Lab 6

1.

1. **NetworkVariable<Vector3>**

Use Case: Synchronises the player's/objects position across the network.

Example: In a multiplayer game, use **NetworkVariable<Vector3>** to keep each players positions the same for every client in the network

2. **NetworkVariable<int>**

Use Case: To track and sync game values like health, score, or ammo count.

Example: Use **NetworkVariable<int>** to update and share a player's health with all clients when it changes due to taking damage.

2.

-**ServerRpc**
Purpose: Marks a method to be called from a client, but executed on the server.
Use Case: When a client wants to inform the server about an action, such as shooting a weapon or interacting with an object.

-**ClientRpc**
Purpose: Marks a method to be called from the server, but executed on clients.
Use Case: When the server needs to update all clients, such as triggering a visual effect or animation.

-**Rpc(SendTo.Server)**
Purpose: Used with Unity's source-generated RPC system as an alternative way to define a ServerRpc.

-**Rpc(SendTo.Clients)**
Purpose: Similar to [ClientRpc], but using the newer source generator approach to define RPCs for better performance.

-**NetworkVariable**
Purpose: Used to define a variable that should be automatically synchronized between the server and clients.
Use Case: Useful for syncing player stats like health, ammo, or score.

3.

ServerRpc :

-Role: Called by clients, executed on the server.

-Purpose: Used when a client wants to notify the server about an event or request an action.

ClientRpc :

-Role: Called by the server, executed on clients.

-Purpose: Used when the server wants to inform or trigger something on all or specific clients.

Example:
A player opens a door in a multiplayer game.

Process:
Client triggers the interaction (pressing 'E' near a door).

Client calls a ServerRpc to request opening the door.

Server responds with a ClientRpc to play the door-opening animation on all clients.

This ensures that the game logic is secure (runs on the server), while the experience is synchronized across all players.

4. **Purpose of Network Manager** :

1. Manages Network State : Controls whether the game is running as a server, client, or host.
2. Spawns and Tracks Networked Objects : Automatically handles the spawning and synchronization of NetworkObjects across the network.
3. Handles Connection and Disconnection : Manages client connections, disconnections, and handles timeouts or network errors.
4. Singleton Access Point : Provides a global way to access network features using NetworkManager.Singleton.
5. Custom Messaging and RPC Routing : Coordinates RPC calls and network message flow between server and clients.

A network manager simplifies the setup and control of multiplayer games by handling connections, object management, and communication between clients and the server.