Practical 1

Aim: Setup DirectX 11, Window Framework and Initialize Direct3D Device, Loading models into DirectX 11 and rendering.

Steps:

Download DirectX SDK from the link given in the description.

https://www.microsoft.com/en-in/downl...

Open control panel:

If you have Windows 64 bit then uninstall these 2 files first.(Since mine is 64bit..)

► Microsoft Visual C++ 2010 x64 Redistributable

► Microsoft Visual C++ 2010 x86 Redistributable

If you have Windows 32 bit then uninstall this file only.

► Microsoft Visual C++ 2010 x86 Redistributable

If you dont Follow the above steps you will get an "ERROR NO:S103"

Install DirectX SDK wait till it gets installed.

How to check whether it is installed or not ?

Go to Local Disk C:

Windows folder.

Microsoft.Net

DirectX Mangaged code....

If the above folder is present then you have successfully installed

DirectX SDK :)

Open visual studio.

File

► New project

► Select Visual C#

► Select .NET Framework 2.0

► Select Windows form Application

► Provide the name

► Then click ok.

Right click on your form and click properties

and edit the Title and Size..

Right click on "References" in Solution explorer.

►Add references

►Browse to the "DirectX Managed Code" directory.

►Select these three files.

► Microsoft.DirectX.Direct3DX

► Microsoft.DirectX.Direct3D.dll

► Microsoft.DirectX.dll

and add...

Right click on the Form1 and click View Code.

Then use the below namespace

using Microsoft.DirectX.Direct3D;

....

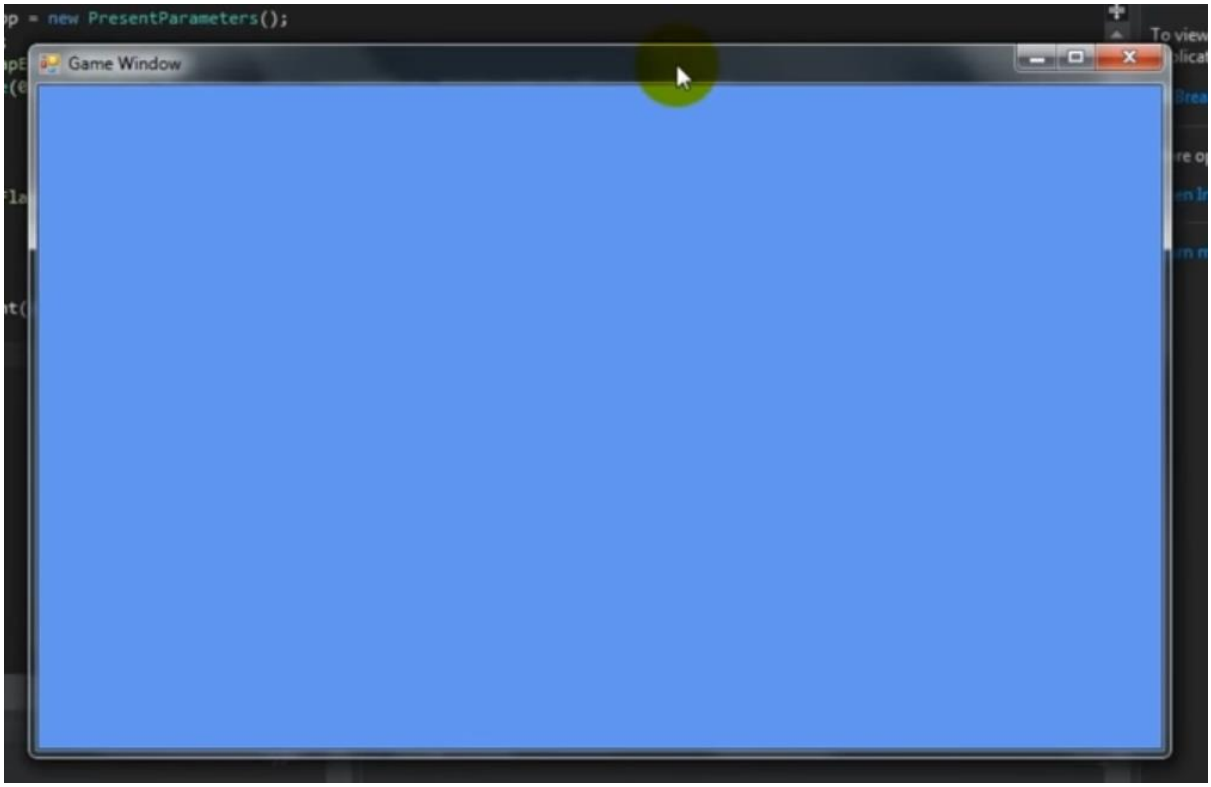
Follow me to add one more method...

Click on BUILD menu then Configuration manager.

In front of your project select Platform ► New ► Select x86 and then close

Then debug...

The O/P will be a window having a light blue background.



Practical 2

Aim: Learn Basic Game Designing Techniques with pygame

Pygame Introduction

Pygame is a cross-platform set of Python modules which is used to create video games. It consists of computer graphics and sound libraries designed to be used with the Python programming language.

Pygame Installation

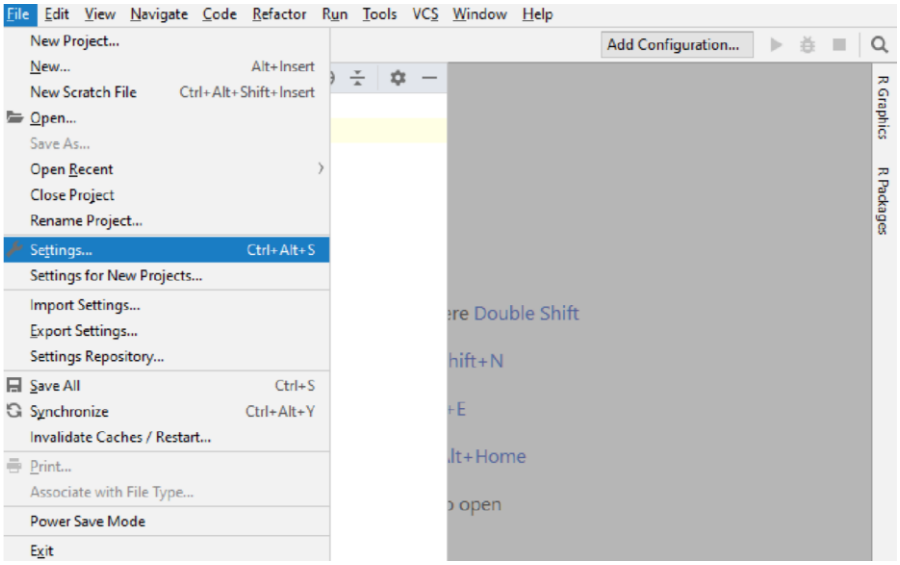
Before installing Pygame, Python should be installed in the system, and it is good to have 3.6.1 or above version because it is much friendlier to beginners, and additionally runs faster.

There are mainly two ways to install Pygame, which are given below:

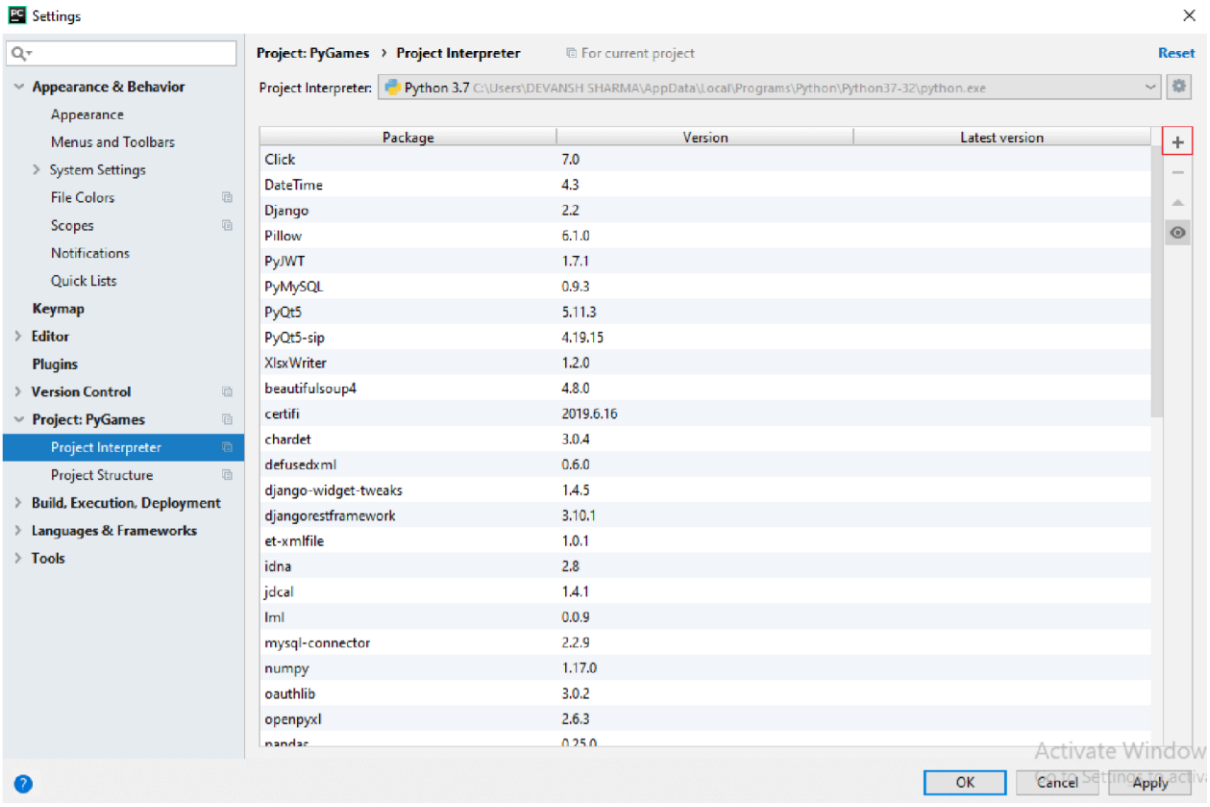
1. **Installing through pip:** The good way to install Pygame is with the pip tool (which is what python uses to install packages). The command is the following:

```
py -m pip install -U pygame --user
```

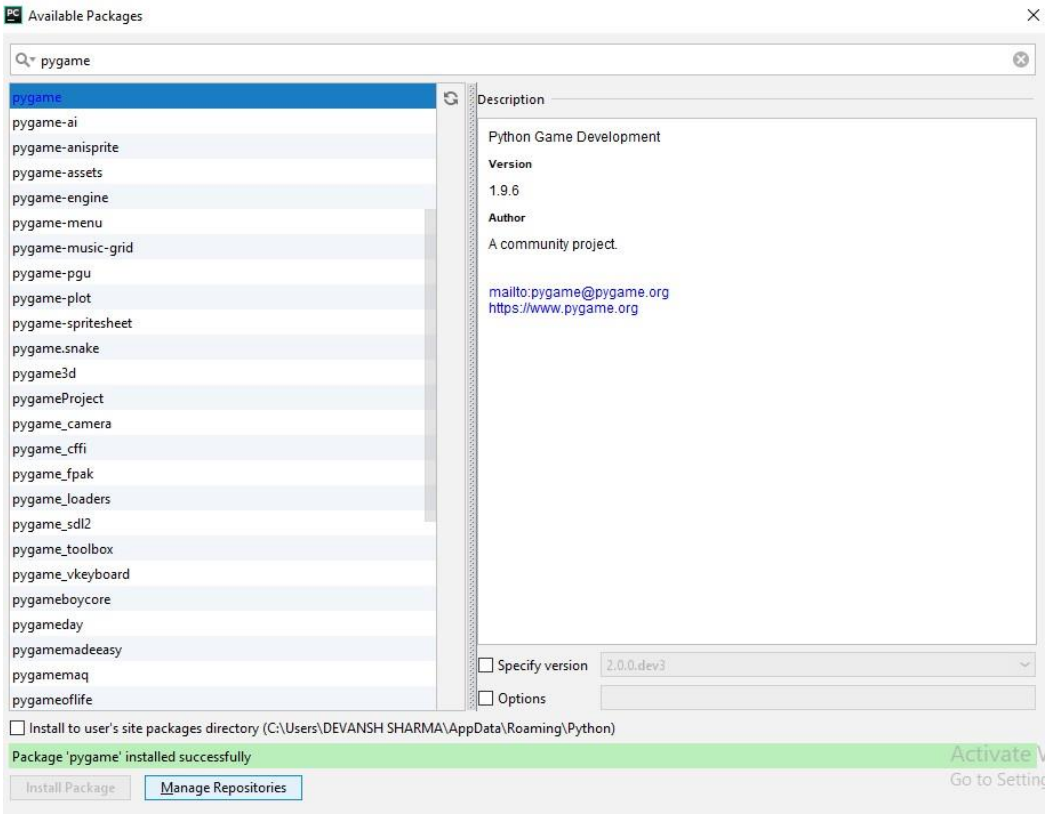
2. **Installing through an IDE:** The second way is to install it through an IDE and here we are using Pycharm IDE. Installation of pygame in the pycharm is straightforward. We can install it by running the above command in the terminal or use the following steps: o Open the **File** tab and click on the **Settings** option.



Select the **Project Interpreter** and click on the + icon.



It will display the search box. Search the pygame and click on the **install package** button.



Note: ***You can use Google Colab or any other jupyter notebook for performing practical.***

Verifying whether pygame is correctly installed or not

To check whether the pygame is properly installed or not, in the IDLE interpreter, type the following command and press Enter:
import pygame

If the command runs successfully without throwing any errors, it means we have successfully installed Pygame and found the correct version of IDLE to use for pygame programming.

Pygame Surface

- The pygame Surface is used to display any image.
- The Surface has a pre-defined resolution and pixel format.
- The Surface color is by default black.
- Its size is defined by passing the **size** argument.

- Surfaces can have the number of extra attributes like **alpha planes, color keys, source rectangle clipping, etc**.
- The blit routines will attempt to use hardware acceleration when possible; otherwise, they will use highly enhanced software blitting methods.

Pygame Clock

- Times are represented in millisecond (1/1000 seconds) in pygame.
- Pygame clock is used to track the time.
- The time is essential to create motion, play a sound, or, react to any event.
- In general, we don't count time in seconds. We count it in milliseconds.
- The clock also provides various functions to help in controlling the game's frame rate. The few functions are the following:

tick()

This function is used to update the clock. The syntax is the following:

tick(framerate=0)

- This method should be called once per frame.
- It will calculate how many milliseconds have passed since the previous call.
- The **framerate** argument is optional to pass in the function, and if it is passed as an argument then the function will delay to keep the game running slower than the given ticks per second.

tick_busy_loop() • The tick_busy_loop() is same as the tick().

- By calling the **Clock.tick_busy_loop(20)** once per frame, the program will never run at more than 20 frames per second.
- The syntax is the following: tick_busy_loop()

get_time()

- The get_time() is used to get the previous tick.
- The number of a millisecond that isdra passed between the last two calls in Clock.tick().

get_time()

Pygame Blit

- The pygame blit is the process to render the game object onto the surface, and this process is called **blitting**.
- When we create the game object, we need to render it.
- If we don't render the game objects and run the program, then it will give the black window as an output.
- Blitting is one of the slowest operations in any game so, we need to be careful to not to blit much onto the screen in every frame.
- The primary function used in blitting is blit(), which is:

blit()

```
blit(source,dest,area=None,special_flags=0)
```

- This function is used to draw one image into another.
- The draw can be placed with the dest argument.
- The dest argument can either be a pair of coordinates representing the upper left corner of the source.

Pygame Adding Image

- To add an image on the window, first, we need to instantiate a blank surface by calling the Surface constructor with a width and height tuple.

```
surface = pygame.Surface((100,100))
```

- The above line creates a blank 24-bit RGB image that's 100*100 pixels with the default black color.
- For the transparent initialization of Surface, pass the SRCALPHA argument.

```
surface = pygame.Surface((100,100), pygame.SRCALPHA)
```

Program to display an Image

Consider the following example to display image on the surface:

```
1. import pygame
2. pygame.init()
3. white = (255, 255, 255)
4. # assigning values to height and width variable
5. height = 400
6. width = 400
7. # creating the display surface object
8. # of specific dimension..e(X, Y).
9. display_surface = pygame.display.set_mode((height, width))
10.
11. # set the pygame window
name    12.
pygame.display.set_caption('Ima
ge')
13.
14. # creating a surface object, image is drawn on it.
15. image = pygame.image.load('C:\Users\
    \Desktop\download.png')
16.
17. # infinite loop
```

```
18. while True:
19. display_surface.fill(white)   20.    display_surface.blit(image, (0,
    0))
21.
22.         for event in pygame.event.get():
23.         if event.type == pygame.QUIT:
24.         pygame.quit()
25.         # quit the program.
26.         quit()
27.         # Draws the surface object to the screen.
28.         pygame.display.update()
```

Pygame Rect

- Rect is used to draw a rectangle in Pygame.
- Pygame uses Rect objects to store and manipulate rectangular areas.
- A Rect can be formed from a combination of left, top, width, and height values.
- It can also be created from Python objects that are already a Rect or have an attribute named "rect".
- The **rect()** function is used to perform changes in the position or size of a rectangle.
- It returns the new copy of the Rect with the affected changes.
- No modification happens in the original rectangle.
- The Rect object has various virtual attributes which can be used to move and align the Rect:

x,y top, left, right, bottom topleft,

bottomleft, topright, bottomright

midtop, midleft, midbottom,

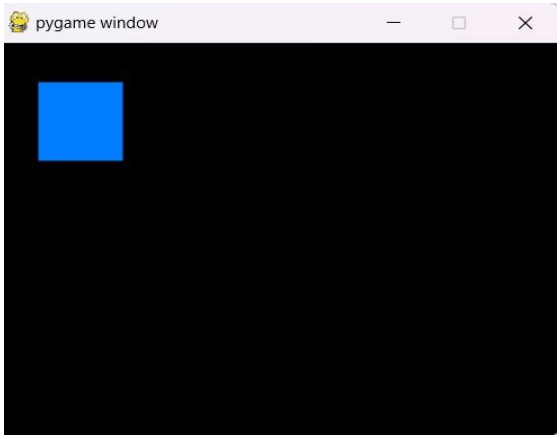
midright center, centerx, centery

size, width, height w,h

- The dimension of the rectangle can be changed by assigning the size, width, or height.
- All other assignment moves the rectangle without resizing it.
- If the width or height is a non-zero value of Rect, then it will return True for a non-zero test.
- Some methods return a Rect with 0 sizes to represent an invalid rectangle.

Program to create a rectangle

```
1. import pygame
2.
3. pygame.init()
4. screen = pygame.display.set_mode((400, 300))
5. done = False
6.
7.         while not done:
8.         for event in pygame.event.get():
9.         if event.type == pygame.QUIT:
10.        done = True
11.        pygame.draw.rect(screen, (0, 125, 255), pygame.Rect(30, 30, 60, 60))
12.
13.    pygame.display.flip()
```



Pygame Keydown

Pygame KEYDOWN and KEYUP detect the event if a key is physically

pressed and released. **KEYDOWN** detects the key press and, **KEYUP** detects the key release. Both events (Key press and Key release) have two attributes which are the following: o **key:** Key is an integer id which represents every key on the keyword. o **mod:** This is a bitmask of all the modifier keys that were in the pressed state when the event occurred.

Consider the following example of the key press and key release.

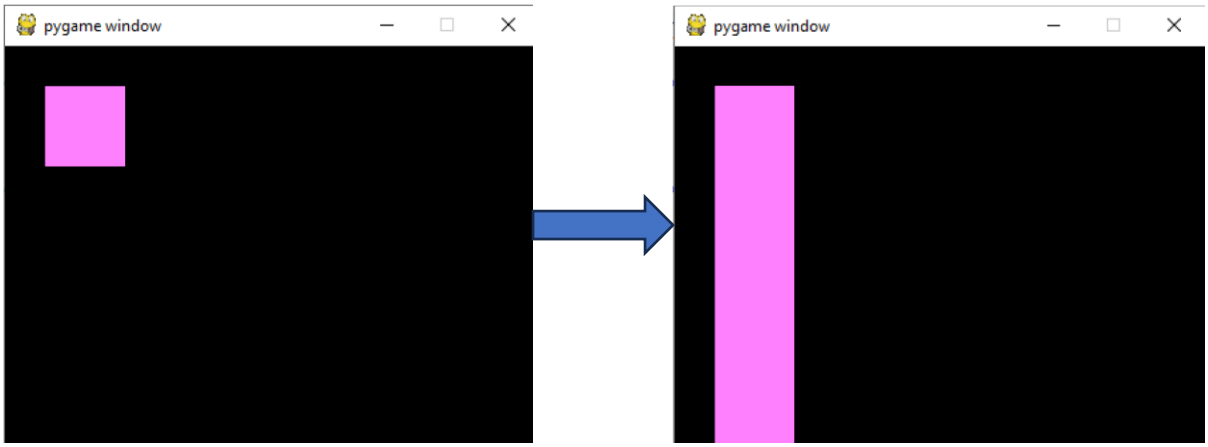
```
import pygame
pygame.init()
# sets the window title
pygame.display.set_caption(u'Keyboard events')
# sets the window size
pygame.display.set_mode((400, 400))

    while True:
    # gets a single event from the event queue
    event = pygame.event.wait()
    # if the 'close' button of the window is pressed
    if event.type == pygame.QUIT:
    # stops the application
    break
    # Detects the 'KEYDOWN' and 'KEYUP' events
    if event.type in (pygame.KEYDOWN, pygame.KEYUP):
    # gets the key name
    key_name = pygame.key.name(event.key)
    # converts to uppercase the key name
    key_name = key_name.upper()
    # if any key is pressed
```

```
22.     if event.type == pygame.KEYDOWN:
23.     # prints on the console the key pressed
24.     print(u'"{}" key pressed'.format(key_name))
25.     # if any key is released
26.     elif event.type == pygame.KEYUP:
27.     # prints on the console the released key
28.     print(u'"{}" key released'.format(key_name))
```

Output :

In the above code, the rectangle will be displayed on the pygame window.

When we press the Down key, the rectangle is reshaped in the downwards.



Pygame Draw

- Pygame provides geometry functions to draw simple shapes to the surface.
- These functions will work for rendering to any format to surfaces.
- Most of the functions accept a width argument to signify the size of the thickness around the edge of the shape.
- If the width is passed 0, then the shape will be solid(filled).

- All the drawing function takes the color argument that can be one of the following formats: o A pygame.Color objects o An (RGB) triplet(tuple/list) o An (RGBA) quadruplet(tuple/list)

  o An integer value that has been mapped to the surface's pixel format

Draw a rectangle

The following functions are used to draw a rectangle on the given surface.

1. pygame.draw.rect(surface, color, rect)
2. pygame.draw.rect(surface, color, rect, width=0)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **rect(Rect)-** Draw rectangle, position, and dimensions. o **width(int)-** This is optional to use the line thickness or to indicate that the rectangle is filled.

1. **if** width == 0, (**default**) fill the rectangle
2. **if** width > 0, used **for** line thickness
3. **if** width < 0, nothing will be drawn

Draw a polygon

The following functions are used to draw a polygon on the given surface. o pygame.draw.polygon(surface,color,points)

- pygame.draw.polygon(surface, color, points, width=0)

  Parameters:

- **surface -** Screen to draw on. o **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.

- **points(tuple(coordinate) or list(coordinate)):** A sequence of 3 or more (x,y) coordinates that make up the vertices of the polygon. Each coordinate in the sequence must be tuple/list.

Draw an ellipse

The following functions are used to draw an ellipse on the given surface.

1. pygame.draw.ellipse(surface, color, rect)
2. pygame.draw.ellipse(surface, color, rect, width=0)

Parameters:

- **surface -** Screen to draw on. o **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.

- **rect(Rect)-** Draw rectangle, position, and dimensions.

Draw a straight line

This method is used to draw a straight line on the given surface. There are no endcaps.

1. pygame.draw.line(surface,color,start_pos,end_pos,width)
2. pygame.draw.line(surface,color,start_pos,end_pos,width=

1) **Parameters:**

- **surface -** Screen to draw on. o **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple. o **start_pos-** start position of the line(x,y) o **end_pos-** End position of the line

Draw a Circle

Below are the functions, which are used to draw a circle on the given surface. o circle(surface, color, center, radius)

- circle(surface, color, center, radius, width=0)

Parameters:

- **surface -** Screen to draw on. o **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **center -** The center point of the circle as a sequence of two int/float, e.g. (x,y)
- **radius(int or float)-** radius of the circle, measured from the center parameter, if the radius is zero, then it will only draw the center pixel.

Draw an elliptical arc

Below functions are used to draw an elliptical arc on the given surface.

1. ? arc(surface, color, rect, start_angle, stop_angle)
2. ? arc(surface, color, rect, start_angle, stop_angle, width=1) **Parameters:**

- **surface -** Screen to draw on.

- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **rect(Rect)-** Draw rectangle, position, and dimensions. o

  start_angle- Start angle of the arc in radians. o **stop_angle-**

  Stop angle of the arc in radians.

There are three conditions for start_angle and stop_angle parameter:

a. If start_angle < stop_angle then the arc will be drawn in a counter-clock direction from the start_angle to end_angle.

b. If start_angle>stop_angle then tau(tau=2*pi) will be added to the stop angle.

c. If start_angle==stop_angle, nothing will be drawn.

Pygame Text and Font • Pygame also provides facilities

to render the font and text.

- We can load fonts from the system by using the **pygame.font.SysFont()** function.
- Pygame comes with the built-in default font which can be accessed by passing the font name or None.
- There are many functions to help to work with the font.
- The font objects are created with **pygame.font.Font()**.
- The actual font objects do most of the works done with fonts.
- Font objects are generally used to render the text into new Surface objects.
- Few important font functions are the following: **render()**

- This function is used to draw text on a new Surface.
- Pygame has no facility to draw text on the existing Surface.

- This creates a new Surface with the specified text render on it.
- The syntax is the following: render(text, antialias, color, background=None)

size()

- This function is used to determine the number of space or positioning needed to render text.
- It can also be used for word-wrapping and other layout effects.
- The syntax is the following: size(bool)

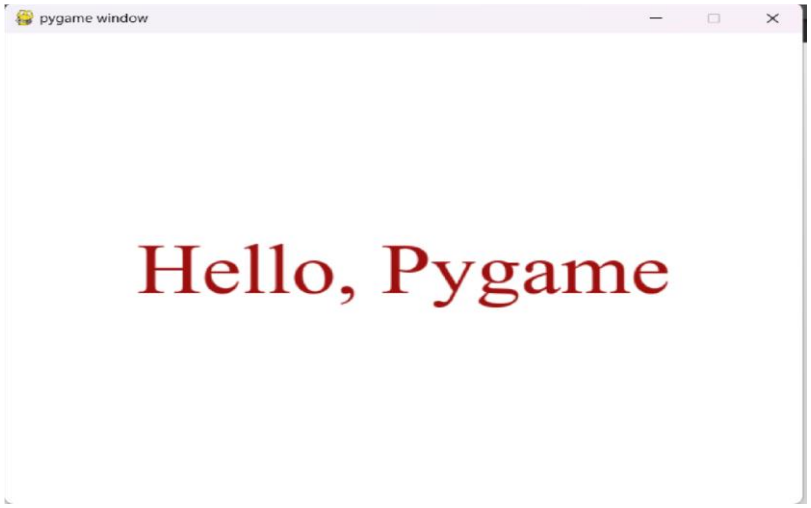
set_bold()

This function is used for bold rending of text. The syntax is following:

set_bold(bool)

```
1. import pygame
2. pygame.init()
3. screen = pygame.display.set_mode((640, 480))
4. done = False
5.
    6. #load the fonts
    7. font = pygame.font.SysFont("Times new Roman", 72)
    8. # Render the text in new surface
    9. text = font.render("Hello, Pygame", True, (158, 16, 16))
    10. while not done:
    11. for event in pygame.event.get():
    12. if event.type == pygame.QUIT:
    13. done = True
    14. if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE
```

```
:
        done = True
        screen.fill((255, 255, 255))
        #We will discuss blit() in the next topic
        screen.blit(text,(320 - text.get_width() // 2, 240 -
            text.get_height() // 2))
    pygame.display.flip()
```



Pygame Sprite and Collision detection

A pygame sprite is a two-dimensional image that is part of the large graphical scene. Usually, a sprite will be some object in the scene.

One of the most advantages of working with sprites is the ability to work with them in groups. We can easily move and draw all the sprites with the one command if they are in the group.

The Sprite module contains the various simple classes to be used within the games. It is optional to use Sprite classes and different group classes when using pygame. Pygame provides sprites and sprite groups that help for collision detection. Collision detection is the process when two objects on the screen collide each other. For example, if a player is hit by the enemy's bullet, then it may lose a life or, the program need to know when the player touches a coin so that they automatically picked up.

Practical 3

Aim: Develop Snake Game using pygame

1. Snake game is one of the most popular arcade games of all time.
2. In this game, the main objective of the player is to catch the maximum number of fruits without hitting the wall or itself.
3. It is one of the best beginner-friendly projects that every novice programmer should take as a challenge.

Prerequisite: Python must be installed on your Computer with Pygame library.

Installation:

- To install Pygame, you need to open up your terminal or command prompt and type the following command:

```
pip install pygame
```

- After installing Pygame we are ready to create our cool snake game.

Step 1: Importing the necessary libraries.

- After that, we are defining the width and height of the window in which the game will be played.
- And define the color in RGB format that we are going to use in our game for displaying text.

Step 2: Initialize Pygame using pygame.init() method.

- Create a game window using the width and height defined in the previous step.
- Here pygame.time.Clock() will be used further in the main logic of the game to change the speed of the snake.

Step 3: Initialize snake position and its size.

- After initializing snake position, initialize the fruit position randomly anywhere in the defined height and width.

- By setting direction to RIGHT we ensure that, whenever a user runs the program/game, the snake must move right to the screen.

Step 4: Create a function to display the score of the player.

- In this function, firstly we're creating a font object i.e. the font color will go here.
- Then we are using render to create a background surface that we are going to change whenever our score updates.
- Create a rectangular object for the text surface object (where text will be refreshed)
- Then, we are displaying our score using **blit. blit** takes two argument **screen.blit(background,(x,y))**

Step 5: Now create a game over function that will represent the score after the snake is hit by a wall or itself.

- In the first line, we are creating a font object to display scores.
- Then we are creating text surfaces to render scores.
- After that, we are setting the position of the text in the middle of the playable area.
- Display the scores using **blit** and updating the score by updating the surface using flip().
- We are using sleep(2) to wait for 2 seconds before closing the window using quit().

Step 6: Now we will be creating our main function that will do the following things:

- We will be validating the keys that will be responsible for the movement of the snake, then we will be creating a special condition that the snake should not be allowed to move in the opposite direction instantaneously.

- After that, if snake and fruit collide we will be incrementing the score by 10 and new fruit will be spanned.
- After that, we are checking that is the snake hit with a wall or not. If a snake hits a wall we will call game over function.
- If the snake hits itself, the game over function will be called.
- And in the end, we will be displaying the scores using the show_score function created earlier.

Implementation

```
import pygame
import time
import random
snake_speed = 15
window_x = 720
window_y = 480
black = pygame.Color(0, 0, 0)
white = pygame.Color(255, 255, 255)
red = pygame.Color(255, 0, 0)
green = pygame.Color(0, 255, 0)
blue = pygame.Color(0, 0, 255)
pygame.init()
pygame.display.set_caption('GeeksforGeeks Snakes')
game_window = pygame.display.set_mode((window_x, window_y))
fps = pygame.time.Clock()
snake_position = [100, 50]
snake_body = [[100, 50],
              [90, 50],
              [80, 50],
              [70, 50]
              ]
# fruit position
fruit_position = [random.randrange(1, (window_x//10)) * 10,
                  random.randrange(1, (window_y//10)) * 10]
fruit_spawn = True
#right
direction = 'RIGHT'
```

```
change_to = direction
# initial score
score = 0
# displaying Score function
def show_score(choice, color, font, size):
    score_font = pygame.font.SysFont(font, size)
    score_surface = score_font.render('Score : ' + str(score), True,
color)
    score_rect = score_surface.get_rect()
    game_window.blit(score_surface, score_rect)
# game over function
def game_over():
    my_font = pygame.font.SysFont('times new roman', 50)
    game_over_surface = my_font.render(
        'Your Score is : ' + str(score), True, red)
    game_over_rect = game_over_surface.get_rect()
    game_over_rect.midtop = (window_x/2, window_y/4)
    game_window.blit(game_over_surface, game_over_rect)
    pygame.display.flip()
    time.sleep(2)
    pygame.quit()
    quit()
# Main Function
while True:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                change_to = 'UP'
            if event.key == pygame.K_DOWN:
                    change_to = 'DOWN'
            if event.key == pygame.K_LEFT:
                    change_to = 'LEFT'
            if event.key == pygame.K_RIGHT:
                    change_to = 'RIGHT'
    if change_to == 'UP' and direction != 'DOWN':
        direction = 'UP'
    if change_to == 'DOWN' and direction != 'UP':
```

```
        direction = 'DOWN'
    if change_to == 'LEFT' and direction != 'RIGHT':
        direction = 'LEFT'
    if change_to == 'RIGHT' and direction != 'LEFT':
        direction = 'RIGHT'
    if direction == 'UP':
        snake_position[1] -= 10
    if direction == 'DOWN':
        snake_position[1] += 10
    if direction == 'LEFT':
        snake_position[0] -= 10
    if direction == 'RIGHT':
        snake_position[0] += 10
    snake_body.insert(0, list(snake_position))
    if snake_position[0] == fruit_position[0] and snake_position[1] ==
fruit_position[1]:
        score += 10
        fruit_spawn = False
    else:
        snake_body.pop()
    if not fruit_spawn:
        fruit_position = [random.randrange(1, (window_x//10)) * 10,
                          random.randrange(1, (window_y//10)) * 10]
    fruit_spawn = True
    game_window.fill(black)
    for pos in snake_body:
        pygame.draw.rect(game_window, green,
                         pygame.Rect(pos[0], pos[1], 10, 10))
        pygame.draw.rect(game_window, white, pygame.Rect(
            fruit_position[0], fruit_position[1], 10, 10))
# Game Over conditions
    if snake_position[0] < 0 or snake_position[0] > window_x-10:
        game_over()
    if snake_position[1] < 0 or snake_position[1] > window_y-10:
        game_over(
    for block in snake_body[1:]:
```

```
        if snake_position[0] == block[0] and snake_position[1] ==
block[1]:
            game_over()
# displaying score continuously
    show_score(1, white, 'times new roman', 20)
# Refresh game screen
    pygame.display.update()
# Frame Per Second /Refresh Rate
    fps.tick(snake_speed)
```

Practical 4

Aim: Create 2D Target Shooting Game

Step 1: Importing pygame, random, sys, and math:

The very first task we will do in developing the Balloon Shooter Game using Python PyGame is importing all the required libraries and modules for this project.

Step 2: Pygame library initialization:

Now after importing, we will initialize the pygame library. All imported pygame modules are initialized via pygame.init().

Step 3: Declaring the required variables

Now after importing, we will declare all the variables that we will use in the development of this project.

Step 4: Declaring variables that store color codes:

This font variable holds the font that is used to display the number of balloons busted. This is done using the SysFont() function from the font module that takes in the name of the font and the size of the font.

Step 5: Defining balloon class:

Now comes the actual coding for the game. First of all, we will define a class and in that class, we will write the logic for functions that can do the following functions: • A function that moves the balloon

- A function to show the balloon on screen
- To check if the balloon is busted or not:
- Function to reset balloons

Step 6: Show balloon location using pointer() function.

Step 7: Display the score

Step 8: To close the game

Step 9: To keep track of the events

Step 10: Call to game() function

CODE:

```
import pygame
import sys
import random
from math import *
pygame.init()
width = 700
height = 600
display = pygame.display.set_mode((width, height))
pygame.display.set_caption("Balloon Shooter Game")
clock = pygame.time.Clock()
margin = 100
lowerBound = 100
score = 0
white = (230, 230, 230)
lightBlue = (4, 27, 96)
red = (231, 76, 60)
lightGreen = (25, 111, 61)
darkGray = (40, 55, 71)
darkBlue = (64, 178, 239)
green = (35, 155, 86)
```

```
yellow = (244, 208, 63)
blue = (46, 134, 193)
purple = (155, 89, 182)
orange = (243, 156, 18)
font = pygame.font.SysFont("Arial", 25)
class Balloon:
    def __init__(self, speed):
        self.a = random.randint(30, 40)
        self.b = self.a + random.randint(0, 10)
        self.x = random.randrange(margin, width - self.a - margin)
        self.y = height - lowerBound
        self.angle = 90
        self.speed = -speed
        self.proPool= [-1, -1, -1, 0, 0, 0, 0, 1, 1, 1]
        self.length = random.randint(50, 100)
        self.color = random.choice([red, green, purple, orange, yellow,
blue])
    def move(self):
        direct = random.choice(self.proPool)
        if direct == -1:
            self.angle += -10
        elif direct == 0:
            self.angle += 0
        else:
            self.angle += 10
```

```
        self.y += self.speed*sin(radians(self.angle))
        self.x += self.speed*cos(radians(self.angle))
        if (self.x + self.a > width) or (self.x < 0):
            if self.y > height/5:
                self.x -= self.speed*cos(radians(self.angle))
            else:
                self.reset()
        if self.y + self.b < 0 or self.y > height + 30:
            self.reset()
    def show(self):
        pygame.draw.line(display, darkBlue, (self.x + self.a/2, self.y +
self.b), (self.x + self.a/2, self.y + self.b + self.length))
        pygame.draw.ellipse(display, self.color, (self.x, self.y, self.a,
self.b))
        pygame.draw.ellipse(display, self.color, (self.x + self.a/2 - 5,
self.y + self.b - 3, 10, 10)
    def burst(self):
        global score
        pos = pygame.mouse.get_pos()
        if isonBalloon(self.x, self.y, self.a, self.b, pos):
            score += 1
            self.reset()
    def reset(self):
        self.a = random.randint(30, 40)
        self.b = self.a + random.randint(0, 10)
        self.x = random.randrange(margin, width - self.a - margin)
```

```
        self.y = height - lowerBound
        self.angle = 90
        self.speed -= 0.002
        self.proPool = [-1, -1, -1, 0, 0, 0, 0, 1, 1, 1]
        self.length = random.randint(50, 100)
        self.color = random.choice([red, green, purple, orange, yellow,
blue])
balloons = []
noBalloon = 10
for i in range(noBalloon):
    obj = Balloon(random.choice([1, 1, 2, 2, 2, 2, 3, 3, 3, 4]))
    balloons.append(obj)
def isonBalloon(x, y, a, b, pos):
    if (x < pos[0] < x + a) and (y < pos[1] < y + b):
        return True
    else:
        return False
def pointer():
    pos = pygame.mouse.get_pos()
    r = 25
    l = 20
    color = lightGreen
    for i in range(noBalloon):
        if isonBalloon(balloons[i].x, balloons[i].y, balloons[i].a,
balloons[i].b, pos):
```

```
            color = red
    pygame.draw.ellipse(display, color, (pos[0] - r/2, pos[1] - r/2, r, r),
4)
    pygame.draw.line(display, color, (pos[0], pos[1] - l/2), (pos[0],
pos[1] - l), 4)
    pygame.draw.line(display, color, (pos[0] + l/2, pos[1]), (pos[0] + l,
pos[1]), 4)
    pygame.draw.line(display, color, (pos[0], pos[1] + l/2), (pos[0],
pos[1] + l), 4)
    pygame.draw.line(display, color, (pos[0] - l/2, pos[1]), (pos[0] - l,
pos[1]), 4)
def lowerPlatform():
    pygame.draw.rect(display, darkGray, (0, height - lowerBound,
width, lowerBound))
def showScore():
    scoreText = font.render("Balloons Bursted : " + str(score), True,
white)
    display.blit(scoreText, (150, height - lowerBound + 50))
def close():
    pygame.quit()
    sys.exit()
def game():
    global score
    loop = True
    while loop:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
```

```
            close()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                close()
            if event.key == pygame.K_r:
                score = 0
                game()
        if event.type == pygame.MOUSEBUTTONDOWN:
            for i in range(noBalloon):
                balloons[i].burst()
    display.fill(lightBlue)
    for i in range(noBalloon):
        balloons[i].show()
    pointer()
    for i in range(noBalloon):
        balloons[i].move()
    lowerPlatform()
    showScore()
    pygame.display.update()
    clock.tick(60)
game()
```

OUTPUT:

Name: Vipul Jadhav
Roll no: S.21.42

Game Programming Practicals

Practical 5

Aim: Creating 2D Infinite Scrolling Background

Pygame offers many advantages for developing basic infrastructure for many games. One of them is the scrolling background. Many of the popular games of all time required this feature of endless scrolling. This scrolling background helps to make the background more creative with less effort.

In a scrolling background, one image is considered as a background that will repeat itself again and again. Thus creating a scrolling endless loop of images. Suppose in a Pygame shell we move a single image from one coordinate to another, thus shifting the pixel of one image to another. Now, these blank pixels can be filled by the other image.

Required libraries

Run the below command in the command prompt to install the Pygame library. pip install pygame

Step 1. Import the libraries and modules required and initialize the declared module.

```
import pygame as py
import math
```

Step 2. Declaring the clock for managing the speed of the scrolling. FrameHeight, FrameWidth, and Frame window for scrolling the background image in Pygame.

Step 3. Uploading the background image for scrolling and setting the scroll variable to 0. And calculate the number of tiles or area boxes required for an uploaded image in a pygame frame.

Step 4. Call the clock function of pygame for managing the fps of

the window screen. Using the **blit function** (**block transfer**) this function helps us to copy the image from one position to another position. This helps us to append the image to the back of the image. In a condition, if the scroller moves beyond the Frame Width then modify the value of the scroller to 0. Thus it will help us to move the same frame again and again and creates an endless scrolling view. And for closing the pygame window we must have to declare the event controller to get the quit request by the user to end the pygame window.

Code Implementation:

```
import math
import pygame as py
py.init()
clock = py.time.Clock()
FrameHeight = 600
FrameWidth = 1200
py.display.set_caption("Endless
Scrolling in pygame") screen =
py.display.set_mode((FrameWidth,
 FrameHeight))
bg = py.image.load("sea6.png").convert()
scroll = 0
tiles = math.ceil(FrameWidth / bg.get_width())
+ 1
while 1:
clock.tick(33)
 i = 0
 while(i < tiles):
 screen.blit(bg, (bg.get_width()*i
```

```
+ scroll, 0))
i += 1
scroll -= 6
if abs(scroll) > bg.get_width():
scroll = 0
for event in py.event.get():
if event.type == py.QUIT:
quit()
py.display.update()
py.quit()
```

Output:

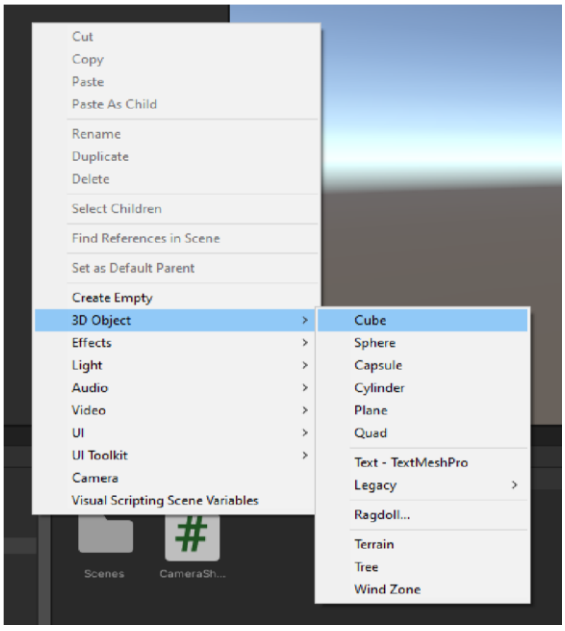
Practical 6

Aim: Create Camera Shake effect in Unity.

Steps:

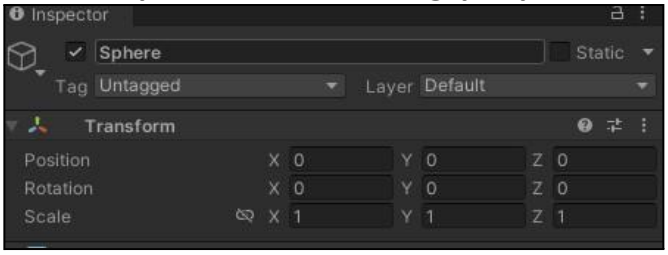
1) Open Unity Hub and Create New Project
2) Select 3D core and name the project
3) Create and scale the objects
   - Right Click on Hierarchy Window. Navigate through 3D Object

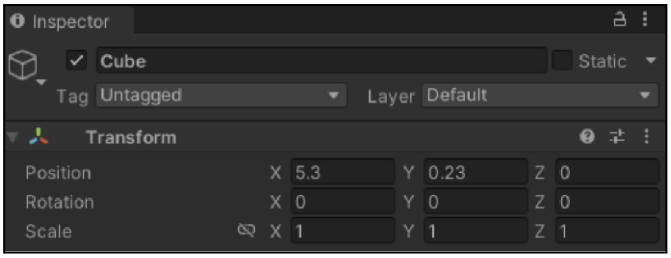
     > Select Spehere & Cube



   - Select your properties one by one and allocate them

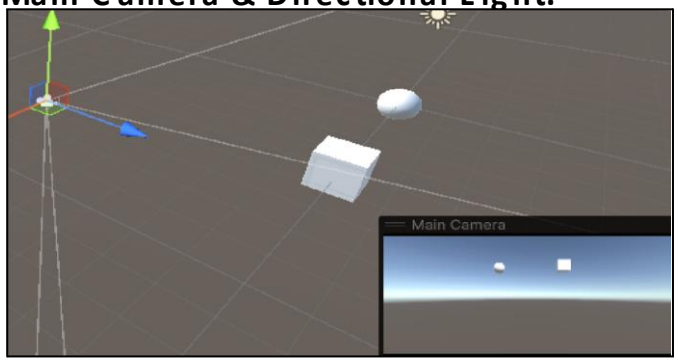
     desired position & scaling properties.





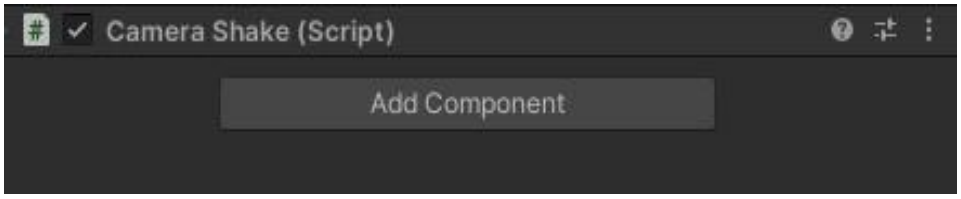
- 3D Object will be displayed in Hierarchy window along with

  Main Camera & Directional Light.

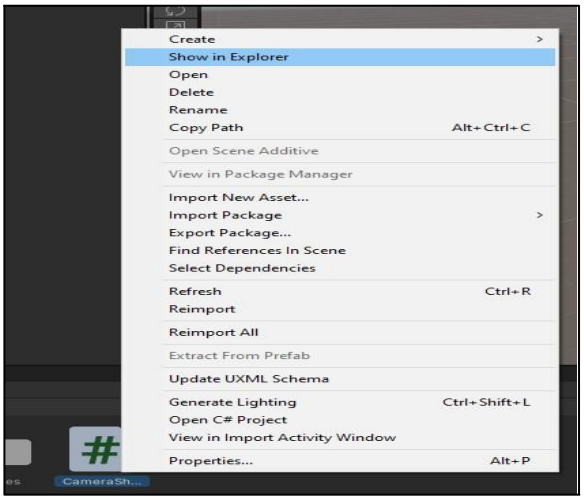


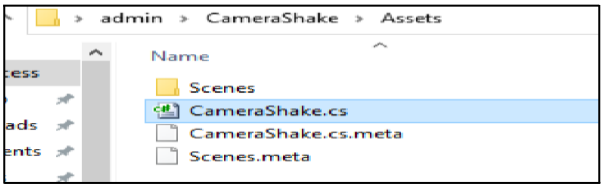
- In the Inspector window, “Add Component” > New Script >

  Name it as “CameraShake”



- In the Asset section, RightClick on “CameraShake”> Show in Explorer > Click on CameraShake.cs > Open in Visual Studio Code





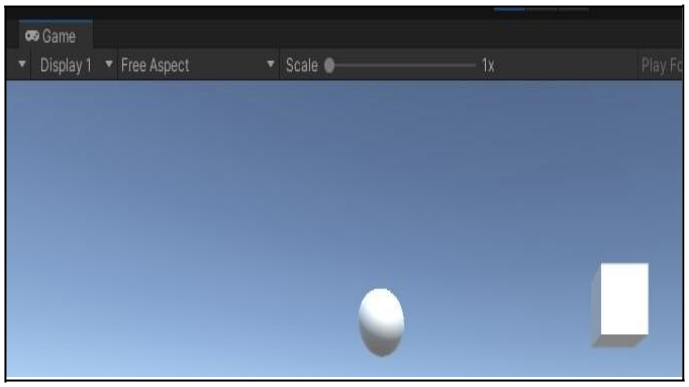
CODE

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraShake : MonoBehaviour
{
  private Transform cameraTransform;
  private Vector3 originalPosition;
  private float shakeDuration = 2.5f;
  private float shakeIntensity = 0.5f; //
  Adjust as needed Start()
  {
cameraTransform = transform; // Assign the camera's transform to the
variable originalPosition = cameraTransform.localPosition;
  }
```

```
        void Update()

        {
Debug.Log("Shake duration: " +
shakeDuration); if (shakeDuration > 0)
{
                    cameraTransform.localPosition = originalPosition +
                                                Random.insideUnitSphere *
    shakeIntensity;
            shakeDuration -=
            Time.deltaTime;
}
else
{
            shakeDuration = 0f;
            cameraTransform.localPosition = originalPosition;
}

if (Input.GetKeyDown(KeyCode.Space)) // This line should be inside
the Update
    method
{
            ShakeCamera(3.5f); // Adjust duration as needed
}
        }

        public void ShakeCamera(float duration)

        {
```

```
Debug.Log("Shake initiated with duration: " + duration);
originalPosition = cameraTransform.localPosition;
shakeDuration = duration;

        }

}
```

Go to game section & play the program



Practical 7

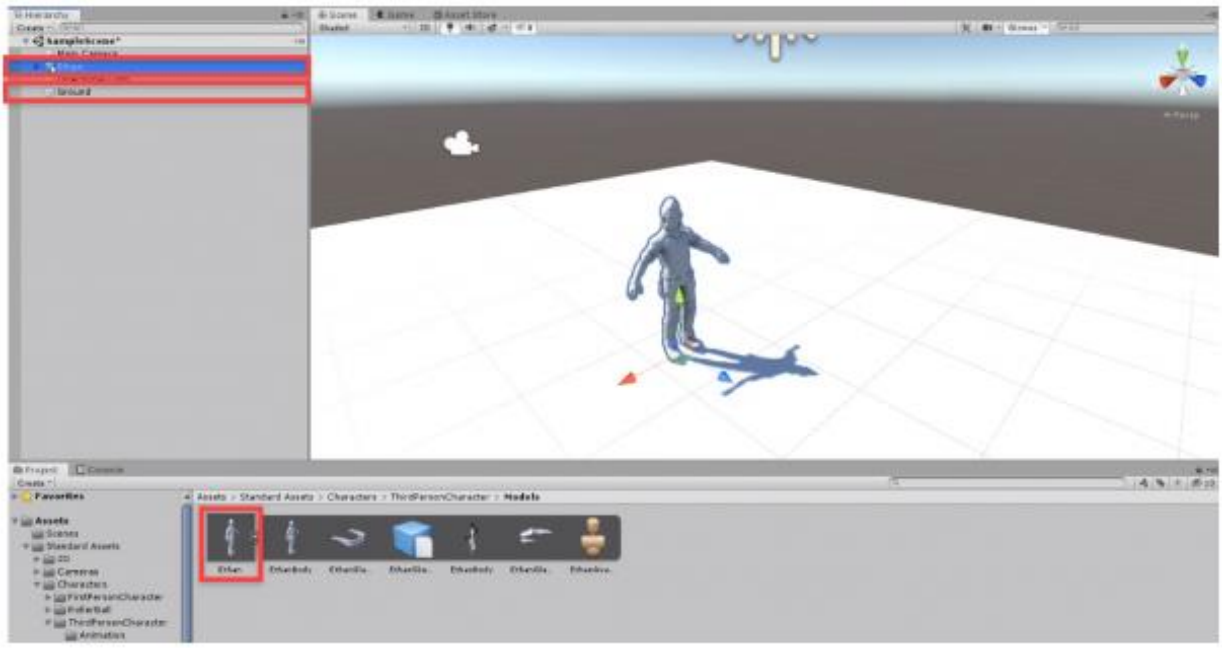
Aim: Design and Animate Game Character in Unity.

STEPS:-

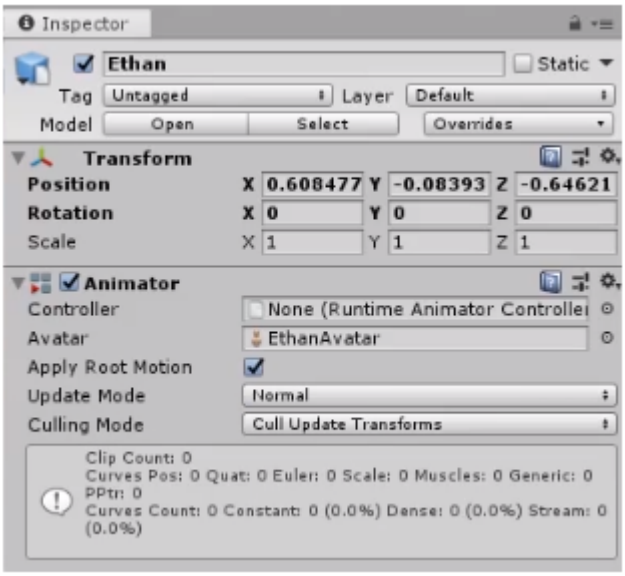
1) Animator Component

In this lesson, we'll be looking at the Animator component in Unity. Let's start by creating a new Unity project. For this project, we're going to be using some of Unity's standard assets. These can be found on the Asset Store (*Window > Asset Store*). Search for *Standard Assets* and import that package from Unity Technologies.

With the standard assets, let's drag in the Ethan model (*Standard Assets > Characters > ThirdPersonCharacter > Models*). Then for a floor, let's create a new cube (right click Hierarchy > *3D Object > Cube*).

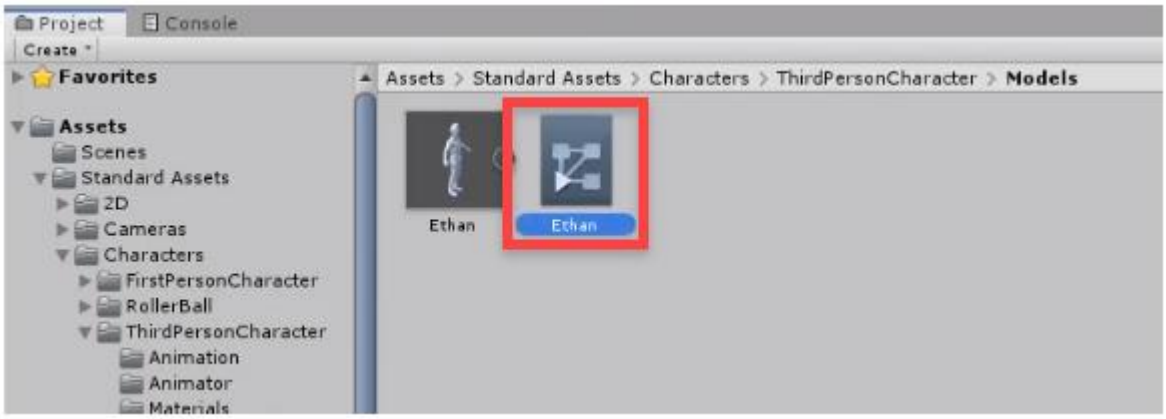


If we select the model in the scene, you'll see that it has an **Animator** component attached

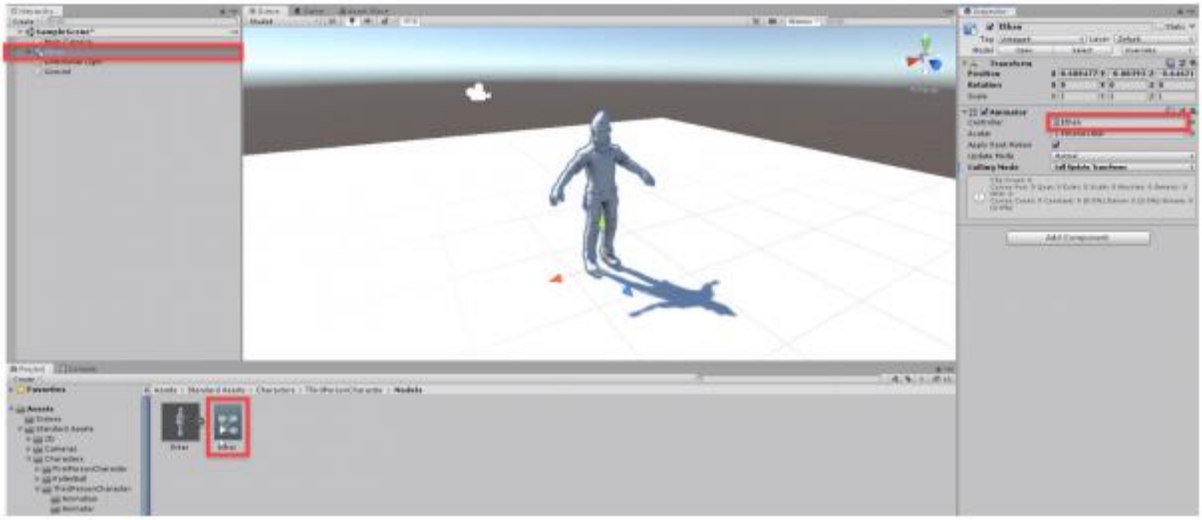


- **Controller:** An animation controller which lays out the animations we can player, their relations/transitions between each other and the conditions to do so.
- **Avatar:** The rig or skeleton that can morph our model.
- **Apply Root Motion:** If disabled, animations that move like a run animation will not move.
- **Update Mode:** How the animation frames play.
  - **Normal:** Updates every frame.
  - **Animate Physics**: Based on physics time step (50 times per second).
  - **Unscaled Time:** Normal, but not tied to Time.timeScale.
- **Culling Mode:** Determines if the animations should play off-screen.
  - **Always Animate:** Always animate the entire character. Object is animated even when offscreen.
  - **Cull Update Transforms:** Retarget, IK and write of Transforms are disabled when renderers are not visible.
  - **Cull Completely:** Animation is completely disabled when renderers are not visible.

For the next lesson, let's create an **Animation Controller** which will contain our animation state machine. In the Project panel, right click the Hierarchy > *Create* > *Animation Controller*. Call it **Ethan**



Now select the Ethan model in the scene and drag the controller into the **Controller** property.



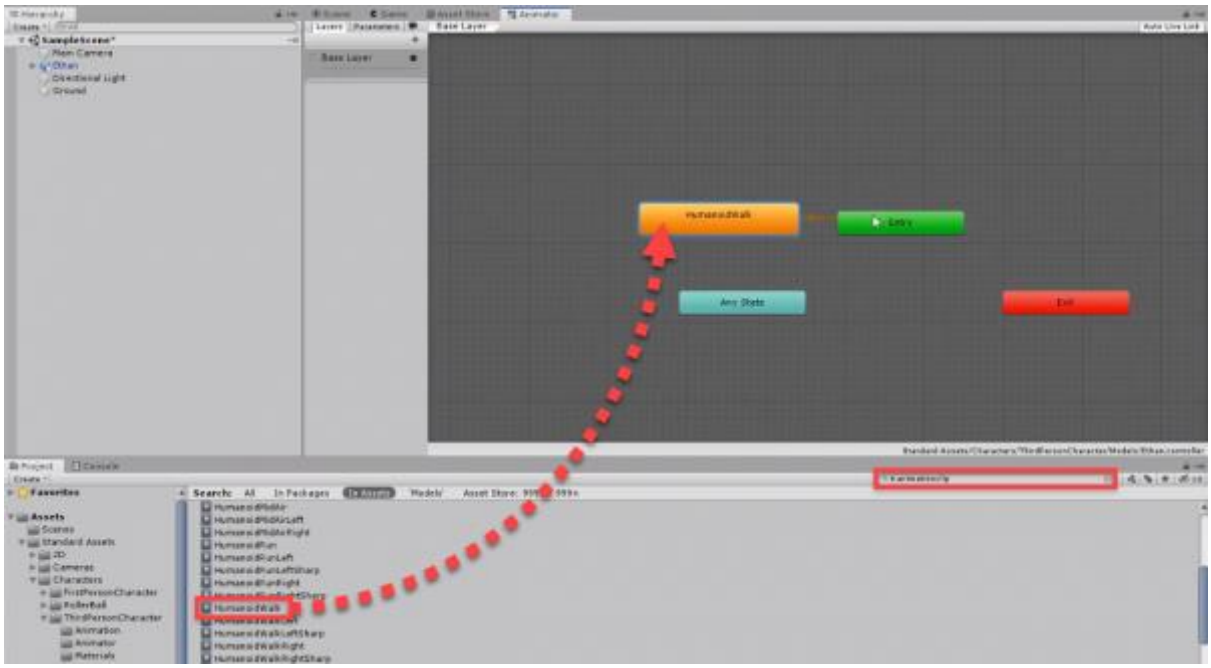
In the next lesson, we're going to be looking at animation state machines.

In this lesson, we're going to be looking at and creating an animation state machine in Unity. This involves states and transitions.

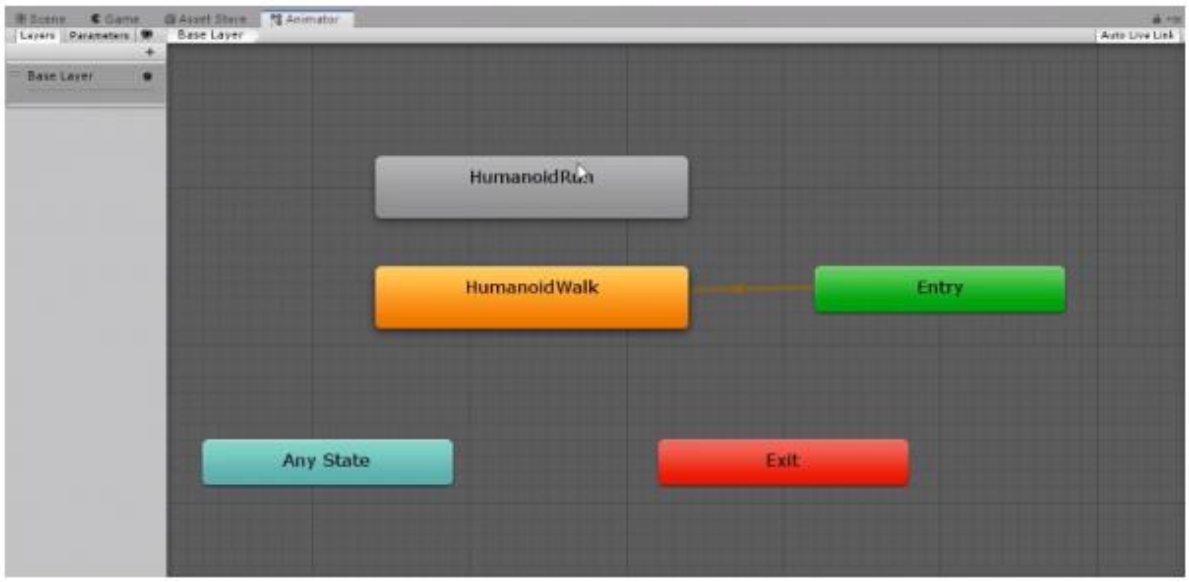
- **States:** Animations that will play when they are transitioned to.
- **Transitions:** Conditions to transition from one state to another.
- Let's now go over to Unity and open up the **Animator** window (*Window > Animator*).

Let's find an animation to drag into the animation controller. Search *t:animationclip* to find all the animation clips in the standard assets. Drag the **HumanoidWalk** animation into the Animator screen.

It being orange, means that this is the default state. This will be played first when the object is initialized.

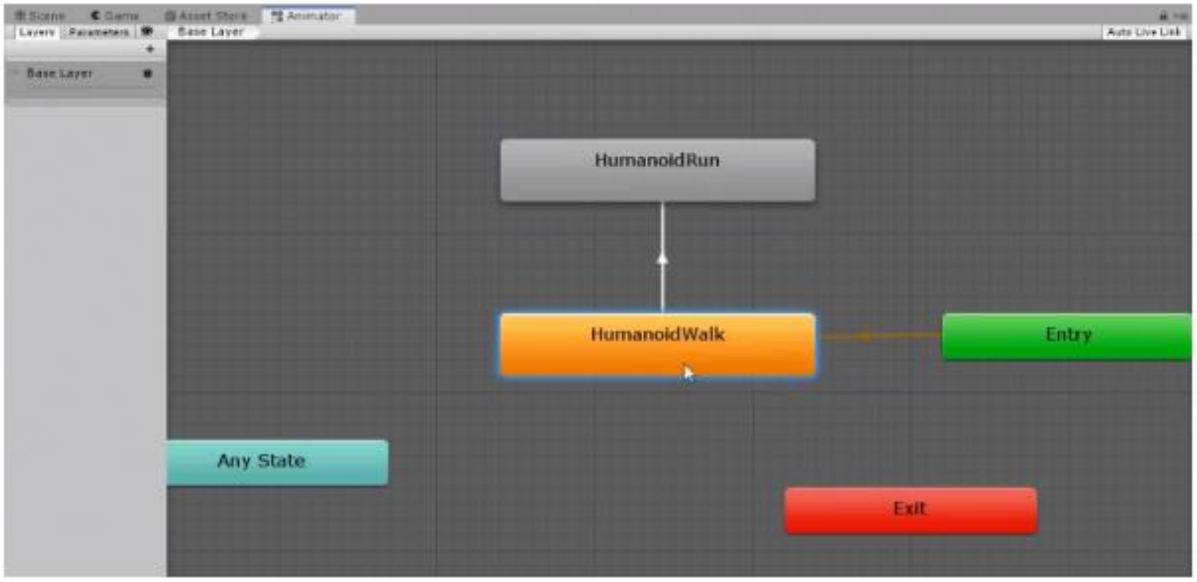


Let's now drag in the **HumanoidRun** clip. You'll see that this clip isn't orange. We can make it the default clip by right clicking it and selecting *Set as Layer Default State*.

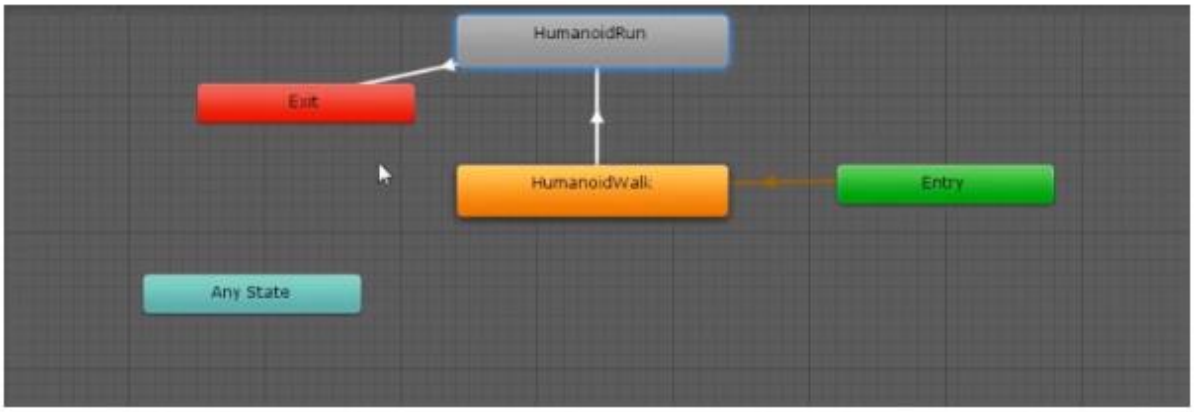


Let's now make a transition between these two states. Select the **HumanoidWalk** and click *Make Transition*. Then click on **HumanoidRun**. This will create a new transition from the walk

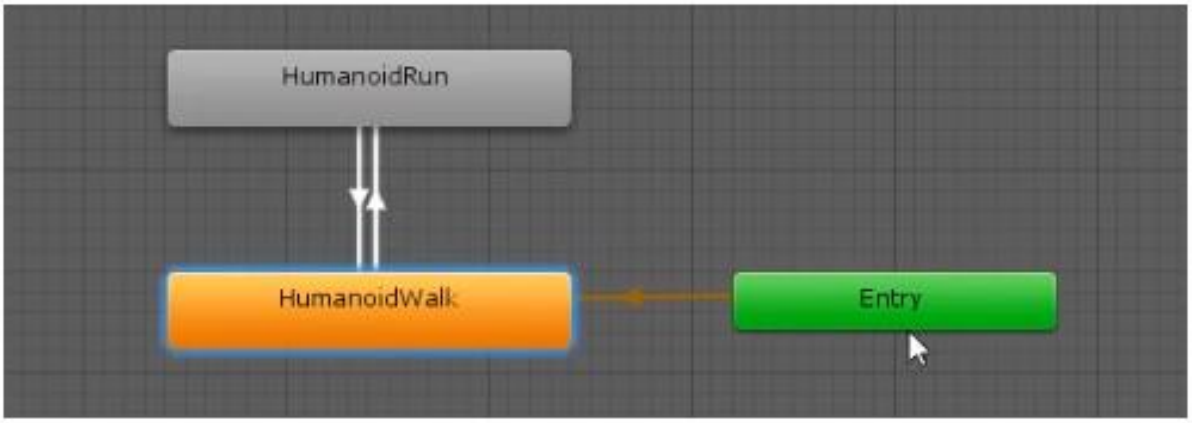
animation to the run animation. By default, this means that when the walk animation has finished playing, the run animation will play.



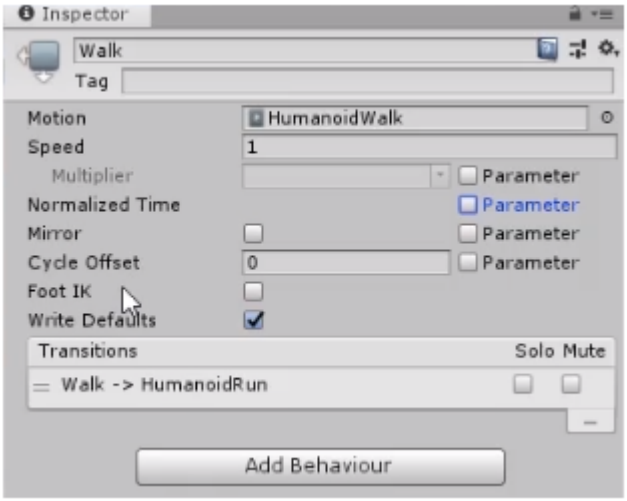
If we press play, the two animations will play. When the run animation is finished, nothing will happen. We can transition that into the **Exit** state, so that it loops back around to the **Entry** state.



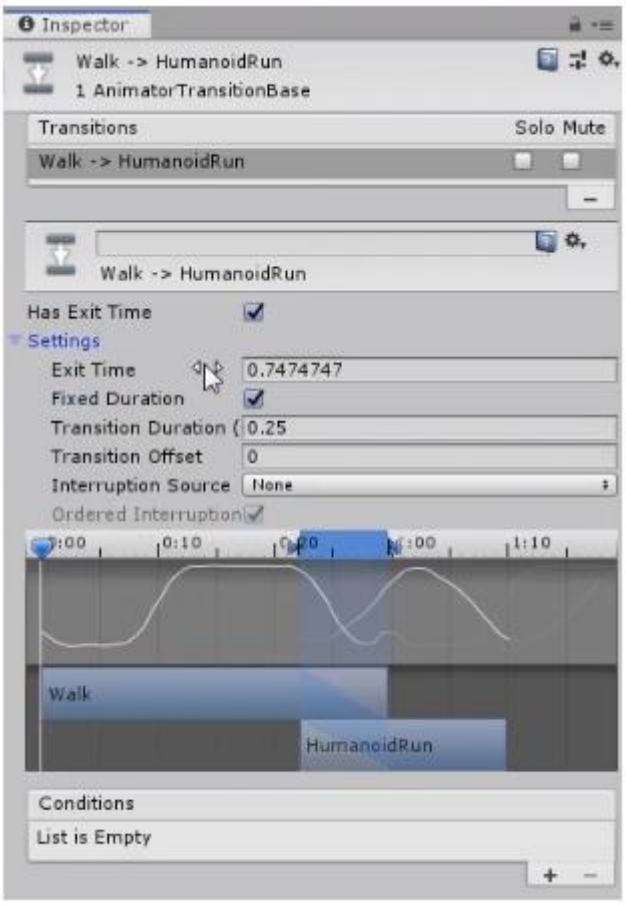
A better method is to just have the run animation transition back into the walk animation.



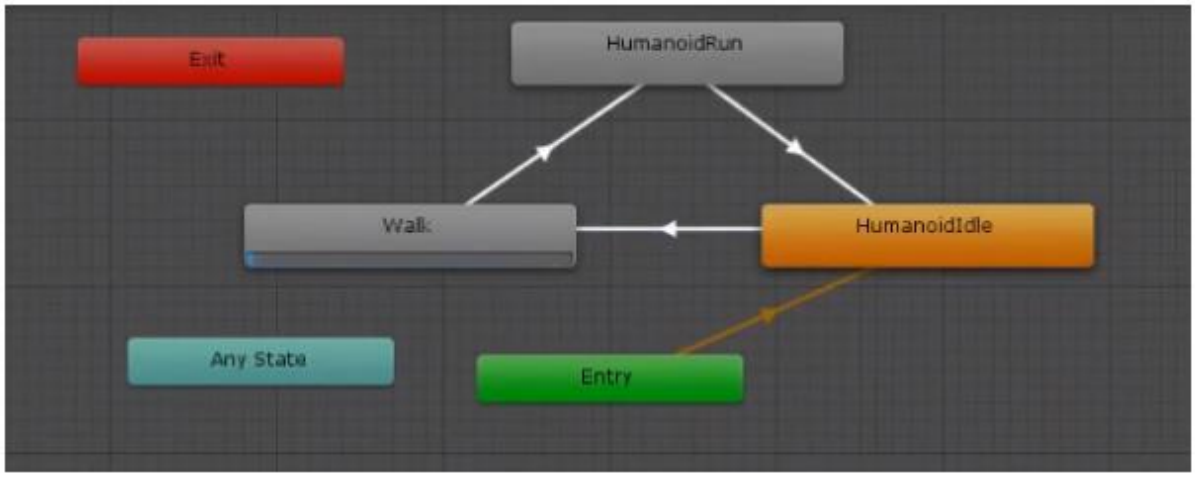
States :- Let's select the humanoid walk state and have a look at the Inspector.



Transitions :- Let's select a transition and have a look at the Inspector. Transitions can allow an animation to fade into another. Here, we can modify when, how and how fast the transition takes place.



Let’s now restructure our controller a bit and add in an idle animation for the default one.



Practical 8

Aim: Create Snowfall Particle effect in Unity.

We can achieve this effect using Unity's Particle System component. Following is a stepby-step guide on how to create a basic snowfall particle effect:

1. **Create a New Particle System**:
    - In the Unity Editor, select the GameObject where you want to add the snowfall effect.
    - Go to the menu bar and select **GameObject > Effects > Particle System**.

      This will create a new Particle System component attached to the selected GameObject.
2. **Configure the Particle System**:
    - In the Inspector window, you'll see the Particle System component's settings. Adjust these settings to create a snowfall effect:
    - **Duration**: Set this to a value that suits the length of your snowfall scene. For an ongoing snowfall, you can set it to a high value or loop the system.
    - **Start Lifetime**: This determines how long each snowflake particle will stay on screen. Set it to a value that makes the snowflakes fall for an appropriate amount of time (e.g., 5 seconds).
    - **Start Speed**: Adjust the initial speed of the snowflakes. Typically, a small value like 1-5 units per second will work.
    - **Start Size**: Set the size of the snowflakes. They should be small, like 0.05 to

      0.2.

- **Start Color**: Choose a light blue or white color to resemble snow.
- **Emission**: Configure the rate at which particles are emitted. Set the rate to create a dense snowfall. For example, you might try starting with 100 particles per second.
- **Shape**: Choose "Cone" as the shape and adjust the angle and radius to control the area where snowflakes will spawn.
- **Gravity Modifier**: Apply a downward force (negative value) to simulate gravity. Use a small negative value, like -0.1, to make snowflakes fall gently.

3. **Texture for Snowflakes**:
   - You can use a custom texture for your snowflakes. In the Particle System settings, under the Renderer module, set the Material to a particle material that uses a snowflake texture. Ensure the snowflake texture is set to have a transparent background.
4. **Tweak Additional Settings**:
   - You can further enhance the effect by adjusting settings like Color Over Lifetime, Size Over Lifetime, and Rotation Over Lifetime to add variation to the snowflakes.
5. **Loop the Snowfall** (Optional):
   - If you want the snowfall to continue indefinitely, check the "Looping" option in the Particle System settings.
6. **Play the Scene**:
   - Press the Play button in Unity to see your snowfall effect in action.
7. **Optimization**:

- Be mindful of performance. If you have a lot of particles, it can impact your game's performance. Adjust the particle count and other settings as needed for your project.

You can further refine and customize the effect by experimenting with different settings and textures to achieve the desired look for your game or scene.

Output: