My little game is a small-scale deathmatch where players can engage in combat, eliminate each other, and respawn after a short delay. The core mechanics include movement, shooting, reloading, jumping, running, and emoting. Players use the following controls:

- W, A, S, D for movement  
- Space for jumping  
- LShift for running  
- LMB for shooting  
- R for reloading (with a 30-round magazine; automatic reload when the magazine is empty)  
- H for emoting (a visual emote that other players can see)  
  
The game is designed to be fast-paced and competitive, with a focus on real-time multiplayer interactions. All critical operations, such as health management and shooting, are handled on the server to ensure fairness and prevent cheating.

It is developed using Unity's Netcode for GameObjects and follows a client-hosted model. Player movement is synchronized across all clients using NetworkTransform. This ensures that all players see each other's movements in real-time.  
Bullets are instantiated on the server and synchronized across clients using NetworkObject&NetworkTransform. Each bullet is owned by the player who fired it, and damage is calculated server-side to prevent cheating.  
Player health is managed using a NetworkVariable, which is synchronized across all clients. When a player's health reaches zero, they are temporarily disabled and respawn after a delay.  
Players can trigger an emote by pressing the H key. The emote is synchronized across all clients using a NetworkVariable and is displayed above the player's head for a short duration.  
In my PlayerNetworkHealth.cs script, I used NetworkVariableReadPermission.Everyone and NetworkVariableWritePermission.Server on the '\_healthVar' and 'isEmojiVisible' variables to make sure they are readable for every client but they can only get updated from server to prevent cheating. For example, a player's health value is only updated by the server. This prevents clients from manipulating the health value and keeps the game fair.  
Also in my PlayerNetworkHealth.cs i used [ServerRpc(RequireOwnership = false)] on TakeDamageServerRpc and ToggleEmojiServerRpc to make sure that the method can be called not only by the player's own object but also by another client or server. For example, when a player deals damage to another player, that damage is processed by the server.  
In this scenario, when a player dies, this information is transmitted to all clients and the player's visibility is updated.  
By using ClientRpc and NetworkVariableReadPermission.Everyone, I ensured that all clients saw the same data. This allowed players to see and interact with each other properly.  
One of the biggest challenges was ensuring that all game objects (Bullets, players, and emotes, etc.) were properly synchronized across clients. This required careful management of NetworkObject and NetworkVariable components.

One of the most challange was ensuring that players could not perform actions like shooting while dead was a challenge. Additionally, respawning players needed to be fully functional and synchronized. And it was not easy to make the connection between the server and the client, I tried many times with debugs to find the right method.  
Although I did not go into much of a detail, I tried to optimize performance by making minor changes to network operations such as tick rate and max packet queue size. Debugging multiplayer games is significantly more complex than single player games. I learned to use logs and debug tools effectively to identify and resolve synchronization issues. Even though I haven't been able to add all the features I wanted yet, I am proud of the final product. I look forward to applying the lessons learned from this project to future multiplayer games.