SVKM'S NMIMS
Mukesh Patel School of Technology Management & Engineering
Department of Computer Engineering
AI Lab
Subject- Introduction to AR-VR

Aim: Build a simple 3D "Playground" that covers 20 Unity fundamentals: editor layout, scenes, GameObjects & components, transforms, C# scripting, Update vs FixedUpdate, keyboard input, frame-rate independent movement, Rigidbody/Colliders, collisions, triggers, object interaction, TextMesh Pro, dynamic UI text, UI feedback on events, hierarchy & naming, building an .exe, quitting on ESC, testing builds, and debugging.

What You'll Learn (Outcomes)

1. Unity Editor Interface & Layout
2. GameObjects & Components
3. Transforms (Position, Rotation, Scale)
4. Creating & Managing Scenes
5. Intro to C# scripting in Unity
6. Update() vs FixedUpdate()
7. Keyboard input
8. Frame-rate independent movement (Time.deltaTime)
9. Rigidbody & Colliders
10. Collision detection (OnCollisionEnter
11. Trigger events (OnTriggerEnter/OnTriggerExit)
12. Object interaction (move, rotate, color change)
13. Intro to TextMesh Pro (TMP)
14. Displaying dynamic text
15. UI feedback on collision/trigger
16. Scene hierarchy & naming conventions
17. Exporting & building (.exe)
18. Application Quit on ESC key
19. Testing executable builds
20. Debugging & Console messages

Prerequisites:

- Unity Hub + a recent LTS Unity Editor (6 or newer).
- Code editor (VS Code/Visual Studio).
- Basic familiarity with C# syntax.

Tip: If prompted, install TMP Essentials via **Window → TextMeshPro → Import TMP Essential Resources**.

Theory:

1. **MonoBehaviour**

The base class from which every Unity script derives. It provides access to Unity's event functions and lifecycle methods.
Unity Docs — MonoBehaviour

Common Methods: - Start() — called once before the first frame update. - Update() — called every frame. - Awake() — called before any Start methods. - OnCollisionEnter(Collision) — detects physical collisions. - OnTriggerEnter(Collider) — detects trigger overlaps.

2. **GameObject**

The fundamental entity in Unity scenes that represents characters, props, and environments.
Unity Docs — GameObject

Important Members: - transform — the Transform component of the object. - AddComponent<T>() — dynamically attach a component. - GetComponent<T>() — retrieve a component.

3. **Transform**

Defines position, rotation, and scale of a GameObject in 3D space.
Unity Docs — Transform

Important Properties: - position — world position of the object.
- rotation — world rotation (Quaternion).
- localScale — scale relative to the parent.

Common Methods: - Translate(Vector3) — move relative to direction and space.
- Rotate(Vector3) — apply rotation in degrees.

4. **Rigidbody**

A component that enables physics simulation (gravity, collisions).
Unity Docs — Rigidbody

Key Members: - velocity — the linear velocity vector.
- AddForce(Vector3, ForceMode) — applies force to the object.
- MovePosition(Vector3) — safely moves a Rigidbody (used here with Time.deltaTime).

5. **Collider**

Defines physical boundaries for collisions.
Unity Docs — Collider

Key Properties: - isTrigger — when true, collider acts as a trigger. - bounds — world-space AABB box.

6. **Input**

Handles player input from keyboard, mouse, or joystick (legacy system).
Unity Docs — Input

Common Methods: - GetAxisRaw(string) — returns raw input for axes (e.g., “Horizontal”).
- GetKeyDown(KeyCode) — checks if a key was pressed during this frame.

7. **Time.deltaTime**

Gives the duration of the last frame in seconds, ensuring smooth motion across different frame rates.
Unity Docs — Time.deltaTime

Use: Multiply speeds by Time.deltaTime to make movement consistent regardless of FPS.

8. **Renderer**

Controls how the object appears visually.
Unity Docs — Renderer

Key Members: - material.color — changes the object’s color dynamically.

9. **TextMeshProUGUI (TMP_Text)**

A component for rendering crisp, flexible text in UI.
Unity Docs — TMP_Text

Used In: TextHud.cs to update score and status text at runtime.

10. **Application**

Handles runtime control such as quitting the app.
Unity Docs — Application

Key Method: - Application.Quit() — closes the built game.

11. **Debug**

Provides logging to the Unity Console for testing and debugging.
Unity Docs — Debug

Common Methods: - Debug.Log(object) — logs messages. - Debug.LogWarning(object) — logs warnings. - Debug.LogError(object) — logs errors.

12. **Color**

Represents RGBA color values in Unity.
Unity Docs — Color

Examples: - Color.red, Color.white, new Color(0.2f, 0.8f, 0.3f).

13. **Quaternion**

Used for rotations without gimbal lock.
Unity Docs — Quaternion

Example: - transform.rotation = Quaternion.Euler(0,90,0);

14. **Vector3**

Represents a 3D vector or point.
Unity Docs — Vector3

Common Uses: - Positions, directions, forces, velocities.

Properties: - Vector3.forward, Vector3.up, Vector3.zero.

Summary of Key Theoretical Concepts

Concept	Description	Key Functions	Official Docs
Object Hierarchy	Scene → GameObject → Components	N/A	Scene Management
Game Loop	Unity executes Update() each frame	Update(), Start()	Execution Order
Physics System	Realistic movement/collision	Rigidbody, Collider, AddForce()	Physics
Input System (Legacy)	Detects keyboard/mouse	Input.GetAxis(), Input.GetKeyDown()	Input Manager
UI System	On-screen feedback	TMP_Text, Canvas	UI Toolkit
Debugging	Identify issues during play	Debug.Log()	Debug Class

Stage 0 - Create Project & Scene (Topics 1, 4)

Do 1. Open **Unity Hub → New Project → 3D (URP or Built-in)**. Name it UnityPlaygroundLab.

Why this matters
A *Scene* is a level or screen in your game. Learning to create and save scenes is core to project structure.

Check
You see SampleScene in your Project window. The Hierarchy shows *Main Camera* and *Directional Light*.

Stage 1 - Editor Interface & Layout (Topic 1)

Do - Explore the **Scene** view (editing), **Game** view (play result), **Hierarchy** (objects in scene), **Inspector** (properties), **Project** (files), **Console** (logs/errors). - Rearrange panels if needed; save a custom layout via **Window → Layouts**.

Why this matters
Understanding where to *find* and *tune* things makes you faster and prevents simple mistakes.

Pitfalls
- Accidentally moving the Scene camera (W/E/R/T for Move/Rotate/Scale/Rect). Use **F** to focus selected object.

Stage 2 — GameObjects & Components (Topics 2, 16)

1.add the following: -

Ground: 3D Object → **Plane**. Scale (4,1,4). Position (0,0,0).
Ramp: Cube, Scale (6,1,3), Position (4,0.5,0), Rotation X = 25°.
Player: Cube at (0,0.5,0). Add **Rigidbody** + **BoxCollider**.
Crate_A: Cube at (2,0.5,0). Add **Rigidbody** + **BoxCollider**.
Spinner_B: Cylinder at (0,1,5). Keep default collider.

Why this matters
GameObjects are containers; Components give them behavior (rendering, physics, scripts). Hierarchy/naming keep things readable.

Check
- You can select objects and see their Components in the Inspector.
- Press **Play**; Player & Crate fall if no Ground collider—so add BoxCollider to **Ground**.

Optional(But Recommended)

Create 3 Empty GameObjects _Environment, _Gameplay, _UI and segregate all the above objects or any objects created into these objects as children to make your hirarchy more systematic and clean

Stage 3 — Transforms (Topic 3)

Do - With **W/E/R**, practice **Move/Rotate/Scale** on objects.
- For precision, edit **Transform** fields in the Inspector.

Why this matters
Nearly all spatial changes flow through Transforms—mastering them is foundational.

Pitfalls
- Scaling physics objects non-uniformly can cause odd collider behavior. Prefer uniform scale for physics bodies.

Stage 4 — Create a Trigger Zone (Topics 2, 11)

1. Create TriggerZone (empty) at (−5,0,0).
2. Add child **TriggerArea**: Cube, Scale (2,1,2). In its **BoxCollider**, check **Is Trigger**.
3. Give it a visible material color.

Why this matters
A *Trigger* detects overlaps without physical pushback. Great for zones, pickups, and sensors.

Check
- Entering/leaving should call OnTriggerEnter/Exit once scripts are added.
- Try making small scripts with a simple Debug log (Debug.Log()).

Stage 5 — Scripting Basics (Topic 5)

Key idea
Unity uses C# classes deriving from MonoBehaviour. Common lifecycle methods: Awake (setup), Start (first frame), Update (per frame), FixedUpdate (per physics tick).

Do - Create a folder **Scripts** in Project.
- Add scripts shown in the **Appendix** and attach as instructed in later stages.

Why this matters
Scripts bridge player input, physics, UI, and object logic.

Stage 6 — Movement, Update vs FixedUpdate, Frame-rate Independence (Topics 6, 7, 8)

Do 1. Create **PlayerController.cs** (attach to *Player*). See Appendix for full code. 2. Set moveSpeed = 6, jumpForce = 5, maxSpeed = 10. 3. Make the camera child of player to make a simple camera follower (Optional) Create **CameraFollow.cs** (attach to *Main Camera*); set target to *Player*.

Why this matters
- **Update()** runs every rendered frame—good for input.
- **FixedUpdate()** runs at fixed steps—good for physics.
- Using Time.deltaTime/physics forces makes movement frame-rate independent.

Check
Press **Play**: WASD moves, Space jumps, camera follows.

Pitfalls
- Moving a Rigidbody directly via transform.position each frame can fight the physics engine. Prefer forces/velocity changes in **FixedUpdate()**.

Stage 7 — Rigidbody & Colliders (Topic 9)

Do

- Ensure **Player** and **Crate_A** both have **Rigidbody + Collider**.
- **Ground** has **BoxCollider** (no Rigidbody) → acts as static floor.

Why this matters
One of the colliding objects must have a Rigidbody for physics interactions/callbacks.

Check
Objects rest on the Ground and collide realistically.

Stage 8 — Collisions & Triggers + UI Feedback (Topics 10, 11, 15, 20)

Do

1.Create a new TextMeshPro UI Object in the hirarchy (Right Click>UI>TextMeshPro) and rename it to HUD_Text1 this should have also created a canvas name the Canvas HUD_Canvas. Position the HUD in bottom right of the canvas

2. Create **CollisionReporter.cs** (attach to *Player*). It logs collisions & triggers and updates HUD. (Don't Forget to assign the freshly created TextMeshPro Object to the script in inspector)

3. Create **TriggerZone.cs** (attach to *TriggerArea*). Assign targets whose color should change on enter/exit. Try using different colors and different objects.

4. click the Player GameObject from hirarchy. In The hirarchy click on untagged>Player (This would tag the player so that it can be filtered from any other unexpected triggers)

Why this matters
- OnCollisionEnter(Collision) fires on physical contact.
- OnTriggerEnter(Collider) fires on overlap (no force).
- Logging to **Console** and showing UI feedback helps testing.

Check
Run into *Crate_A* → see Console log and HUD update. Enter/leave TriggerArea → see color change + HUD text.

Pitfalls
- Forgot to tag Player as Player? The trigger script may early-return.
- Missing Rigidbody on at least one of the overlapping objects? No trigger/collision calls.

Stage 9 — Object Interaction: Move/Rotate/Color (Topic 12)

Do

- Create **Rotator.cs** (attach to *Spinner_B*) to rotate every frame.
- Create **MoverPingPong.cs** (attach to *Crate_A*) to move back-and-forth.
- Use **TriggerZone.cs** to change colors while Player stays in the zone.

Why this matters
Small scripts like these teach continuous motion & property manipulation.

Check
Spinner rotates smoothly; Crate slides in a ping-pong pattern; entering trigger recolors targets.

Try doing

- Modify the TriggerZone.cs and Rotator.cs or MovePingPong.cs to stop or start rotation or the back and forth motion.(Requesting you to refrain from using genAI for this)

Stage 10 — TextMesh Pro & Dynamic Text (Topics 13, 14, 15)

Do

1. **Hierarchy → UI → Canvas (Screen Space – Overlay)**.
2. Under Canvas add: **TMP Text – Text (UI)** named TMP_ScoreText (top-left), and another TMP_InfoText (top-right).

3. Create empty child of Canvas named HUD_Controls and attach **TextHud.cs**. Drag both TMP references into Inspector fields.

4. (Optional) Add **ScorePickup.cs** to *Crate_A* to award points on collision.

Why this matters
TMP provides crisp, flexible text with rich markup—ideal for debugging and player feedback.

Check
On play, HUD shows a welcome message. Collisions/Triggers update Info; hitting Crate bumps score.

Pitfalls
- If TMP objects show Missing (TMP), import **TMP Essentials** from the menu.

Stage 11 — Naming & Hierarchy Conventions (Topic 16)

Guidelines

- Use group prefixes: _Environment, _Gameplay, _UI.
- PascalCase for scripts; specific names for scene objects (Crate_A, TriggerArea).
- Keep related objects under empty parents to stay organized.

Why this matters
Clean hierarchy reduces search time and makes collaboration easier.

Stage 12 — Build to Executable (Topics 17, 18, 19)

Do

1. **File → Build Settings… → Add Open Scenes**. Choose **Windows** (or your OS).
2. Click **Player Settings…**: set Product Name if desired.
3. Click **Build**, choose Builds/Windows folder.

Quit on ESC
Add **QuitOnEsc.cs** to Canvas (or any always-present object). In Editor, it stops Play Mode; in a build, it quits the app.

Test the .exe - Move/jump; verify physics & HUD.
- Press **ESC** → application closes.

Why this matters
Shipping a build verifies that project settings, code, and assets work outside the Editor.

Appendix — Scripts

PlayerController.cs

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class PlayerController : MonoBehaviour
{
    public float moveSpeed = 6f;        // WASD movement
    public float jumpForce = 5f;        // Space jump
    public float maxSpeed = 10f;        // Clamp velocity

    Rigidbody rb;
    Vector3 input;

    void Awake() => rb = GetComponent<Rigidbody>();

    void Update() // Input & non-physics
    {
        float h = Input.GetAxisRaw("Horizontal");
        float v = Input.GetAxisRaw("Vertical");
        input = new Vector3(h, 0f, v).normalized;

        if (Input.GetKeyDown(KeyCode.Space) && Mathf.Abs(rb.velocity.y
) < 0.05f)
            rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
    }

    void FixedUpdate() // Physics
    {
        Vector3 targetVel = input * moveSpeed;
        Vector3 vel = rb.velocity;
        Vector3 velChange = (targetVel - new Vector3(vel.x, 0f, vel.z)
);
        rb.AddForce(velChange, ForceMode.VelocityChange);

        Vector3 horiz = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
        if (horiz.magnitude > maxSpeed)
        {
            Vector3 clamped = horiz.normalized * maxSpeed;
            rb.velocity = new Vector3(clamped.x, rb.velocity.y, clampe
d.z);
        }
    }
}
```

CameraFollow.cs

```
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public Transform target;
    public Vector3 offset = new Vector3(0f, 6f, -8f);
    public float smooth = 8f;

    void LateUpdate()
    {
        if (!target) return;
        transform.position = Vector3.Lerp(transform.position, target.p
osition + offset, smooth * Time.deltaTime);
        transform.LookAt(target);
    }
}
```

CollisionReporter.cs

```
using UnityEngine;

public class CollisionReporter : MonoBehaviour
{
    public TextMeshProUGUI hud; // assign in Inspector

    void OnCollisionEnter(Collision c)
    {
        Debug.Log($"Player collided with {c.gameObject.name}");
        if (hud) hud.SetInfo($"Collision with <b>{c.gameObject.name}</
b>");
    }

    void OnTriggerEnter(Collider other)
    {
        Debug.Log($"Entered trigger: {other.gameObject.name}");
        if (hud) hud.SetInfo($"Entered trigger: <b>{other.gameObject.n
ame}</b>");
    }

    void OnTriggerExit(Collider other)
    {
        Debug.Log($"Exited trigger: {other.gameObject.name}");
        if (hud) hud.SetInfo($"Exited trigger: <b>{other.gameObject.na
me}</b>");
    }
}
```

TriggerZone.cs

```
using UnityEngine;

public class TriggerZone : MonoBehaviour
{
    public Renderer[] colorTargets; // drop objects whose color you wa
nt to change
    public Color enterColor = Color.green;
    public Color exitColor = Color.white;

    void OnTriggerEnter(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        foreach (var r in colorTargets)
            if (r) r.material.color = enterColor; // color change on e
nter
    }

    void OnTriggerExit(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        foreach (var r in colorTargets)
            if (r) r.material.color = exitColor; // revert on exit
    }
}
```

Rotator.cs

```
using UnityEngine;

public class Rotator : MonoBehaviour
{
    public Vector3 eulerPerSecond = new Vector3(0f, 90f, 0f);

    void Update()
    {
        transform.Rotate(eulerPerSecond * Time.deltaTime, Space.World)
;
    }
}
```

MoverPingPong.cs

```
using UnityEngine;

public class MoverPingPong : MonoBehaviour
{
    public Vector3 localAxis = Vector3.right;
    public float distance = 2f;
    public float speed = 1f;
    Vector3 start;

    void Start() => start = transform.position;

    void Update()
    {
        float t = (Mathf.Sin(Time.time * speed) + 1f) * 0.5f; // 0..1
        transform.position = start + localAxis.normalized * (t * dista
nce);
    }
}
```

TextHud.cs

```
using UnityEngine;
using TMPro;

public class TextHud : MonoBehaviour
{
    public TMP_Text scoreText; // assign TMP_ScoreText
    public TMP_Text infoText;  // assign TMP_InfoText

    int score;

    void Start()
    {
        SetScore(0);
        SetInfo("Welcome! WASD to move, Space to jump.");
    }

    public void AddScore(int delta)
    {
        score += delta;
        SetScore(score);
        Debug.Log($"Score changed to {score}");
    }

    public void SetInfo(string message)
```

```
    {
        if (infoText) infoText.text = message;
    }

    void SetScore(int s)
    {
        if (scoreText) scoreText.text = $"Score: <b>{s}</b>";
    }
}
```

ScorePickup.cs

```
using UnityEngine;

public class ScorePickup : MonoBehaviour
{
    public int value = 5;

    void OnCollisionEnter(Collision c)
    {
        var hud = FindFirstObjectByType<TextHud>();
        if (hud)
        {
            hud.AddScore(value);
            hud.SetInfo($"Picked up: +{value}");
        }
        if (TryGetComponent<Rigidbody>(out var rb))
            rb.AddForce(Vector3.up * 2f, ForceMode.Impulse);
    }
}
```

QuitOnEsc.cs

```
using UnityEngine;

public class QuitOnEsc : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
#if UNITY_EDITOR
            Debug.Log("ESC pressed: would quit app (Editor detected)."
);
            UnityEditor.EditorApplication.isPlaying = false; // stop p
laymode
#else
            Debug.Log("ESC pressed: quitting application.");
```

```
            Application.Quit();
#endif
        }
    }
}
```

Optional Extensions

- Replace cylinder with a windmill and use Rotator for blades.
- Add pickups using trigger colliders (IsTrigger) and destroy them on collect.
- Add a simple menu scene and load Playground via SceneManager.