**Concept of Distributed Computing Using**

**Multiplayer Game**

**PROJECT REPORT**

Submitted for the course: **Parallel and Distributed Computing (CSE 4001)**

SLOT – D1

By

**Team Members:**

**RAVI VOLETI - 15BCE0082**

**RONIT CHAUDHURI - 15BCE0054**

**ANSHUL HEDAU - 15BCE2079**

**SOURAV KUMAR DASH – 15BCE0320**

**Submitted to:**

Prof. **Manoov R**

**B.Tech.** in

**Computer Science and Engineering**

**School of Computing Science & Engineering (SCOPE)**

**Acknowledgement**

The outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all along the completion of this project work.

We express our sincere gratitude to Prof. **Manoov R** (School of Computing Science & Engineering) Associate Professor, VIT University, Vellore, for giving us an opportunity to do the project work on **Concept of Distributed Computing Using Multiplayer Game**. We are extremely grateful to him for his constant support and guidance.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all our friends and teachers, which helped us in successfully completing our project work.

RAVI VOLETI

[15BCE0082 ]

RONIT CHAUDHURI

[15BCE0054 ]

ANSHUL HEDAU

[15BCE2079 ]

SOURAV KUMAR DASH

[15BCE0320 ]

**Aim:**

The aim of the project is to enhance the concept of distributed computing using multiplayer game. We run the multiplayer game in different screens parallel and use client server connection using router to connect the different players.

**Objective:**

The main objectives of the project are as follows:

* Designing a simple game using the software unity, having a suitable map, and dynamically moving players and enemies.
* The simple features of the game include players being able to move in all possible directions, players being able to hit the enemies, and suitable decrease in lifeline on being hit.
* The game can parallely run in many screens on a single machine and also players can connect to each other using distributed system features and player on a single screen.
* Basically the game uses the shared memory where every instance of the game is stored and there is an internal memory for each system in the distributed environment.
* The game has a dual feature such that the player can both play against the enemy as well as the system.
* The player can move in various directions and the game doesn’t limit the number of distributed system.

**Software Requirements**

The following software are required for implementing the project successfully:

* Unity Software – The latest version for game developers.
* Windows
* Mac OS –X
* C++ compiler
* GitHub repository

**Hardware Requirements**

The following hardware is required for implementing the project:

* Router with appropriate bandwidth for connecting machines
* Desktop with suitable processor of at least 1Mhz
* More than one machine to show the connection of players successfully.

**Description about distributed architecture in multiplayer game**

Networked games are rapidly evolving from small 4-8 person, one-time play games to large-scale games involving thousands of participants and persistent game worlds. However, like most Internet applications, current networked games are centralized. Players send control messages to a central server and the server sends (relevant) state updates to all other active players. This design suffers from the well-known robustness and scalability problems of single server designs. For example, complex game-play and AI computation prevent even well provisioned servers from supporting more than several tens of players for first person shooter (FPS) games. Further, client-server game designs often force players to rely on infrastructure provided by the game manufacturers. These infrastructures are sometimes not well provisioned or long-lived; thus, they either provide poor performance or prevent users from playing their game long after their purchase. A distributed design can potentially address the above shortcomings.

However, architecting distributed applications is difficult. The most fundamental challenge in distributing an application is in partitioning the application’s state (e.g., the game world state) and execution (e.g., the logic to simulate player and game AI actions) among the participating nodes. Distributing a networked game is made even more difficult by the performance demands of the real-time game-play. In addition, since the game-play of an individual player translates to updates to the shared state of the game application, there is much more write traffic and write-sharing than most distributed applications. Fortunately, there are two fundamental properties of such games that we can take advantage of in addressing these challenges. First, games tolerate weak consistency in the application state. For example, even current client-server implementations minimize interactive response time by presenting a weakly consistent view of the game world to players. Second, game-play is usually governed by a strict set of rules that make the reads/writes of the shared state highly predictable. For example, most reads and writes of a player occur upon objects which are physically close to the player.

The challenge, then, is to arrive at a scalable and efficient state and logic partitioning that enables reasonably consistent game-play without incurring much latency. This paper presents the design, implementation and evaluation of Colyseus, a novel distributed architecture for interactive multiplayer games designed to achieve the above goals. Our design is based on two key architectural decisions: First, distributed objects in Colyseus follow a single-copy consistency model – i.e., all writes to an object are serialized through exactly one node in the system. This allows low-latency reads and writes at the cost of weak consistency. This mirrors the consistency model of a client-server architecture, albeit on a per-object basis. Secondly, Colyseus utilizes locality and predictability in the movement patterns of players to speculatively pre-fetch objects needed for driving game logic computation. Efficiently locating objects to pre-fetch is achieved using a scalable distributed range-query lookup service.

So this game basically uses the distributed architecture to implement the multiplayer feature on different screens. Here we split the screens and play the game on the same map, so this uses a shared memory having distributed architecture. Proper client server communication between systems has been achieved with the use of the router.

**Networking and Connections**

The networking for this game is managed by a Game Object called “**NetworkManager**” This is basically the main object which handles all networking in the game.

There is also an event manager Game Object called “**EventSystem**” responsible for input handling in the network that includes control over the player.

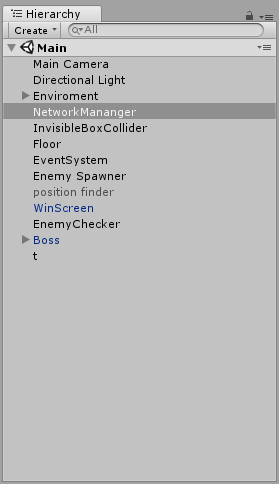


Figure 1 : Position of Network Manager in Hierarchy

The GameObject **“NetworkManager”** has two components called **Network Manager** and **Network Manager HUD.** Thesecomponents are provided by unity developers for the following reasons:

1. **Network Manager**

This handles network information such as transmission of data, determining if an IP address is a client or a server, it is also particularly good for matchmaking with other players if the server is hosted through a cloud.

This also handles spawning of prefabs such as bullets form players or enemies that are present.

Qos channels are present to smoothen the packet transfer and does a cyclic redundancy check

so that packets are handled correctly.

1. **Network Manager HUD**

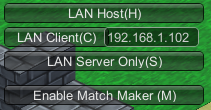


Figure 2 : Network Manager HUD

The above image is the Network Manager HUD. Its main operation is to provide a runtime GUI to the user when he wants to host the game or join it.

**LAN Host** creates an instance of a game which lists its IP address as its primary client called server.

**LAN Client** connects the user over web sockets to the server client (primary host).

**LAN Server Only** is for cloud where the instance of the server is always running and many clients can join later whenever they wish.

**Enable Match Maker** we do not use this feature but it is for a faster match making process.

**Networking for Individual Components**

Individual game components are handlked by **Network Identity** and **Network Transform**.

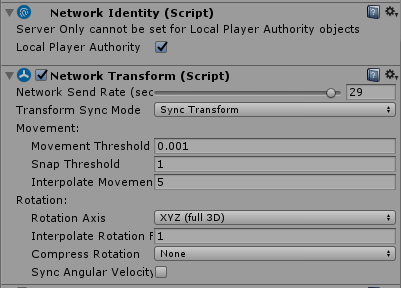


Figure 3 : Network Identity (Script)

**Network identity** determines if the GameObject should be handled either server sided or client sided. It gives a unique identity to each GameObject with this component and allows then to sync via network behaviour script if we wish to do so.

**Network Transform** all the movement is synced via this component. It also handles the rate of packet transfer per model and how each client views other character models in the game.

**Network Manager –**

For the network to recognize and handle players and spawning we use a prefab called network manager which has a network manager script attached to it.

****

Figure 4 : Statistics when Game starts

****

Figure 5 : Statistics when 1 Player Joins the game

****

Figure 6 : Statistics when 2 Player Joins the game

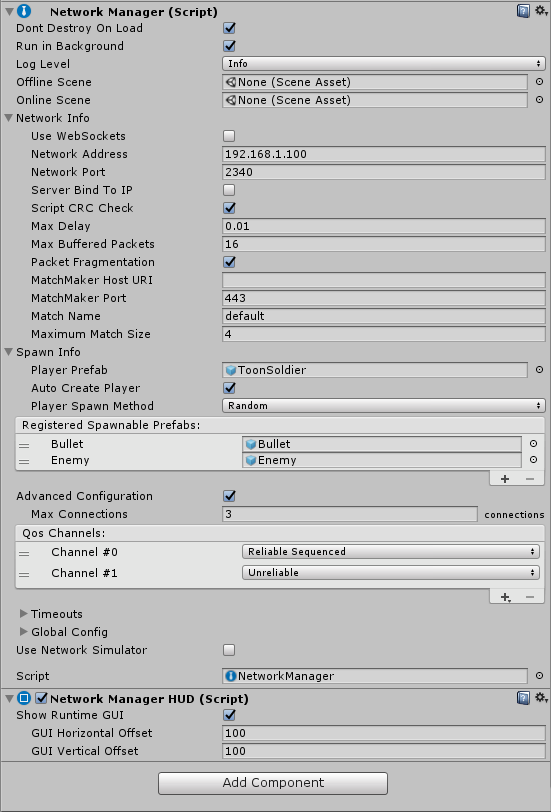


Figure 7 : Network manager (Script)

**Outputs –**

****

Figure 8 : Map of the Game

****

Figure 9 : Starting Scene of Game



Figure 10 : Fighting Sequence of Game



Figure 11 : Image of Hero

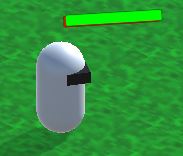


Figure 12 : Image of Enemy

**Link of the project**

<https://github.com/xelese/PDC-Project>

**Download link**

<https://drive.google.com/drive/folders/0B1ly-qt-vJNAY2F1MC1OTTlvZlE?usp=sharing>

**Conclusion**

The project is almost complete and proper display of screenshots and video for demonstration of the game will be uploaded for the final review.

**Future Enhancements**

**References**