UNIT — IV EVENT HANDLING & PLAYER CONTROLLER

Event Handling — Notifications Manager — Send Message and Broadcast Message — Character Controllers and the First-Person Controller — Beginning the Universal First Person Controller — Handling Cash Collection — Life and Death: Getting Started. need detailed lecture notes with examples

1. EVENT HANDLING

Meaning

Event Handling is the process of **detecting and responding to events** that occur during gameplay.
An *event* is any significant action or occurrence recognized by the program — for example:

- The player pressing a key or clicking the mouse.
- Two game objects colliding.
- The player picking up an item.
- The timer reaching zero.

Unity uses an event-driven system where **functions are automatically called** when certain events occur.
This makes the game interactive and dynamic.

Types of Events in Games

1. **Input Events:**
   Triggered by the player using input devices like the keyboard, mouse, or controller.
   Example: When the player presses the **Spacebar**, the character jumps.
2. **Collision Events:**
   Occur when two game objects come into contact or overlap.
   Example: The player touches an enemy and loses health.
3. **Trigger Events:**
   Used for detecting when objects enter certain invisible zones or "triggers."
   Example: The player enters a checkpoint area, and the next level unlocks.
4. **Custom Events:**
   User-defined events created by the developer to represent specific game situations.
   Example: A "CoinCollected" event is triggered whenever a coin is picked up.

Importance of Event Handling

- Makes the game responsive to player actions.
- Connects different parts of the game logically (e.g., collecting an item updates the score).
- Simplifies interaction between game objects.
- Reduces complexity by avoiding direct dependencies between scripts.

Example (Conceptual):

When a player's character collides with a coin:

1. A **collision event** is triggered.
2. The **coin** sends a message to the **score manager**.
3. The **score manager** updates the score.
4. The **coin** disappears from the scene.

This is a simple event chain showing how Unity handles actions dynamically.

2. NOTIFICATIONS MANAGER

Meaning

A **Notifications Manager** (or Event Manager) is a centralized system used to **broadcast important events** to different parts of the game.
It follows the **Observer Pattern**, where:

- One object **raises (sends)** an event (called the *publisher*).
- Other objects **listen (subscribe)** to that event (called the *subscribers*).

This system allows multiple game components to respond to one event without directly referencing each other.

Advantages

- Reduces dependency between game objects.
- Makes code and event communication easier to manage.
- Helps in organizing global game actions like:
    - Updating scores,
    - Displaying notifications,
    - Managing level completion.

Conceptual Example

Imagine a game with coins, a score display, and background music:

1. When the **player collects a coin**, the coin sends a **notification** saying "Coin Collected."
2. The **Score Manager** listens for this notification and increases the score.
3. The **UI Manager** listens and updates the score display.
4. The **Sound Manager** plays a coin collection sound.

All these happen **simultaneously**, because they are all "subscribed" to the same event.

3. SEND MESSAGE & BROADCAST MESSAGE

Send Message

- Used to **send a message (or function call)** from one object to another specific GameObject.
- Only the **target object** receives the message.

Example (Conceptual):
When an enemy hits the player:

- The enemy sends a message called **"TakeDamage"** to the player.
- The player receives it and decreases its health.
- Only that player object is affected.

Broadcast Message

- Sends a message to **all child objects** of a GameObject in the hierarchy.
- Used when several objects need to perform the same action together.

Example (Conceptual):
When the player respawns:

- A "Reset" message is broadcast to all components attached to the player, such as health, position, and animations.
- Every component resets itself accordingly.

Comparison Table

Feature	SendMessage	BroadcastMessage
Target	A single GameObject	GameObject and all its children
Communication Type	One-to-one	One-to-many
Typical Use	Damage or update a single object	Reset or update a whole group of objects

4. CHARACTER CONTROLLERS AND FIRST-PERSON CONTROLLER

Character Controller

A **Character Controller** is a Unity component designed for controlling player movement without using rigidbody physics.
It simplifies movement like walking, running, and jumping.

Unlike rigidbody movement (which relies on physical forces), a Character Controller moves the player **directly** based on input values.

Features

- Handles movement and collision automatically.
- Works smoothly on uneven terrain.
- Prevents unwanted physical reactions (e.g., bouncing).

First-Person Controller (FPC)

The **First-Person Controller** allows the player to move and look around **from the player's own viewpoint** — like in shooter or adventure games.

It typically includes:

1. **Movement Control:** Forward, backward, left, right.
2. **Camera Rotation:** Controlled by mouse movement.
3. **Jumping:** Controlled by a jump key.
4. **Gravity Handling:** Keeps the player on the ground.

Conceptual Example

In a first-person game:

- The player uses **W, A, S, D** keys to move.
- The **mouse** controls where the player looks.
- When the player presses **Space**, the character jumps.
- The camera is attached to the player's head position, creating a realistic first-person view.

Advantages of Using a First-Person Controller

- Provides immersive gameplay.
- Useful in action, shooting, and exploration games.
- Simplifies handling of player movement logic.

5. HANDLING CASH COLLECTION

Meaning

Cash or Coin collection is a common game mechanic that rewards the player for achieving certain objectives.
It is an example of **event-based gameplay**, where an event (coin collection) triggers other updates (score increase, sound effect, visual feedback).

Process (Conceptually)

1. The player collides with a cash object (event occurs).
2. The game detects this and:
    - Removes the cash object from the scene.
    - Adds a certain amount to the player's total cash or score.
    - Plays a collection sound or animation.
3. The updated cash value is displayed on the screen.

Example Scenario

In a racing game:

- The player drives through a coin.
- The coin disappears with a sparkling effect.
- A "+100" appears on the screen.
- The total score increases, and a coin sound plays.

This entire sequence represents an event-driven interaction.

Purpose

- Encourages player exploration.
- Tracks progress and rewards.
- Connects gameplay to score or resource management systems.

6. LIFE AND DEATH: GETTING STARTED

Meaning

In most games, players or characters have a **life system** (or health system).
It determines how many hits or damages a player can take before "dying" or losing the game.

Components of Life System

1. **Maximum Health:**
   The total life points available (e.g., 100 HP).
2. **Current Health:**
   The remaining life points after damage.
3. **Damage System:**
   When an enemy attacks, the player's current health decreases.
4. **Death Condition:**
   When health reaches zero, the player "dies," leading to:
    - Game Over screen.
    - Respawn.
    - Restarting the level.

Conceptual Example

- The player starts with 100 health points.
- Each time an enemy hits the player, 20 points are lost.
- When health becomes 0:
    - The player falls to the ground.
    - The "Game Over" message appears.
    - The level restarts after a few seconds.

Importance of Life and Death Mechanics

- Adds challenge and realism.
- Helps manage game flow and difficulty.
- Allows for strategic play and progression.

7. SUMMARY OF UNIT

Concept	Description	Real-World Example
Event Handling	Responding to user or system actions	Player presses jump key → Character jumps
Notifications Manager	Broadcasting information across the game	Updating score display when a coin is collected
Send Message	One object sends a signal to another	Enemy tells Player to "Take Damage"
Broadcast Message	Send message to multiple child objects	Player respawn → All components reset
Character Controller	Simplified system for player movement	FPS movement in Unity
First-Person Controller	Player sees through their character's eyes	Shooting or exploration games
Cash Collection	Reward mechanism in games	Player collects coins or money
Life and Death System	Manages health and game-over conditions	Player loses lives when hit

8. CONCLUSION

This unit teaches how to make games **interactive and responsive** using event-based programming. Through proper event handling, notifications, and controller systems, developers can:

- Create smooth player movement.
- Design effective reward and health systems.
- Ensure objects in the game world communicate efficiently.

Ultimately, **Event Handling and Player Controllers** are the foundation of **player interaction and game logic** in Unity.