**Lab 9 – Multiplayer in Unity**

(a) What is Docker? How is containerization used in game development?

Its a platform that uses host level OS virtualization to deliver software in packages called containers. These containers bundle the application code with all its dependencies, libraries, and configuration files, ensuring it runs identically across all different computing environments to maintain reliablity and not produce unexpected results or "it runs on my machine" issues.

Game development example of using docker:

If we've a multiplayer online game with a backend service that manages player matchmaking, game sessions, or leaderboards. Developers working on different parts of the game like the physics or UI need a consistent development and test environment that remains constant.

Each service can run inside its own Docker container.

The dev team can spin up exact replicas of the game backend with one command, without worrying about dependency issues or environment mismatches.

The game servers can be deployed as containers in cloud environment for testing or during production.

(b) Role of Orchestration Tools like Kubernetes in Game Server Management

Kubernetes automates the deployment, scaling, and management of container applications.

Game Server Infrastructure Example:

In online games like mmo's or battle royales, game servers need to be dynamically spun up and down based on player demand.

How Kubernetes Helps:

Automated Deployment: Automatically deploys new game server pods when updates are released and roles them so theres no downtime.

Scaling: Monitors load and automatically scales game servers up/down based on player traffic.

Load Balancing: Distributes traffic to available servers for optimal performance.

Self-Healing: Restarts failed game server instances.

(c) Three Primary Cloud Service Models in Game Development

Infrastructure as a Service (IaaS):

Provides virtualized computing resources over the internet.

Example in Game Dev: Running dedicated game servers on AWS EC2 or Google Compute Engine.

Use Case: Full control over the environment for custom networking or performance tuning.

Platform as a Service (PaaS):

Provides a platform allowing developers to build, run, and manage applications without managing the underlying infrastructure.

Example: Using Azure App Service or Google App Engine to host a game's backend API such as user authentication, leaderboards.

Use Case: Focus on writing game logic without worrying about lower level tech details.

Software as a Service (SaaS):

Delivers software over the internet on a subscription basis.

Example: Using PlayFab a backend as a service platform by Microsoft for user analytics, matchmaking, and cloud saves.

Use Case: Quickly integrate robust backend features without building them from scratch.

(d) Hole-Punching Technique in Port Restricted NAT Devices

What is NAT Hole Punching?

Hole punching is a method used in networking to establish a direct connection between two devices behind NAT.

Problem:

NATs don’t allow unsolicited incoming connections, which prevents peer to peer connections.

Solution - Hole Punching:

Both clients (A and B) behind NAT connect to a mutual server (S) with a public IP.

S learns the external IP and port of both clients.

S shares this info with the clients.

A and B send UDP packets to each others external IP/port.

NATs see the outgoing traffic and allow responses from the corresponding IP/port, creating a "hole" and creating a direct connection to one another.