Program 2:

Objective:

To create 2D assets, implement basic character movement, and apply sprite animation in Unity.

Step-by-Step Guide (Simplified):

STEP 1: Creating a Character GameObject

1. Right-click in the Hierarchy → Create Empty → Rename to "Player."
2. Right-click Player → 2D Object → Sprite to add a Sprite Renderer.

Select the Player GameObject and assign your character sprite in the Sprite Renderer.

STEP 2: Adding Physics Components

1. Click Add Component → Rigidbody2D:

1. Set Gravity Scale = 0 (for top-down games) or 1 (for platformers).
2. Click Add Component → Capsule Collider 2D (or Box Collider 2D) to detect collisions.

Step 3: Character Movement (Core Mechanic)

1. Creating the Movement Script
2. In the Assets folder, create a new folder called Scripts.
3. Right-click → Create → C# Script, name it PlayerMovement.cs.
4. Open the script and add the following code:

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;
    private Rigidbody2D rb;
    private Vector2 move;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }
    void Update()
    {
        move.x = Input.GetAxisRaw("Horizontal");
        move.y = Input.GetAxisRaw("Vertical");
    }

    void FixedUpdate()
    {
        rb.MovePosition(rb.position + move * speed * Time.fixedDeltaTime);
    }
}
```

Step 4:

Attaching the Script to the Player

1. Drag and drop the PlayerMovement.cs script onto the Player GameObject.
2. In the Inspector, adjust the Speed value (default 5).
3. Press Play and move the character using the Arrow keys.

Program 3 :

Step 1: Set Up Your Unity Scene

1. Open Unity and create a new 2D project.

2. Go to Window → Package Manager → Install 2D Tilemap Editor (if not

already

installed).

Step 2: Create a Tilemap

1. In the Hierarchy, right-click → 2D Object → Tilemap → Rectangular.

2. This automatically creates:

o Grid (Parent object)

o Tilemap (Child object for tiles)

o Tile Map Renderer (Handles rendering)

3. Select Tilemap → In Inspector, set:

o Tilemap Collider 2D (to detect collisions)

o Rigidbody 2D (set Body Type to Static)

Step 3: Import Tiles and Create a Tile Palette

1. Download or create tile images (PNG format with transparency).

2. Drag the tile images into Assets (Unity).

3. Open Tile Palette (Window → 2D → Tile Palette).

4. Click Create New Palette → Name it → Select a folder.

5. Drag your tile sprites into the Tile Palette.

6. Select Tilemap in the Hierarchy, then use the Brush Tool to paint tiles in the Scene.

OR

Step 3: Right click Grid --&gt; Design your levels(Multiple Squares)

Step 4: Add Interactive Objects (Coins, Doors, etc.)

1. Creating a Collectible (Coin)

1. Drag a coin sprite into the Scene.

2. Add Collider:

o Select the Coin object → In Inspector, click Add Component → Choose Circle

Collider 2D.

o Check Is Trigger (so it doesn’t act as a solid object).

3. Add a Script (CoinCollect.cs):

o Right-click in Assets → Create → C# Script → Name it CoinCollect.

```
using UnityEngine;
public class CoinCollect : MonoBehaviour
{
void OnTriggerEnter2D(Collider2D other)
{
if (other.CompareTag(“Player”))
{
Debug.Log(“Coin Collected!”);
Destroy(gameObject); // Removes coin
}
}
}
```

4. Assign this script to the Coin object.

5. Attach this script to the Player GameObject.

Tag the Coin Object

1. Select your Coin GameObject.

2. In the Inspector, click on the Tag dropdown (top of Inspector).

3. Click &quot;Add Tag&quot; → Create a new tag named &quot;Coin&quot;.

4. Assign the &quot;Coin&quot; tag to all coin objects in the scene.

2. Creating an Interactive Door

1. Drag a door sprite into the Scene.

2. Add a Box Collider 2D and check Is Trigger.

3. Add a Script (Door.cs):

4. Attach this script to the Door object.

```
using UnityEngine;

public class Door : MonoBehaviour

{

public GameObject player;

void OnTriggerEnter2D(Collider2D other)

{

        if (other.CompareTag("Player"))

        {

                player.SetActive(true); // Disables player when entering the trigger

        }

}

void OnTriggerExit2D(Collider2D other)

{

        if (other.CompareTag("Player"))

        {

                player.SetActive(false); // Enables player when exiting the trigger

        }

}
```

Step 5: Add a Player Character

1. Drag your player sprite into the Scene.

2. Add Components:

o Rigidbody 2D (Set Gravity Scale = 0 for a top-down game).

o Box Collider 2D (to detect collisions).

o Player Movement Script:

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour

{
```

```
public float moveSpeed = 5f;

private Rigidbody2D rb;

private Vector2 moveInput;

void Start()

{

rb = GetComponent<Rigidbody2D>();

}

void Update()

{

moveInput.x = Input.GetAxis(“Horizontal”);

moveInput.y = Input.GetAxis(“Vertical”);

}

void FixedUpdate()

{

rb.velocity = moveInput * moveSpeed;

}

}
```

3. Attach this script to the Player.

Step 6: Test the Level

Press Play and move the player around.

Program 4 :

Step 1: Set Up the Scene

1. Open Unity and create a new 3D project.

2. In the Hierarchy:

○ Right-click → 3D Object → Plane (this will be the ground).

○ Right-click → 3D Object → Cube (this will be the box to push).

○ Right-click → 3D Object → Capsule (this will be the player).

Step 2: Add Physics Components

1. Box (Cube):

○ Select the Cube.

○ In the Inspector, click "Add Component" → Rigidbody.

○ This makes the cube interact with Unity's physics engine.

2. Player (Capsule):

○ Add a Character Controller (Add Component → Character Controller).

○ (Optional) Add a Rigidbody if you want the player to be pushed by other forces (but often skipped if using Character Controller).

3. Set up the camera:

○ Click on the Camera in the Hierarchy.

○ In the Inspector, set the camera's position so it can view the player (e.g., Position: (0, 2, -10)).

○ Adjust the Camera's rotation to point towards the player (Rotation: (30, 0, 0)).

Step 3: Adding Player Movement with Physics

1. Add a Rigidbody component:

○ Select the "Player" object.

○ In the Inspector, click "Add Component" and search for "Rigidbody."

○ Add the Rigidbody component, which will allow physics interactions like gravity and collisions.

2. Create a Player Controller Script:

○ Right-click in the Project window and create a new C# script (e.g., PlayerController).

○ Double-click to open the script in Visual Studio.

3. Write basic movement code: Add the following code to your PlayerController script:

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
public float moveSpeed = 5f;
public float turnSpeed = 700f;
private Rigidbody rb;
void Start()
{
rb = GetComponent<Rigidbody>();
}
void Update()
{
// Movement
float moveHorizontal = Input.GetAxis("Horizontal");
float moveVertical = Input.GetAxis("Vertical");
Vector3 movement = new Vector3(moveHorizontal, 0, moveVertical) * moveSpeed *
Time.deltaTime;
rb.MovePosition(transform.position + movement);
// Turning
if (movement.magnitude > 0)
{
Quaternion targetRotation = Quaternion.LookRotation(movement);
transform.rotation = Quaternion.RotateTowards(transform.rotation, targetRotation,
turnSpeed * Time.deltaTime);

}

}

}
```

Program 5 :

```
import pygame

import sys


# Initialize Pygame

pygame.init()


# Screen dimensions

WIDTH, HEIGHT = 600, 600

screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption("Pygame 2D Game")


# Colors

WHITE = (255, 255, 255)

RED = (255, 0, 0)

BLACK = (0, 0, 0)


# Player settings

player_width, player_height = 50, 50

player_x, player_y = WIDTH // 2, HEIGHT // 2

player_speed = 5


# Font

font = pygame.font.Font(None, 56)


# Rectangles (obstacles, collectibles, etc.)

blocks = [

    pygame.Rect(100, 100, 200, 20),
    pygame.Rect(100, 200, 20, 200),
    pygame.Rect(200, 200, 20, 200),
    pygame.Rect(100, 400, 200, 20),
]

# Load sound (commented if no file present)
# move_sound = pygame.mixer.Sound("move.wav")

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Key press handling
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        player_x -= player_speed
        # move_sound.play()
    if keys[pygame.K_RIGHT]:
        player_x += player_speed
        # move_sound.play()
    if keys[pygame.K_UP]:
        player_y -= player_speed
        # move_sound.play()
    if keys[pygame.K_DOWN]:
        player_y += player_speed
```

```
            # move_sound.play()

        # Stay within bounds
        player_x = max(0, min(player_x, WIDTH - player_width))
        player_y = max(0, min(player_y, HEIGHT - player_height))

        # Drawing
        screen.fill(WHITE)
        pygame.draw.rect(screen, RED, (player_x, player_y, player_width, player_height))

        for block in blocks:
            pygame.draw.rect(screen, BLACK, block)

        score_text = font.render("Score: 0", True, BLACK)
        screen.blit(score_text, (10, 10))

        pygame.display.update()

    # Quit
    pygame.quit()
    sys.exit()
```

Program 6 :

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Screen settings
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Local Multiplayer Example")
clock = pygame.time.Clock()

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

# Player settings
player_size = 50
player1_pos = [100, 100]
player2_pos = [600, 400]
player_speed = 5

# Game loop
running = True
while running:
    clock.tick(60)  # 60 FPS

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Key handling
    keys = pygame.key.get_pressed()

    # Player 1 controls (WASD)
    if keys[pygame.K_w]: player1_pos[1] -= player_speed
    if keys[pygame.K_s]: player1_pos[1] += player_speed
    if keys[pygame.K_a]: player1_pos[0] -= player_speed
    if keys[pygame.K_d]: player1_pos[0] += player_speed

    # Player 2 controls (Arrow Keys)
    if keys[pygame.K_UP]: player2_pos[1] -= player_speed
    if keys[pygame.K_DOWN]: player2_pos[1] += player_speed
    if keys[pygame.K_LEFT]: player2_pos[0] -= player_speed
    if keys[pygame.K_RIGHT]: player2_pos[0] += player_speed

    # Drawing
    screen.fill(WHITE)
    pygame.draw.rect(screen, RED, (*player1_pos, player_size, player_size))
    pygame.draw.rect(screen, BLUE, (*player2_pos, player_size, player_size))
    pygame.display.flip()

pygame.quit()
sys.exit()
```

Program 7:

Create a basic **3D scene** in Unity with:

1. A player you can move
2. An enemy that follows the player (basic AI)
3. Simple animations (e.g., walking)

Step 1:

A. Create Ground

- Right-click in Hierarchy > `3D Object > Plane`
- Rename to `Ground`
- Set `Scale = (10, 1, 10)` in Inspector

B. Add Player Cube

- Right-click > `3D Object > Cube`
- Rename to `Player`
- Position: `(0, 0.5, 0)`

C. Add Enemy Cube

- Right-click > `3D Object > Cube`
- Rename to `Enemy`
- Position: `(5, 0.5, 5)`
- Color it red (optional):
  - Add `Material`, set color red, drag it onto the cube

Step 2: Add Player Movement

A. Create Player Script

- Right-click in `Assets` > `Create > C# Script` > name it `PlayerMovement`
- Drag the script onto the `Player` object

B. Code for PlayerMovement.cs:

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float moveSpeed = 5f;
    void Update()
    {
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");
        Vector3 move = new Vector3(h, 0, v);
        transform.Translate(move * moveSpeed * Time.deltaTime);
    }
}
```

This lets the player move with **WASD** or arrow keys.

Step 3: Add Simple AI (Enemy Follows Player)

A. Create Enemy Script

- Right-click in Assets > Create > C# Script > name it EnemyAI
- Attach it to the Enemy object

B. Code for EnemyAI.cs:

```
using UnityEngine;

public class EnemyAI : MonoBehaviour
{
    public Transform player;
    public float speed = 3f;

    void Update()
    {
        if (player != null)
        {
            Vector3 direction = (player.position - transform.position).normalized;
            transform.position += direction * speed * Time.deltaTime;
        }
    }
}
```

C. Link Player to Enemy:

- Click on `Enemy` in the Hierarchy
- In Inspector > EnemyAI script:
    - Drag `Player` from Hierarchy into the **Player** field

Now the enemy will move toward the player every frame.

Program 8 :

8th

```
import pygame

import sys

import random


# Initialize Pygame

pygame.init()


# Constants

WIDTH, HEIGHT = 800, 600

WHITE = (255, 255, 255)

PLAYER_COLOR = (0, 100, 255)

ENEMY_COLOR = (255, 150, 0)

PLAYER_SIZE = 50

ENEMY_SIZE = 80

PLAYER_SPEED = 8

ENEMY_SPEED = 2


# Setup

screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption("Dodge the Enemy")

clock = pygame.time.Clock()
```

```
font = pygame.font.SysFont(None, 36)


# Player and Enemy

player_pos = [WIDTH // 2, HEIGHT // 2]

enemy_pos = [random.randint(0, WIDTH - ENEMY_SIZE), random.randint(0, HEIGHT -
ENEMY_SIZE)]


# Game State

score = 0

start_ticks = pygame.time.get_ticks()

running = True

game_over = False


# --- Functions ---


def move_enemy(enemy_pos, player_pos):

    if enemy_pos[0] < player_pos[0]:

        enemy_pos[0] += ENEMY_SPEED

    elif enemy_pos[0] > player_pos[0]:

        enemy_pos[0] -= ENEMY_SPEED


    if enemy_pos[1] < player_pos[1]:

        enemy_pos[1] += ENEMY_SPEED
    elif enemy_pos[1] > player_pos[1]:
        enemy_pos[1] -= ENEMY_SPEED


def detect_collision(p_pos, e_pos):
    px, py = p_pos
    ex, ey = e_pos

    return (
        px < ex + ENEMY_SIZE and
        px + PLAYER_SIZE > ex and
        py < ey + ENEMY_SIZE and
        py + PLAYER_SIZE > ey
    )

# --- Game Loop ---

while running:
    clock.tick(60)
    screen.fill(WHITE)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```

```
if not game_over:

    # Movement

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT] and player_pos[0] > 0:

        player_pos[0] -= PLAYER_SPEED

    if keys[pygame.K_RIGHT] and player_pos[0] < WIDTH - PLAYER_SIZE:

        player_pos[0] += PLAYER_SPEED

    if keys[pygame.K_UP] and player_pos[1] > 0:

        player_pos[1] -= PLAYER_SPEED

    if keys[pygame.K_DOWN] and player_pos[1] < HEIGHT - PLAYER_SIZE:

        player_pos[1] += PLAYER_SPEED


    # Enemy Movement

    move_enemy(enemy_pos, player_pos)


    # Drawing

    pygame.draw.rect(screen, PLAYER_COLOR, (*player_pos, PLAYER_SIZE, PLAYER_SIZE))

    pygame.draw.rect(screen, ENEMY_COLOR, (*enemy_pos, ENEMY_SIZE, ENEMY_SIZE))


    # Score

    seconds = (pygame.time.get_ticks() - start_ticks) // 1000

    score_text = font.render(f"Score: {seconds}", True, (0, 0, 0))
```

```
        screen.blit(score_text, (10, 10))


        # Collision Check

        if detect_collision(player_pos, enemy_pos):

            game_text = font.render("Game Over!", True, (200, 0, 0))

            screen.blit(game_text, (WIDTH // 2 - 80, HEIGHT // 2 - 20))

            pygame.display.flip()

            pygame.time.delay(2000)

            game_over = True


    pygame.display.flip()


# Exit

pygame.quit()

sys.exit()
```

Program 9 :

9)

Developing a Mechanis for Camera 3D Physis and Game Core Game

---

Step 1: Set up a New Scenu

1. Open Unity → Create 3D profeet

2. In Hierarchy:

→ Right-click 3D object > Plane (Ground)

→ Right-click 3D object > Cube (Player)

→ Rename to Player.

→ Position: (10, 0.08, 0.3)

→ Add a Main Camera (already in scene)

Step 2: Add Physics to Player

→ Select the Player cube.

→ In Inspector:

→ Click Add Component > Rigidbody

→ This gives gravity and physics behaviour.

→ Remove “Use Gravity” for floating control

Step 3: Player Movement & Jumping

A. Creati the Script

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class BallController : MonoBehaviour
{
    public float force = 10f;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");
        Vector3 move = new Vector3(h, 0, v);

        rb.AddForce(move * force);

    }

}
```

→ Attach this option script to the player (Inspector)

→ Add tag in window

Step 4: Optimization of 3D Gonse

A. Statu Balching

→ Set “nan” as static moving objects in Inspector like ground and walls

B. Lighting

→ Use Baked lolling

→ Creindoro ligliding when possible

→ (Lightening bloked)

Step 5: Testing Your Game

→ Test mechani individually

→ Use Unity’s play mode to quickly debug

→ Press enemies to duplicati for layout testing (Obstacles)

Step 6: Build & Publish

1. Go to File > Build Youx Settings → Gane

2. Add your current scene > Add Open Scenes

3. Choose Platform (PC / WebGL / Android)

4. Click “Switch Platform” if needed

ENT: Player Settings

1. Set Add Ganu name

2. Full screen icon, resolution modi for realix settings

3. Turn builds off development

4. Build the game

→ Choose a folder

→ Click Build

→ Unity experts 01 executable fete (.exe or .apk)