UNIT — III (08 Hours)

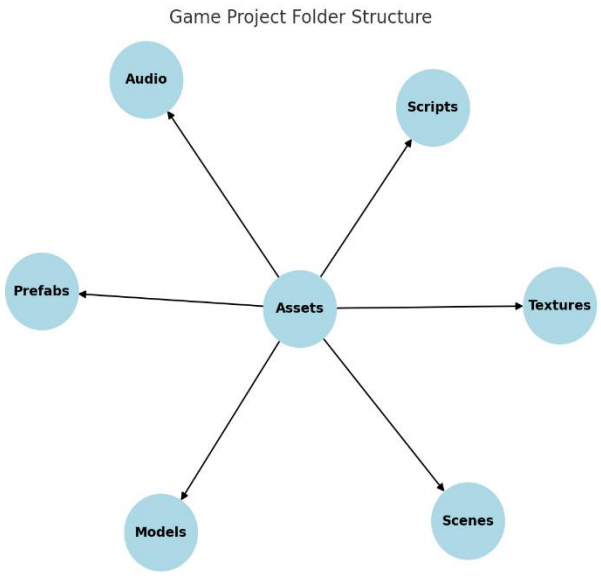
GETTING STARTED WITH GAME DEVELOPMENT

Create Folders — Importing Textures and Meshes — Configuring Meshes — Planning and Configuring Textures — Building Sprites — Importing Audio — Create Prefabs — Scene Building — Lighting and Lightmapping — Building a Navigation Mesh.

1. Creating Folders

Before importing or creating assets, organizing your project is critical.

- **Why Organization Matters?**
  - Large games can have **thousands of assets** (textures, models, sounds).
  - Without structure, debugging and asset replacement is difficult.
  - Industry standard: Clear naming conventions + categorized folders.
- **Standard Folder Setup in Unity**:
- Assets/
- ├— Audio
- ├— Materials
- ├— Meshes
- ├— Prefabs
- ├— Scenes
- ├— Scripts
- ├— Sprites
- └— Textures



- **Best Practices**:
  - Use **camelCase or PascalCase** for naming (e.g., PlayerController.cs).
  - Avoid spaces in filenames (use _ or -).
  - Group assets logically (e.g., Assets/Enemies/Zombie/).

👉 Example: In a racing game → Assets/Cars/SportsCar/ can contain model, textures, sounds, and prefab of that specific car.

2. Importing Textures and Meshes

Assets created in external software are brought into the engine.

- **Textures**:
    - **File types**: PNG (supports transparency), JPEG (smaller size, no transparency), TGA (used for high-quality textures).
    - **Settings in Unity**:
        - Max Size → limit resolution (e.g., 512, 1024, 2048).
        - Compression → balances quality vs. memory usage.
        - Filter Mode → Point (pixel-art), Bilinear, Trilinear (smooth textures).
- **Meshes (3D Models)**:
    - **File types**: FBX (preferred), OBJ (simple geometry), DAE.
    - Import options:
        - **Scale Factor** (ensure same unit scale as engine).
        - **Normals** (Smooth vs Flat shading).
        - **Materials** (imported along with mesh).

👉 Example: Importing a **tree mesh** made in Blender into Unity, with bark texture and leaf texture separately applied.

3. Configuring Meshes

After importing, meshes must be optimized for performance.

- **Mesh Collider**:
    - Mesh Collider = matches exact shape → expensive.
    - Use **BoxCollider / CapsuleCollider** instead when possible.
- **Pivot Points**:
    - Determines **rotation/positioning**.
    - Example: A door pivot should be on its hinge, not in the center.
- **Level of Detail (LOD)**:
    - Different mesh versions (high → medium → low poly).
    - Automatically switches based on camera distance.
- **UV Mapping**:
    - Ensures textures wrap correctly.

- Example: A cube may have one texture stretched across all sides or unique textures for each face.

👉 Best Practice: Always test imported mesh with colliders and materials before using in gameplay.

4. Planning and Configuring Textures

Textures give surfaces realism or style.

- **Planning Stage**:
  - Decide **style**: photorealistic, stylized, pixel-art.
  - Optimize **memory usage**: avoid oversized textures (e.g., 8K for a tiny object).
  - Use tiling textures for repeating surfaces (walls, roads).
- **Common Texture Maps**:
  - **Albedo (Base Color)** → visible color of object.
  - **Normal Map** → fakes depth (e.g., stone cracks without extra geometry).
  - **Specular / Metallic** → controls shininess.
  - **Roughness** → defines surface smoothness.
  - **Ambient Occlusion (AO)** → adds shadow in crevices.

👉 Example: A **brick wall**:

- Albedo (brick pattern),
- Normal map (3D depth),
- AO (shadows between bricks).

5. Building Sprites

Used in 2D games or for UI.

- **Sprite Basics**:
  - Imported as PNG (with transparency).
  - Set Texture Type = Sprite (2D/UI) in Unity.
- **Sprite Sheets**:
  - Multiple frames packed in one image.
  - Use **Sprite Editor** to slice into frames.
- **Animations**:
  - Combine sliced sprites into **Animator**.

- Example: Idle → Walk → Run → Jump.

👉 Example: A 2D character running → sprite sheet of 8 frames stitched into a looping animation.

6. Importing Audio

Sound creates immersion.

- **File Types**:
  - WAV → high quality, large size.
  - MP3/OGG → compressed, small size.
- **Audio in Unity**:
  - **Audio Source** → attached to object (e.g., footsteps sound on player).
  - **Audio Listener** → usually on the Main Camera (acts as "ears").
- **Audio Usage**:
  - Background music (loop).
  - Environmental sounds (birds, wind).
  - One-shot SFX (gunshot, coin pickup).

👉 Example: Player collects a coin → coin prefab triggers "coin_pickup.wav" once.

7. Creating Prefabs

Prefabs = reusable templates.

- **Why Prefabs?**
  - Update once → changes apply everywhere.
  - Saves time in large projects.
  - Allows modular level design.
- **Examples**:
  - **Enemy prefab** → includes AI, mesh, collider.
  - **Collectible prefab** → includes mesh, rotation script, sound.

👉 Example: If you create 100 coins, making a **coin prefab** lets you update material/sound later for all coins at once.

8. Scene Building

A Scene = Game Level.

- **Steps**:
    - Add ground (terrain or plane).
    - Place environment (trees, buildings).
    - Add lighting + skybox.
    - Place gameplay elements (enemies, pickups, objectives).
    - Set **Player spawn point** and camera.
- **Scene Composition**:
    - Balance between empty space and detail.
    - Use occlusion culling to hide unseen objects for optimization.

👉 Example: A forest scene with player at center, enemies patrolling paths, exit gate as objective.

9. Lighting and Lightmapping

Lighting affects both visuals and performance.

- **Types of Lights**:
    - **Directional Light** → sunlight/moonlight (infinite).
    - **Point Light** → radiates in all directions (bulb, torch).
    - **Spot Light** → cone shape (flashlight, lamp).
    - **Ambient Light** → global brightness.
- **Real-time vs Baked Lighting**:
    - **Real-time** → dynamic, changes with objects. Expensive.
    - **Baked (Lightmapping)** → precomputed, great for static objects.

👉 Example: A dungeon with **baked torch lighting** for walls, but a **real-time light** for moving player's torch.

10. Building a Navigation Mesh (NavMesh)

NavMesh = AI movement system.

- **Workflow**:
    1. Mark terrain and environment as **static**.
    2. Bake NavMesh (defines walkable areas).

3. Add **NavMeshAgent** to AI characters.
4. Control movement through AI script (agent.SetDestination(target.position);).

- **Features**:
  - Obstacle avoidance.
  - Pathfinding around static objects.
  - Supports dynamic obstacles with **NavMeshObstacle**.

👉 Example: Enemy AI patrols area but when player enters range, AI uses NavMesh path to chase player.