygame. Here we are resizing our background image of size 1920x1080 to the size of our pygame window i.e. 600x600.

This ‘pygame.transform.scale()’ method takes two arguments – one is image and other is size in form of tuple.

Notice the way we are passing the size, which is as tuple.

**15. score\_font = pygame.font.SysFont('comicsans', 40)**

By this way we define the type and size of font we want for the result text we will display in the pygame window.

**16. coin\_image = pygame.image.load(r'./coin.png')**

**17. coin = pygame.transform.scale(coin\_image,(coin\_width, coin\_height))**

These two lines will work as same line 13 & 14. These lines will insert the coin image into pygame and resize it to 30x30 pixels. We can change the size of coin to a bigger one or smaller one we feel. We have kept the size of coin small because we do not want it to cover whole area of pygame window. And we have also kept in mind that it should not be too small also that it may become unclickable.

**18. coin\_space = pygame.Rect(random.randint(50,550), 20, coin\_width, coin\_height)**

pygame.rect() create a rectangular in pygame window. This rectangle will hold the coin image. As the image is not familiar to pygame (pygame has not created the image, rather we have given it to pygame), thus it is unknown to pygame. And hence, if we click on this coin image pygame will not know that it is clicked or not. So, pygame will not be able to tell us that this coin has been clicked and we can now increase the score.

But, pygame knows the rectangle it has created by pygane.rect() method. So when we click this rectangle pygame can tell us that this rectangle has been clicked. This rectangle is not visible on the screen because we have not given border to it.

So, what this line will do is that it will create a new rectangle of size of coin. And later we will attach and put our coin in this rectangle. So, whenever user clicks this rectangle we will know that user has clicked on the coin and we can increase the score.

This line will be the most important line as it is this line which will spawn a new coin every time from the top of the screen. So, in future whenever this line is called it will create a new coin on the pygame window.

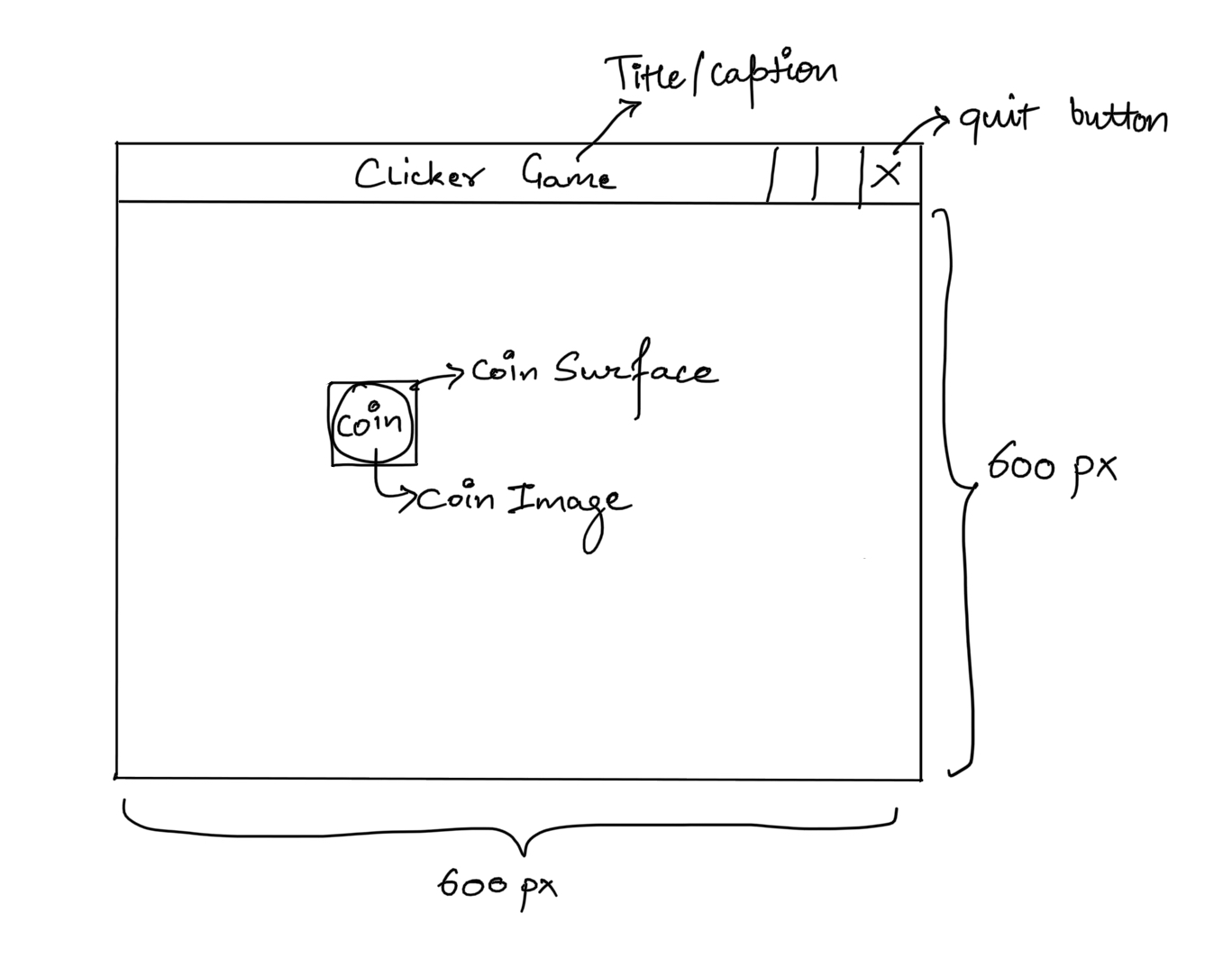
Pygame.rect() method takes 4 arguments:

1. X-position 2. Y-position 3. Width 4. Height

Here we have used “random.randint(50,550)” as X-position. This will give us a random X-position of the pixel i.e. from 50th pixel to the 550th, leaving 50 pixel in the left of the window and 50 pixel from the right side of the window. We have done this so that our coin does not get spawned out of the window. And random method helps to spawn the coin at a random position each time it is spawned.

We have given Y-position as from 20th pixel from the top. As we will display our score in those 20 pixel at the top of the window and coin may not overlap the score.

Width & Height are given are that of the coin. So that we can easily accommodate our coin inside it.



**19. def updateCoin():**

**20. window.blit(coin, (coin\_space.x, coin\_space.y))**

**21. if coin\_space.y <= 500:**

**22. coin\_space.y += 2**

Here we have defined a method named updateCoin() which will update the position of the coin inside the pygame window. This method is responsible for giving the effect of coin falling down from the top.

blit() method of ‘window’ finally puts the coin image in the rectangle created by pygame.rect i.e. coin\_space. blit() method helps in this regard that it will take that rectangular surface of coin and put it on top of the screen.

2 parameters of blit() tells it to take the coin image and put it in the coordinates of ‘coin\_space’ rectangle. Here also x & y position is passed inside a tuple.

Then ‘if’ condition checks that if the coin’s position (from here whenever we say coin we mean the rectangle that holds the coin) is before 500th pixel along the y-axis i.e. if the coin has not reached the bottom part of the screen then put the coin 2 pixels below the current position. When this method is called in a loop it gives the effect that coin is falling down.

If we want to control the speed of coin we can change the value we are adding to ‘coin\_space.y’ each time i.e. lower value will slow it down and vice-versa.

Here for now we are adding 2 pixels to coin\_space.y to move the coin 2 pixels down each time.

**23. clock = pygame.time.Clock()**

This will help to calculate the time inside our game. We will calculate time to control how many times per second we want to paint our pygame window with the objects. Afterall, moving objects are just several still images produced very fast.

**24. run = True**

**25. while run:**

These 2 lines will start an infinite loop until the value of ‘run’ becomes ‘false’. This loop is the main functioning of this game. It will handle the job of updating the pygame window infinitely until user quits the game

**26. clock.tick((fps)) # to control the speed of while loop**

Tick is just a measure of time in PyGame. As the speed of execution of while loop in our CPU is very fast i.e. it can run thousands of times within a second but according to our code line no. 22 tells our coin to move only 2 pixels at a time. This means that our while loop is way too fast and the coin falling from top of the window will pass so fast that we will not even be able to see it. Thus, to control our while loop we tell pygame to tick(i.e. count/wait) till 1/60 of a second before continuing the loop. This way it slows the ‘while’ loop down.

So, in brief, we could say the loop is controlling your game display and clock.tick() specifies how fast you want to change the game display in other words how fast the loop runs. In our case loop will run at most 60 times per second (fps value). In other words it will allow to paint pygame window 60 times in a second.

It also takes a tuple as parameter which in this case contains only ‘fps’(frames per second).

**27. window.blit(background, (0, 0))**

This blit method will put the background image in the top left corner of the pygame window i.e. x=0 & y=0.

And because we have set ‘fps=60’ this blit() line will paint the background 60 times in a second.

**28. for event in pygame.event.get():**

This for loop will have two main functions by capturing 2 evnts:

1. To get out of the ‘while’ loop when user quits the game.

2. To check if player has clicked the coin or not.

An ‘event’ is an action which is done by the user. For e.g. clicking a button or pressing a key, etc.

This for loop will run till user does some action and provides an event to pygame which pygame senses through ‘pygame.event.get()’ method.

In our case it will run when user clicks either on the “QUIT” button of game window or on the ‘coin’.

**29. if event.type == pygame.QUIT:**

**30. run = False**

This ‘if’ condition will check if the user has clicked on “QUIT” button of pygame window i.e. if type of event triggered is ‘pygame.QUIT’ or not.

And if user clicked on Quit button the value of run is changed to ‘false’ and line 25 ‘while’ loop condition becomes false and the loop terminates.

**31. if event.type == pygame.MOUSEBUTTONDOWN:**

**32. x, y = event.pos**

**33. if coin\_space.collidepoint(x, y):**

**34. score += 1**

**35. coin\_space = pygame.Rect(random.randint(50,550),**

**20, coin\_width, coin\_height)**

This ‘if’ condition will check if the user has clicked on the coin or not. And if the coin is clicked we will increase the score by 1 point and spawn a new coin at a random place along the X-axis at the top of the window and the last coin which was clicked will disappear.

**36. if coin\_space.y <= 500:**

**37. updateCoin()**

**38. else:**

**39. coin\_space = pygame.Rect(random.randint(50,550), 20,**

**coin\_width, coin\_height)**

This ‘if’ condition will check whether if coin has reached the 500th pixel i.e. 100 pixel above the bottom of the screen. We can also compare it with 600th pixel, then it will just pass through the bottom of the screen. We can change the number in the ‘if’ condition as it suits us. So, until the coin reaches the bottom of the screen this ‘if’ condition will be always true and continues update the position of coin by calling updateCoin() method.

Else part of this ‘if’ condition means that coin has reached the bottom of the pygame window and user has missed to click on the screen. Then we will just spawn a new coin at the top of the window at a random position on x-axis.

**40. draw\_text = score\_font.render("Score: " + str(score),**

**1,'black')**

Once the font is created, its size cannot be changed. A Font object is used to create a Surface object from a string. Pygame does not provide a direct way to write text onto a Surface object. The method render() must be used to create a Surface object from the text, which then can be blit to the screen. The method render() can only render single lines. A newline character is not rendered.

In simple words this line converts the text “Score” into a surface that can be putted on the pygame window.

We have passed 1st parameter as string we want to display, 2nd parameter as 0 or 1 (1 means text will be smoother and 0 means text will look a little bit rough on its edges), and 3rd parameter as color of the text.

**41. window.blit(draw\_text, (5, 5))**

This statement will put the text on pygame window at position x=5 & y=5 from top-left of the pygame window.

**42. pygame.display.update()**

Without this method we will see no changes at all on the pygame window screen. This is the method responsible for updating all the changes in the game screen. This method will update the screen 60 times in a second as it is still inside the while loop. It will update everything which is blit(putted) on the pygame window like background, coin, score, etc. and that too at 60 fps(60 times in a second).

Ask students to check what happens if we do not update background each time and leave it after putting it just once on the screen i.e. out of the ‘while’ loop. Answer- Last image of coin will not disappear and new coin will overlap it just 2 pixels below it and hence it will seem like the coin is dragging along the screen and will look like a line of coins.

**43. pygame.quit()**

This statement is the very last statement of our program and is out of the loop. This means this statement will run only when loop breaks (i.e. user clicks on ‘QUIT’).

pygame.quit() tells the python to close the pygame which was initialised by pygame.init() method. And the game will be closed.

----------- xxxxxxx -----------

Number of lectures required to complete this code in class is at-least 4 to 5 lectures which should be as follows:

1st lecture: Familiarise students with pygame and build a simple pygame

window and try out different background colors.

2nd lecture: inserting a background image and setting its size.

3rd lecture: inserting coins into the game and making them move.

4th lecture: learning how to register an event and checking event of

coin clicked.

5th lecture: adding score to the game and spawning a continuous stream

of coins