GAME DEVELOPMENT
(Effective from the Academic Year 2023 - 2024)
VI SEMESTER

Course Code	**CS62298CC**	CIA Marks	50
Number of Contact Hours/Week (L: T: P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	20P	Exam Hours	03

CREDITS – 1

COURSE PREREQUISITES:

- Basics of C-Sharp language
- Basics of unity game engine

COURSE OBJECTIVES: This course will enable students:

- To install Unity and Unreal engine and become proficient in their GUI for game development.
- Develop the ability to conceptualize and define engaging themes for 2D games.
- To acquire skills in character design, sprite creation, character control and movement to create functional 2D gameplay.
- To design interactive game environments with tiles, interactive objects, and collectibles to enhance player engagement.
- To explore the design of player world interactions, with the option of using physics engines, for immersive and dynamic gameplay experiences.

TEACHING - LEARNING STRATEGY:

Following are some sample strategies that can be incorporate for the Course Delivery

- Chalk and Talk Method/Blended Mode Method
- Power Point Presentation
- Expert Talk/Webinar/Seminar
- Video Streaming/Self-Study/Simulations
- Peer-to-Peer Activities
- Activity/Problem Based Learning
- Case Studies
- MOOC/NPTEL Courses
- Any other innovative initiatives with respect to the Course contents

LIST OF EXPERIMENTS

Sl. No.	Description
1	Installation of a game engine, e.g., Unity familiarization of the GUI.
2	Creation of 2D assets, Character Movement Program core mechanic, Sprite animation.
3	Level design: design of the world in the form of tiles along with interactive and collectible objects.
4	Design of interaction between the player and the world, optionally using the physics engine.
5	Developing a 2D interactive using Pygame.
6	Developing a multiplayer experience.
7	Design 3D environment, animation and AI behavior
8	Developing a camera, Physics and core game mechanics for 3D game.

9	Developing a physics-based mechanic for 3D game, optimization of 3D, Testing, Publishing and delivery.

COURSE OUTCOMES

Upon completion of this course, the students will be able to:

CO No.	Course Outcome Description	Bloom's Taxonomy Level
CO1	Apply game engine expertise to install and navigate game engines like unity and unreal engine.	CL3
CO2	Create conceptually sound 2D game themes.	CL3
CO3	Implement 2D game elements by executing their character design, character control, and movement to construct functional game play experiences.	CL3
CO4	Design interactive game environments through the creation of game worlds using tiles, interactive objects, and collectibles.	CL3
CO5	Implement 3D environment, animation and behavior along with physics based and shooter-based 3D game mechanics	CL3

CO-PO-PSO MAPPING

CO No.	Programme Outcomes (PO)												Programme Specific Outcome (PSO)		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
CO1	3	2	3		2							1	1	1	1
CO2	3	2	3		2							1	1	1	1
CO3	3	2	3		2							1	1		1
CO4	3	2	3		2							1		1	
CO5	3	2	3		2							1	1		1

3: Substantial (High)	2: Moderate (Medium)	1: Poor (Low)

PROGRAM 1:

Installation of a game engine, e.g., Unity familiarization of the GUI.

Objective:

To install Unity and become familiar with its graphical user interface (GUI) for game development.

Step-by-Step Procedure:

1. **Download Unity Hub:**
   - Go to the official Unity website: https://unity.com
   - Navigate to the "Downloads" section and download **Unity Hub** for your operating system.
2. **Install Unity Hub:**
   - Run the installer file.
   - Follow on-screen instructions to complete installation.
3. **Create a Unity Account (if not already created):**
   - Open Unity Hub.
   - Sign in with an existing Unity account or create a new one.
4. **Install Unity Editor via Unity Hub:**
   - In Unity Hub, go to the **Installs** tab.
   - Click **Install Editor**, select the desired version (LTS recommended).
   - Choose additional modules (e.g., Windows Build Support, Android, etc.) as needed.
   - Click **Install** and wait for the process to complete.
5. **Create a New Project:**
   - Go to the **Projects** tab in Unity Hub.
   - Click **New Project**.
   - Select a template (e.g., 2D, 3D).
   - Name your project and select a location.
   - Click **Create**.
6. **Familiarize with Unity GUI:**
   - **Scene View:** Allows you to visually place and manipulate game objects.
   - **Game View:** Shows what the game will look like to the player.
   - **Hierarchy Window:** Lists all game objects in the current scene.
   - **Inspector Window:** Displays and allows editing of selected object properties.
   - **Project Window:** Shows all assets (scripts, prefabs, textures, etc.).
   - **Console Window:** Displays error messages, warnings, and debug logs.
7. **Basic GUI Interactions:**
   - Move, rotate, and scale objects in the Scene.
   - Create a new GameObject (e.g., 2D Sprite or 3D Cube).
   - Modify components like Transform, Rigidbody, and Sprite Renderer in the Inspector.
8. **Save the Scene and Project:**
   - Go to **File > Save Scene** and name your scene.
   - Use **Ctrl + S** regularly to save your work.

PROGRAM 2:

Creation of 2D assets, Character Movement Program core mechanic, Sprite animation.

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:

Level design: design of the world in the form of tiles along with interactive and collectible objects.

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 --> 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!&amp;quot");
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 "Add Tag" → Create a new tag named "Coin".

4. Assign the "Coin" 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(false); // Disables player when entering the trigger
        }
    }

    void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            player.SetActive(true); // 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&lt;Rigidbody2D&gt;();
}

void Update()
{
moveInput.x = Input.GetAxis(&quot;Horizontal&quot;);
moveInput.y = Input.GetAxis(&quot;Vertical&quot;);
}

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

Design of interaction between the player and the world, optionally using the physics engine.

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);
        }
    }
}
```

Expt. No. .....

Date

5. Developing a 2D interactive using pybrame.

Install Pygame:
pip install pygame.

Program:

```
import pygame.
import sys.

pygame.init()

WIDTH, HEIGHT = 300,600.
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Pygame 2D game").

WHITE = (255, 255,255)
RED = (255,0,0).

player_width, player_height =50,50.
player_x, player_y = WIDTH//2, HEIGHT//2.
player_speed =5.

font = pygame.font.Font( None, 36).

walls = [
    pygame.Rect (100, 100, 200, 20),
    pygame.Rect (100, 200, 20, 200),
    pygame.Rect (300, 200, 20, 200),
```

Expt. No. .....

Date

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

move_sound = pygame.mixer.Sound("move.wav").

running = True.

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

    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().

player_x = max(0, min(player_x, WIDTH - player_width))
player_y = max(0, min(player_y, HEIGHT - player_height))
```

```
screen.fill(WHITE)

pygame.draw.rect(screen, RED, (player_x, player_y,
player_width, player_height))

for wall in walls:
    pygame.draw.rect(screen, (0, 0, 0), wall)

score_text = font.render("Score: 0", True, (0,0,0))

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

pygame.display.update()

pygame.quit()

sys.exit()
```

10/10

Seen 30/5/25

6. Developing a multiplayer experience.

```
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()
```

7. Design 3D environment, animation and AI behavior;

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.

Expt. No. .....

Date

8. Developing a python program to add a camera physics and core game mechanics for 3D game.

```
import pygame
import sys
import random
pygame.init()

WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Dodge the Enemy")
WHITE = (255, 255, 255)
PLAYER_COLOR = (0, 100, 255)
ENEMY_COLOR = (255, 50, 50)
clock = pygame.time.Clock()
font = pygame.font.SysFont(None, 36)
player_size = 50
player_pos = [WIDTH // 2, HEIGHT // 2]
player_speed = 8
enemy_size = 50
enemy_pos = [random.randint(0, WIDTH - enemy_size),
    random.randint(0, HEIGHT - enemy_size)]

enemy_speed = 2
score = 0

start_ticks = pygame.time.get_ticks()
```

```
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
    )

running = True
game_over = False
while running:
    clock.tick(50)
    screen.fill(WHITE)

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

Expt. No. ..... Date □□□□□□

```
if not game_over:
    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[20] += 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

move_enemy(enemy_pos, player_pos)
pygame.draw.rect(screen, PLAYER_COLOR(*player_pos, player_size
                                        player_size))
pygame.draw.rect(screen, ENEMY_COLOR,(*enemy_pos, enemy_size,
                                        enemy_size))

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))

if detect_collision(player_pos, enemy_pos):
    game_over_text = font.render("Game Over!", True, (20,0,0))
    screen.blit(game_over_text, (WIDTH//2-20, HEIGHT//2))
    pygame.display.flip()
```

Expt. No. .....

Date

```
pygame.time.delay(2000)
game_over = True:

pygame.display.flip()
pygame.quit()
```

10/10

seen 20/5/25

Expt. No. ..... Date ☐☐☐☐☐☐

9 Developing a camera physics and core game mechanics for 3D game.

Step 1: Set up a New Scene.

1. Open Unity -> Create a 3D project.
2. In Hierarchy:
   - Right-click -> 3D object > Plane (Ground).
   - Right-click -> 3D object > Cube (Player).
     - Rename to Player.
     - Position: (0, 0.5, 0).
   - Add a Main Camera (already in scene).

Step 2: Add Physics to Player.

1. Select the Player cube.
2. In Inspector:
   - Click Add Component > Rigidbody.
     - This gives gravity and physics behaviour.
     - Remove use gravity for floating control.

Step 3: Player Movement & Jumping.

A. Create the Script

```
Using UnityEngine;

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

Expt. No. ..... Date

```
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 script to player and add tag option in the inspector window.

Step 2: Optimization of 3D game.

A. Static batching.

Set non moving objects like ground as walls as static in inspector.

B. Lighting:

Use baked lighting when possible, (Window → Lighting → baked).

Step 3:

Testing your game.

→ Test each mechanic individually

Expt. No. ..... Date □□□□□□

→ Use Unity's play mode to quickly debug.
→ Press ctrl+D to duplicate obstacles or enemies for layout testing.

Step 4:

1) Build & publish Your game.
Go to file > build Settings.
2) Add you current scene → add open scenes.
3) Choose a Platform (PC, WebGL / android).
4) Click switch platform if needed.

Player Settings.

i) Set game name.
ii) Add icon, resolution, full screen settings. turn off development mode for realse builds.
iii) Build the game.
→ Choose a folder.
→ Click build.
→ Unity exports an executable file (.exe or .html)