UG Report_B6

by Vishnu Tej

Submission date: 30-Jun-2022 04:48PM (UTC+0530)
Submission ID: 1864980207
File name: PreFinalReport.pdf (1.32M)
Word count: 5548
Character count: 27198

Abstract

An online multiplayer game allows users from across the world to connect and play together. Most modern multiplayer games use the client-server architecture. However, in these types of online games, there are issues of latency and performance. To solve these issues, we are building an online scalable low latency multiplayer game using Docker containers. Docker containers can be deployed on a cloud nearest to the client reducing latency and they also enable easy management of game servers.

The game would be implemented in the form of a web application. The basic architecture consists of the game is a client-server model. Any number of clients can join the server and each client has their own game instance in the server, grouped under rooms. A cluster of docker containers are used to run the server. We make use of the phaser game framework to render the game, maintain the game state and we use MongoDB for the database. The Docker containers will be hosted on a cloud.

The game will be able to provide cross-platform support. The game would be a fast-paced combat game and will be able to establish real time communication between the user gameplay session and the server. The game will also be able to allow users to create private and public rooms for game sessions and the game server will run on docker containers.

Chapter 1

Introduction

1.1 Background Study

A multiplayer game is one in which more than one person can play in the same game environment at the same time, either locally and on the same computing system, locally and on different computing systems via a local area network, or locally and on different computing systems via a wide area network, most commonly the Internet.

The game will be having the following features:

- Orthogonal Top-down view
- Co-op Combat
- Character Customization
- Public and private rooms for game sessions

The types of multiplayer games are:

1. Non-networked Games: These are predominantly 2-player games played on a single device with multiple controllers.
2. Networked Games: These allow users from different devices to connect to each other and play the games. It has following two categories:

a. Local Multiplayer: They are always played on the same local network.

b. Online Multiplayer: They Can be played through any device across the Internet.

The game would be an online multiplayer game where players are not restricted to the same network. This can be implemented in two ways as follows:

1. Peer-to-Peer: In these types of games, one of the players become the authority and all other players has to send the data to the authoritative person. The authority updates the game states and sends the updated states to all the other players. Peer-to-Peer type of game has the following disadvantages:
   - The peer who becomes the authority has to have a large bandwidth.
   - If the authoritative peer exits, the game automatically ends for all players.
   - Peer server can change the states, which affects the playing experience for all players.
2. Client Server: In these types of games, a dedicated server is set up. All clients must connect to this server and the server is the authority. This successfully overcomes all the disadvantages of the Peer-to-Peer types of online multiplayer games.

Most modern multiplayer games use the client-server architecture. However, in these types of online games, there are issues of latency and performance. The primary causes of latency are and other performance issues are:

- Bandwidth: Online multiplayer games generally requires a high bandwidth.
- Wired connections have low latency than Wireless connections.
- Internet network hardware.
- Remote server locations and connections used by remote server.
- Internet service provider.
- Network Interference.

The solution to the above problems is to use docker containers. Docker is an open platform for app development, shipping, and running. Docker allows us to decouple our apps from our infrastructure, allowing us to release software swiftly. Docker allows us to bundle and run an application in a container, which is a loosely isolated environment. Because of the isolation and security, we can run multiple containers on a single host at the same time. Containers are lightweight and include everything required to run a programme, eliminating the need to rely on what is currently installed on the host.

Docker provides a container lifecycle management platform and related tooling and thus:

- Containers are used to develop applications and their supporting components.
- The container serves as the distribution and testing unit for applications.
- Deploy applications as containers or orchestrated services into a production environment. It provides the same experience irrespective of the production environment being a local data centre, or a cloud provider, or a hybrid of the two.

In the case of online multiplayer game, docker enables easy management of game servers and allows for deployment on the cloud. This allows an instance of the game to run on the nearest cloud server to the user solving the issues of latency. As there are multiple docker containers running the server, the performance is boosted. Further, using docker containers enables support for rolling deployment, autoscaling, monitoring and maintenance of the game server.

1.2 Problem Statement and objectives

Problem Statement: To develop an Online Scalable Low Latency Multiplayer Game Using Docker

Objectives:

- To enable cross-platform support by building a web application.
- To build a fast-paced combat game.
- To establish a two-way interactive communication session between a user's browser and a server in real time.
- To enable users to create private and public rooms for game sessions.
- To containerize the game server and reduce the game server overhead.

1.3 Organization of the Report

This report is organized into six chapters followed by a Bibliography section as follows:

Chapter 1 provides an overview of the project. It covers relevant works that are similar to ours and served as inspirations for creating this project. This chapter also contains the problem statement and the work objectives.

Chapter 2 contains a literature review, in which we detail all of the technical articles that we have read in order to better grasp the concepts behind this project and come up with relevant changes and features to add.

Chapter 3 interprets the system's high-level design. This section also includes the project's architecture, functional requirements, non-functional requirements, and software development process used.

Chapter 4 focuses on the project's implementation phase. It discusses the tools and technologies used in the project, as well as the project's implementation procedure.

The details of the test scenarios we evaluated for testing our application are presented in Chapter 5. It also includes a results section where the work's output is documented.

The conclusion to the project is presented in Chapter 6. It also includes potential future work connected to this project.

Chapter 2

High-level Design

2.1 Software development Methodology

2.1.1 Agile Methodology

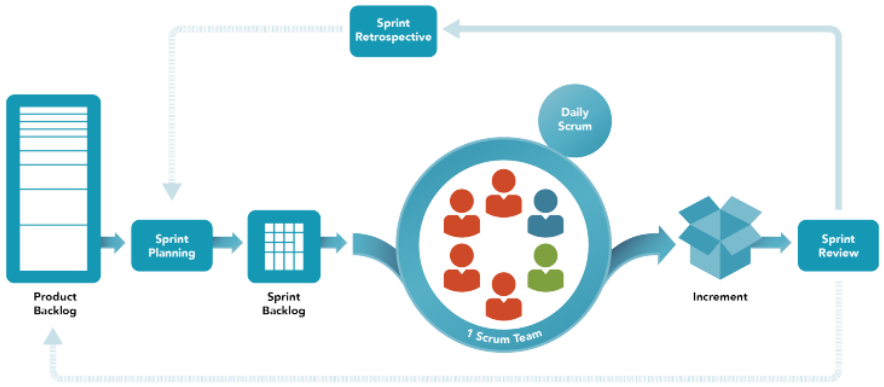


Figure 2.1: Figure showing the Agile software development methodology

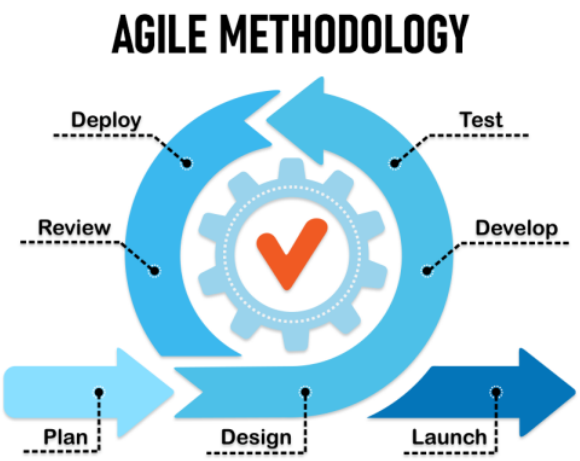


Figure 2.2: Figure showing the steps involved in the Agile model

The Agile methodology is a project management technique that focusses on building a project iteratively by dividing it into small deliverable stages. At each stage, it requires continuous collaboration and progress. When the job begins, agile teams go through the cycle of planning, execution, and evaluation.

Fig. 3.1 shows the agile methodology. The product backlog is prepared after requirement analysis and is consists of all the features that are to be implemented in the project. A sprint is a brief period of time during which a team works to complete a specific amount of work. A sprint backlog is then prepared from the product backlog which consists of the tasks to be completed in a particular sprint. This process is reffered to as sprint planning.

The following activities happen within a sprint cycle as shown in Fig. 3.2:

- Plan
- Design

- Develop
- Test
- Deploy
- Review
- Launch

The sprint cycles are repeated for each increment of the project.

2.1.2 Why Agile for our project?

We have made use of the Agile methodology for the development of this project as it offers the following advantages over traditional methods:

- Agile enables teams to deliver a prototype and iteratively enhance it with each cycle.
- Agile enables teams and individuals to prioritise features more effectively.
- Agile techniques encourages reactive decisions, shorten development periods, and enable just-in-time request analysis.
- Based on the test results, teams can make immediate course adjustments.

2.2 Architecture

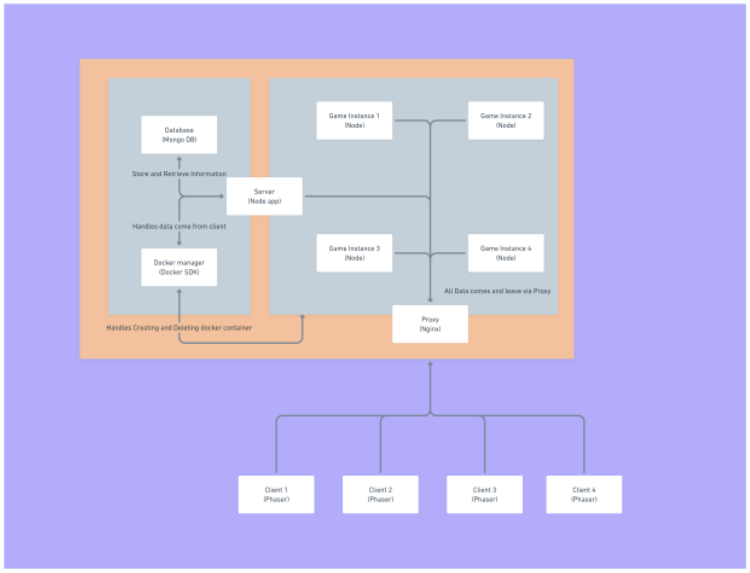


Figure 2.3: Figure showing the architecture for the project

The architecture of a system describes its structure and behaviour. It also comprises a graphical depiction of the ideas, concepts, elements, and components of the system.

The following are the major components of the architecture of our project as shown in Fig. 3.3:

1. **Database:** In this project we make use of MongoDB database. MongoDB is a cross-platform database, classified as a NoSQL database. It uses JSON like documents with optional schema. We use this database primarily to store the state of the game. We store the state for each room (A room represents a group of users playing together) which a group of users connect to. We store the room ID, the members of the room, their characters and respective inventory ie., the weapons, coins collected etc.

2. **Docker Manager:** Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. Docker manager is a tool that runs the docker SDK. It helps us to create and manage the various rooms in the game. Each docker instance will run a single instance of the game. The database and the docker manager constitute the backend of our application.

3. **Server:** The server uses Node.js to connect to the docker containers running the game to the users. Node.js is an open-source, cross-platform backend JavaScript runtime environment that runs on the V8 engine and executes the JavaScript code outside the web browser. It acts as a middleware between backend and frontend. It provides a layer of abstraction between the database and docker manager. It also manages the user authentication.

4. **Game Instance:** It is a docker container that is created and managed by the docker manager. It acts as the frontend. Each game instance runs a game server within a docker container which the players can join. It manages the users who have joined, broadcasts the game state to all the users and creates the maps of the game procedurally.

5. **Proxy(Nginx)**: Each user connecting to the application must connect to the containers through different ports. But, the number of ports available ports is limited - there are 65535 ports out of which 2000 are meant for special purposes. As a result of which we need something that maps each container to the outside world. The advantage of using docker is that, a user can connect to a docker instance or container using its name. So we use a proxy that maps the external request of the user to the internal network. In our application, the proxy works by mapping a request with subdomain having the name of a docker instance to that particular docker instance. For example, *abc.domain.com* connects to the *abc* container.

6. **Client:** The client program uses the Phaser game engine and it runs on the browser of the user. The specifics of the game are written in TypeScript files. The client obtains the data from the game instance and renders it in the web browser for the user. Any action made by the user, constitutes a

change in the state of the game, will be sent to the game instance which is then broadcasted to the room that the game instance is part of.

2.3 Functional Requirements

A functional requirement in software engineering and systems engineering describes a function of a system or its component, where a function is defined as a specification of behaviour between inputs and outputs.

Calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is expected to perform are some functional requirements. Behavioral requirements explain all of the situations in which the system applies functional requirements, which are documented in use cases.

2.3.1 Scenes

2.3.1.1 BootScene

This class loads all the required assets for the game. It shows a loading image until loading of all the assets is complete.

2.3.1.2 GameScene

This class renders the current game state to the user. It renders the neccessary images, adds the colliders and initializes other players. It obtains all the required data from the server using listeners and diplays it to the users. It uses the listeners from the *Listeners* class

2.3.2 Listeners

This class implements various listeners for the different events that occur in the game. Any module in the application can make use of the listeners implemented in this file. It makes use of sockets to receive events broadcasted by other users and to broadcast events. The following are the important listeners implemented.

i. Socket Events: These events are received through the socket from other players.

- Game Server Connection: This event is triggered when a user joins a room.
- Game Server Disconnection: This event is triggered when a user disconnects from a room.
- Game Server New Player: This event is triggered when a new user joins a room. After the joining the room, the user must obtain the data of all other users in the room, this is achieved by making use of the Game Server New Player event.
- Game Server Player Moved: This event is used to send the position and movements off the player.
- Game Server Get Map: This event is used to receive the map of a level.
- Game Server End Level: This event is triggered when a level is completed and a new level is loaded for all the players.
- Server Game Remove Player: This event is triggered when a user leaves the room. His player is deleted from the game and is broadcasted to all the users.
- Server Game Spawn New Player: This event is triggered when a new user joins a room. When a new user joins the room, all the other users must be notified and the corresponding player must be created. This is achieved by making use of this event.

ii. Game Events: These events occur by the actions of a user and are broadcasted to all the other users.

- Socket Game Spawn New Player: This event is triggered when a new user joins a room. His corresponding player is created and this information is broadcasted to all the other players.
- Game Socket Player Moved: This event is triggered whenever a user moves his player. The updated position is broadcasted to all the other players.

- Game Socket Get Map: Requests the server for the map of the level.
- Game Socket End Level: This event is triggered when a user completes a level. This information is broadcasted to all the other users and the server is requested to generate a new level.

2.3.3 Client

2.3.3.1 Player

This class implements the logic required for handling a player in the game. It keeps track of the main player (representing the current user), obtains the inputs from the user and makes the camera follow the main player with smooth animation.

2.3.3.2 Enemy

This class implements the logic required for handling an enemy in the game. It represents one enemy unit and track the players to follow and attack them.

2.3.4 Server

2.3.4.1 Proxy

We make of Nginx for the proxy. We have implemented it in a way that *subdomain.domain* request is mapped to the *subdomain* docker container.

2.3.4.2 Main Server

The main server is the primary server, which facilitates all the functionalities of the game including OAuth and creation of public and private rooms. It notifies the database and docker manager instance that a new game is to be started.

2.3.4.3 Game Server

This class listens for the socket events and broadcasts the data. It creates the procedural maps using room_generator. The room_generator generates a procedural map for a level. It first initializes a 11*11 matrix, and sets the middle of this matrix as the start room. It uses the Drunkard Walk algorithm to populate the

rest of the matrix until the maximum number of rooms for all the types of rooms are created. Based on that matrix, we create the map with variable width and height. We then generate the walls for each room and the pathway between the rooms as given by the algorithm. We then add the location of the colliders and overlappers. The map converted into JSON format and returned to the client.

2.3.4.4 Docker Manager

The docker manager class handles the creation and deletion of rooms. During the creation of a room, the docker manager allocates the necessary resources to a container and notifies the database.

2.3.4.5 Database

We use the database primarily to store the state of the game. We store the state for each room (A room represents a group of users playing together) which a group of users connect to. We store the room ID, the members of the room, their characters and respective inventory ie., the weapons, coins collected etc.

2.4 Non-Functional Requirements

A non-functional requirement (NFR) is a requirement in systems engineering and requirements engineering that specifies criteria that can be used to judge the performance of a system rather than specific behaviours. They differ from functional requirements, which define precise behaviour or functions. The system design includes a thorough plan for implementing functional requirements. Because non-functional requirements are frequently architecturally significant, the plan for implementing them is outlined in the system architecture.

- **Availability**: The cloud service providers like AWS, Azure or GCP have high uptimes. A high uptime implies a highly available system. We shall make use of GCP for hosting our application, thereby providing high availability.

- **Cost Effectiveness:** One major benifit of making use of cloud-based architecture in this project is the pay-as-you-use model. It provides significant savings for the cost of hosting the game.
- **Maintainlability**: The cloud-based architecture supports rolling deployment. This is a deployment approach in which earlier versions of an application are gradually replaced with new versions of the application by replacing the infrastructure on which the application is executing.
- **Resilience:** It is the ability of a system or an application to provide and maintain acceptable levels of service in the face of challenges to normal operation. Most cloud service providers facilitate container managers which ensure that a new container replaces a faulty one.
- **Resource Constraints:** This game is delivered to the end users in the form of a web application eliminating the need for high end configurations of hardware from the users.

Chapter 3

Implementation

3.1 Tools and Technologies

The following are software tools and technologies that we have employed in this project:

3.1.1 Server Side

- Docker – For containerization
- Docker Manager – For managing the container
- Node – For server scripting
- Express – For server web application framework
- Socket.io – For handling the WebSockets
- Nginx – For reverse proxy
- MongoDB – Database

3.1.2 Client Side

- TypeScript - Add static typing to the language
- Phaser – Game engine

3.1.3 Development

- Docker desktop
- VS Code

3.1.4 Deployment

- Cloud Run: It is used to develop and deploy highly scalable containerized apps on a fully managed serverless platform utilising languages such as Go, Python, Java, Node.js,.NET, and others.
- Container Registry: Container Registry is a centralised location for teams to maintain Docker images, conduct vulnerability assessments, and select who has access to what using fine-grained access control. Existing CI/CD connectors allow us to set up completely automated Docker pipelines for quick feedback.
- Firebase Auth: Firebase Authentication aims to simplify the development of secure authentication systems while improving end-user sign-in and onboarding. It provides a complete identity solution, including email and password accounts, phone authentication, and login to Google, Twitter, Facebook, and GitHub, among other services.

3.1.5 Testing the latency and response times

- BurpSuite: It is an integrated platform and graphical tool for doing web application security testing. We have used this tool for testing the latency and response times of our application.

3.2 Implementation

3.2.1 UML Diagrams

3.2.1.1 Use-Case Diagram

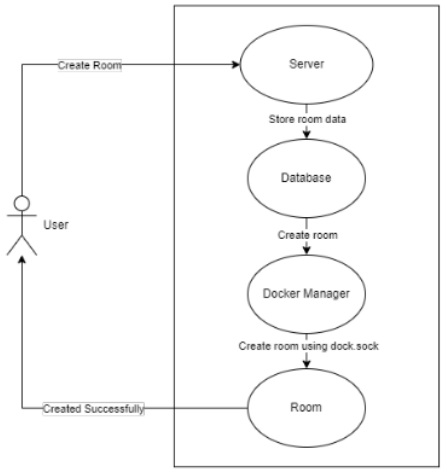


Figure 3.1: Use Case Diagram for creation of a room

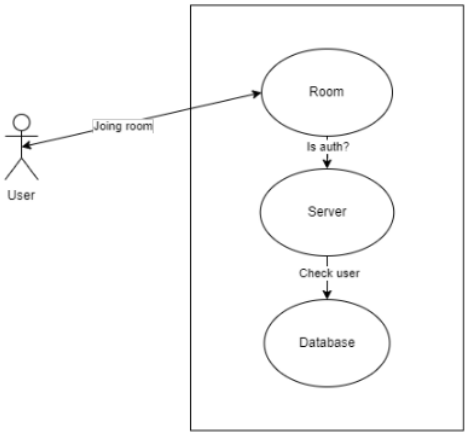


Figure 3.2: Use Case Diagram for a user joining a room

The Fig. 4.1 shows the use case of a user creating a room. The user requests the server to create a room. The server generates a unique identifier for the game sessions and notifies the docker manager to start the game session. The server stores the unique identifier and the docker manager stores the container information in the database. The created container acts as a server for the created game instance i.e., room. This room information is sent to the user upon successful creation.

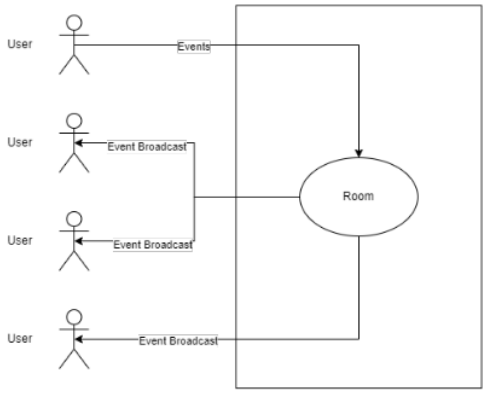


Figure 3.3: Use Case Diagram for an action performed by a player

The Fig. 4.2 shows the use case of a user joining a room. The user is authenticated and the identity is established. To connect to a private room, the user must provide the unique room identifier. Whereas, to connect to connect to a puiblic room, the user is connected to the closest room from a list open rooms. Once a user joins a room, the data is broadcasted to all the existing players in the room.

The Fig. 4.3 shows the use case of an action being performed by a player. The user actions are mapped to the events in the server and these events are broadcasted to other players to notify them about the changes brought about in the game by that action. This is a constant two-way communication between players and server to facilitate a seamless gameplay experience.

3.2.1.2 Sequence Diagram

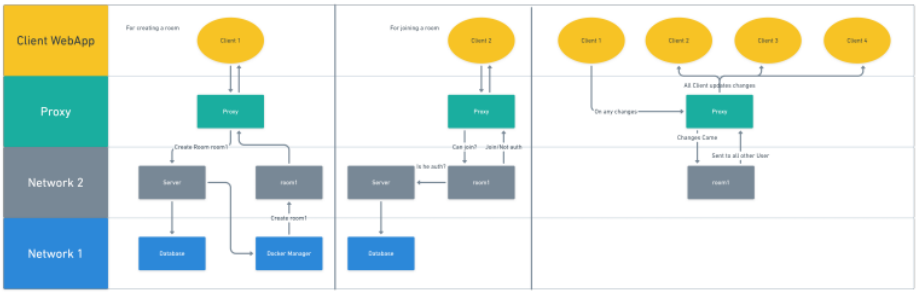


Figure 3.4: Sequence diagram for the project

The sequence diagram for the project is as shown in Fig. 4.4. A sequence diagram chronologically shows the different events that occur throughout the life cycle of the game.

The sequence diagrm shows the following events:

1. **Creation of a room:** A room is created upon the request of the user and the details are stored in the database.
2. **A user joing a room:** The user is authenticated and connected to the room.
3. **The game play:** The game states from one player is consistently broadcasted with the players in the room.

Chapter 4

Testing

This section provides an overview of testing, testing procedure, and test case information. Software testing is a mechanism for ensuring the proper functionality of software as well as determining whether or not the application fits the defined requirements. They verify that the application is bug and defect free. This ensures that the final consumer receives a high-quality product.

4.1 Test Workflow

Testing is done to uncover problems in the product at an early stage, before the product is put into production. Testing can be performed at various levels, such as module and sub-module. The entire product can be tested to guarantee proper module integration and that the product is fault-free from start to finish. There are numerous testing procedures available. These methodologies necessitate distinct strategies and test the numerous components that comprise the application.

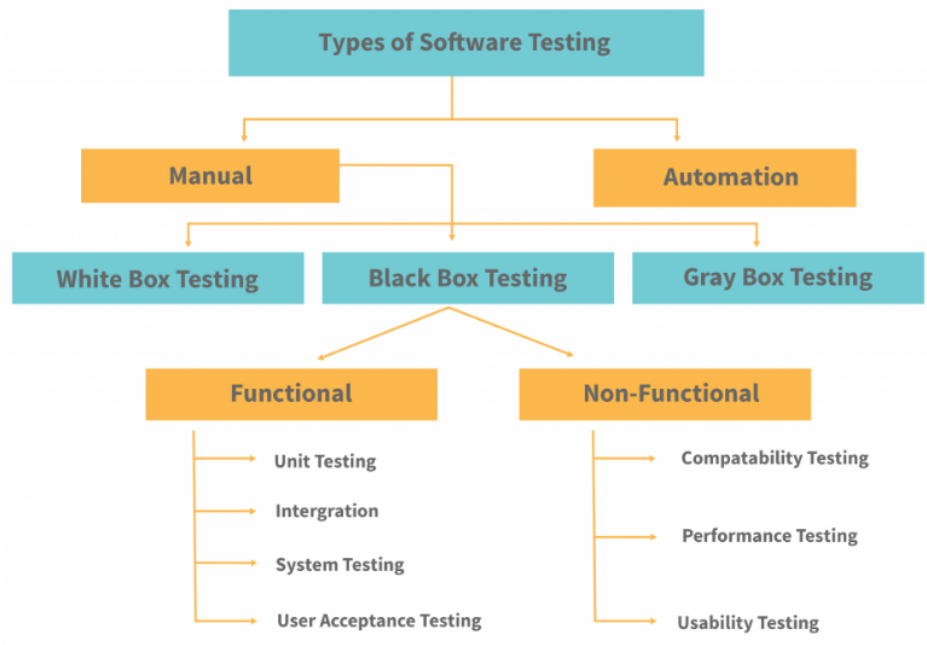


Figure 4.1: Classification of Testing Methodologies

A variety of tests are run on the developed application before it is deployed. Tests are carried out to ensure that the application is carrying out the required duties correctly. A test suite is a collection of tests with a common goal. Fig. 5.1 shows the classification of testing methodologies. The methodologies that are relevent to our work are briefly described in the following sections.

4.1.1 Objectives of Testing

The process of discovering defects in a programme is known as testing. The purpose of the testing is to find errors that aren't immediately apparent to the developer or user but cause problems with the application's workflow. It is a way of detecting flaws or vulnerabilities in a code base in a methodical manner. The main advantage of testing is that it assures that the product fits the end-user criteria before it is

issued for use. It also verifies that it is logically correct. Testing ensures that the application workflow is followed and that it functions as intended. The testing is done from the perspective of the users, thus all possible edge situations will be examined, lowering the danger of the application breaking.

4.1.2 Test Cases

The fundamental unit of the testing paradigm is the test case. They can be used to test a single entity, the functionality of a complete module, or the integration of the modules. To produce a more dependable product, additional test cases must be written to evaluate the application's end-to-end functionality while taking boundary conditions into account. A large number of test cases aids in determining the accuracy and efficacy of a project.

4.1.3 Black-Box Testing

Black box testing is another name for behavioural testing. This type of testing is largely concerned with the functionality of the application. In general, the individual performing black-box testing does not need to understand how the program works internally. This form of testing can handle the following types of errors.

- Errors caused by a missing or improper interface.
- Errors in choice of the data structure or the database used.
- Errors caused by data initialization and data exchange.

4.1.4 White-Box Testing

White-box testing is a type of software testing that examines an application's internal structures or workings rather than its functionality. To design test cases in white-box testing, an internal perspective of the system as well as programming abilities are used. The tester selects inputs to exercise code routes and identify expected outputs. White-box testing can be used in the software testing process at the unit, integration, and system levels. Data flow within a unit, Data flow between units during integration, and Data flow between subsystems during a system-level

test can all be tested. This testing methodology applies in the following situations:

- Ensuring the proper functionalities of all the sub-modules.
- Ensuring consistent behaviour the system even for boundary conditions.

4.1.5 Unit Testing

Unit testing is in charge of testing the application's most basic unit. A unit is a little piece of code that can be thought of as its own programme. It is used to ensure the authenticity of a module by checking for faults. The basic operation of the modules and their sub-modules is validated. Module interactions are also checked to ensure that data is passed between modules as expected. Unit testing is carried out throughout the product development process. Following the completion of product development, the units are tested to ensure that they continue to function as intended.

4.1.6 Integration Testing

Integration testing is the stage of testing in which related sub-modules joined and assessed as a group. Integration testing is used to determine whether a system or component meets the functional requirements. It happens after unit testing but before system testing. Integration testing takes as input unit-tested modules, collects them into bigger aggregates, applies tests described in an integration test plan to those aggregates, and outputs the integrated system suitable for system testing. Top-down or bottom-up integration testing might be used.

4.1.7 Functional Testing

Functional testing is a sort of black-box testing, where the test cases on a software component being tested is base on the specifications of the component. This testing method concentrates on the system's functional components while neglecting the logical features of the application.

4.2 Test Case Details

4.2.1 Test Case ID: 1

Unit to test: User Movement
Test data: Keyboard inputs from the user
Expected Result: The player character movements in the game reflects the user's keyboard input and this movement it is replicated across all the users in the room.
Pass/Fail: Pass

4.2.2 Test Case ID: 2

Unit to test: User Attack
Test data: Spacebar input and mouse pointer position (for aiming) from the user
Expected Result: The player character fires a bullet when the spacebar is keyed in the direction of the mouse pointer and it is replicated across all the users in the room.
Pass/Fail: Pass

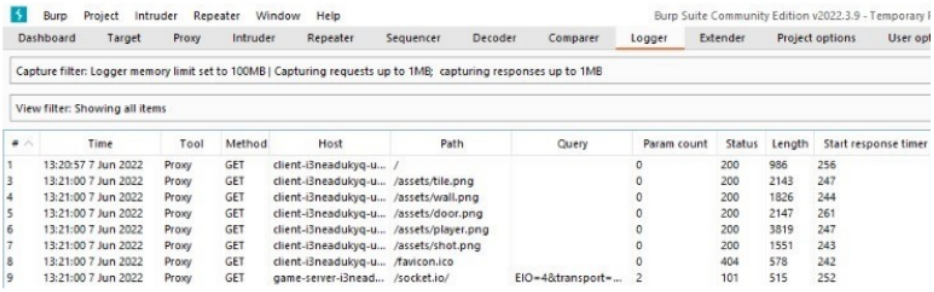
4.2.3 Test Case ID: 3

Unit to test: Enemy Pathfinding
Test data: Movement of the player character
Expected Result: The enemy characters moves towards the closest player character and it is replicated across all the users in the room.
Pass/Fail: Pass

4.2.4 Test Case ID: 4

Unit to test: Generation of the game world:
Test data: Null
Expected Result: A new game world (map) is procedurally created and is broadcasted to all the users.
Pass/Fail: Pass

4.3 Results



#	Time	Tool	Method	Host	Path	Query	Param count	Status	Length	Start response timer
1	13:20:57 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/		0	200	986	256
3	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/assets/tile.png		0	200	2143	247
4	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/assets/wall.png		0	200	1826	244
5	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/assets/door.png		0	200	2147	261
6	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/assets/player.png		0	200	3819	247
7	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/assets/shot.png		0	200	1551	243
8	13:21:00 7 Jun 2022	Proxy	GET	client-i3neadukyq-u...	/favicon.ico		0	404	578	242
9	13:21:00 7 Jun 2022	Proxy	GET	game-server-i3nead...	/socket.io/	EIO=4&transport=...	2	101	515	252

Figure 4.2: Response time test result from BurpSuite

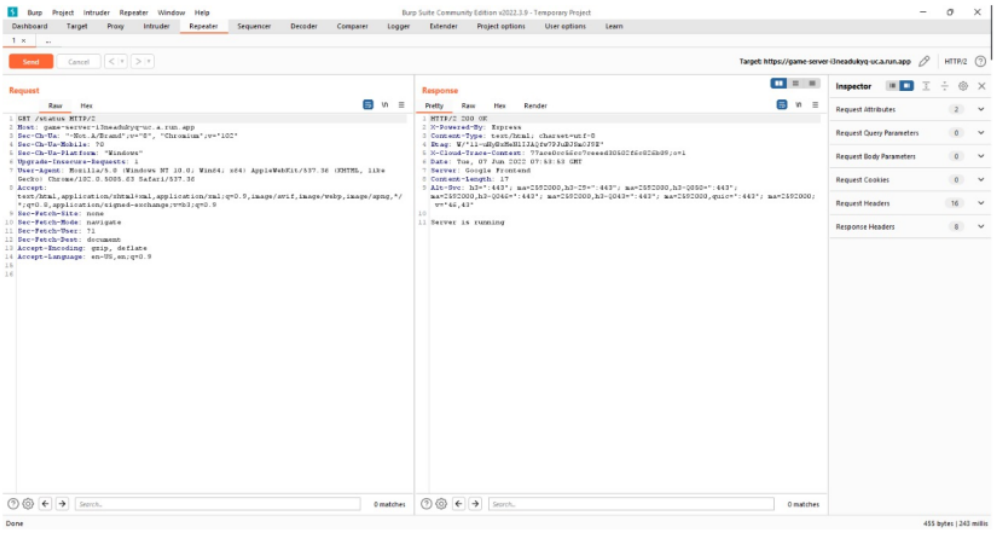


Figure 4.3: Response time for WebSockets from BurpSuite

Fig. 5.2 shows the response time of client instances of the game as seen in BurpSuite. It can be seen that the response time is very low and is in the range of 240 - 250 milliseconds (ms).

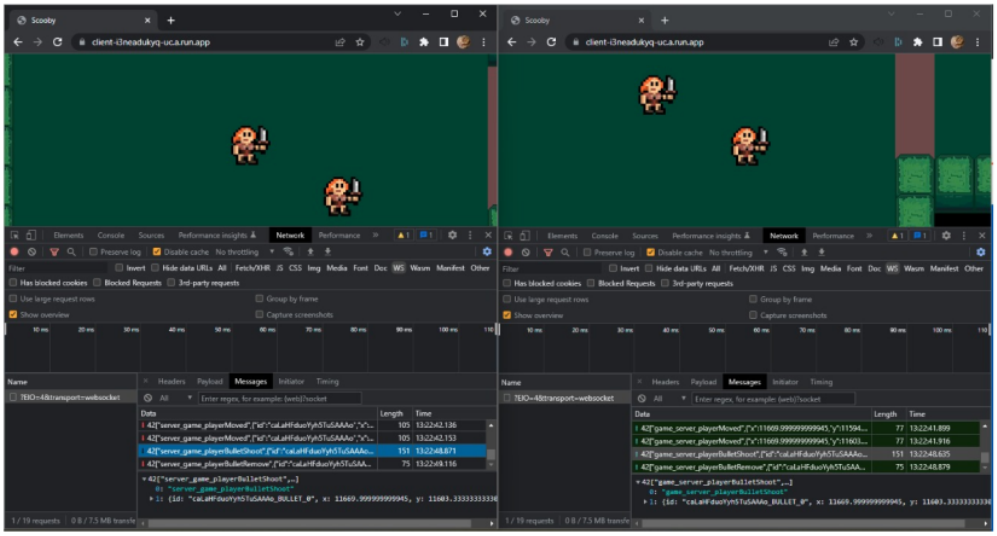


Figure 4.4: Response time for WebSockets from Chrome inspect

Fig. 5.3 shows the response time of the request from the server to the client as seen in BurpSuite. It can be seen that we have achieved a low response time of 243 (ms).

Fig. 5.4 shows the latency of the application as 236 (871 - 635) as seen from the Google Chrome inspect

Fig. 5.5 is a still from the game showing the movement of the player chracter in the room and some bullets being fired from it.

Fig. 5.6 shows the teleportation of the player characters. The still also shows the closure of the room boundary when all the player characters enter.

Fig. 5.7 shows some enemy characters swarming towards the player character.

Figure 4.5: Player Movement and Attack

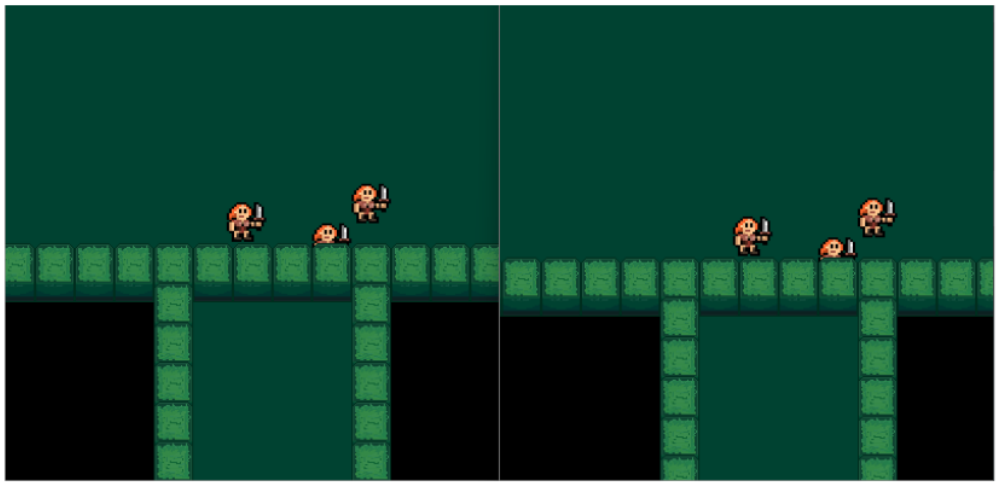


Figure 4.6: Player teleportation and door closure

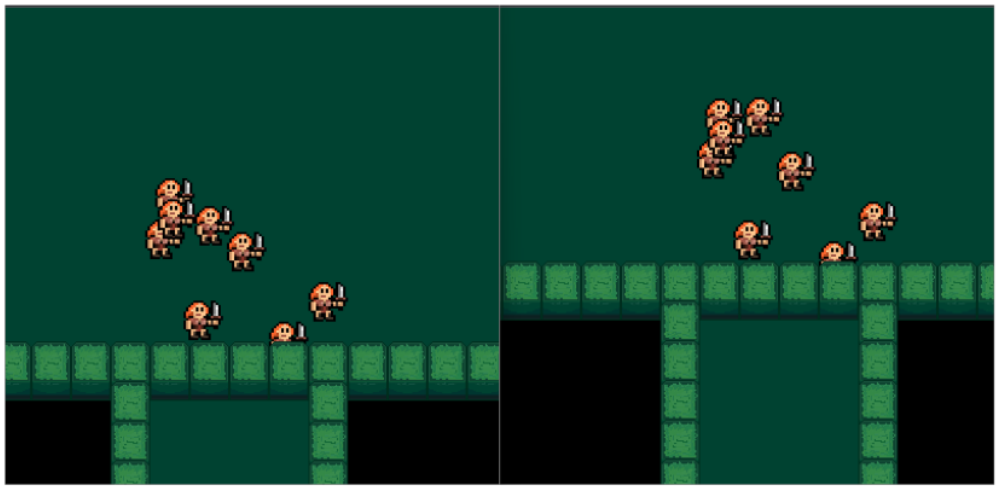


Figure 4.7: Enemy Character movement

Chapter 5

Conclusions and Future Scope

The online multiplayer games constitute a significant portion on the gaming industry. With an active userbase of more than two billion players, the industry is worth around thirty thousand crores. Consequently, it is very important for the games to be highly scalable for meeting the evergrowing demands. To enhance user experience, it is crucial for the games to have low latency and feel more responsive.

We have used docker containers for the game server. The containers allow us to easily manage, monitor and maintain the game server. They allow clound deployment, reduce latency by running an instance of the game on the cloud server nearest to the user, boost server performance by using multiple docker containers to run the server, it also enables support for rolling deployment and autoscaling.

We have built an online multiplayer game that allow users from across the globe to connect and play together. The game would be a 2D multiplayer game that supports features like orthogonal Top-down view, co-op combat, Character customization and public and private rooms for game sessions.

5.1 Future Scope

The following additional enhancements can be implemented in this project for better gameplay experience:

- **Microtransactions:** A microtransaction is a business concept in which consumers can buy virtual goods for small sums of money. Microtransactions are frequently found in free-to-play games, which means there is no fee to download the game, only to purchase the online virtual products. Microtransactions are typically used to unlock certain features or to provide distinctive cosmetic, and occasionally game-changing, additions.

- **Non-Fungible Tokens:** NFT-enabled games are blockchain games that allow users to buy in-game collectibles to use as weapons to complete tasks or to sell to other users within the game and on external NFT marketplaces. Games with NFTs are not only exciting but also possibly rewarding. NFT gaming is growing fast, but more importantly NFT gamers are finding a purpose in their hobby.

- Additional features like global leaderboards, weekly challenges, quests etc., can be added to the game.

UG Report_B6

ORIGINALITY REPORT

15%	5%	2%	14%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Visvesvaraya Technological University, Belagavi Student Paper	2%
2	Submitted to The British College Student Paper	1%
3	Submitted to Siddaganga Institute of Technology Student Paper	1%
4	tuxmachines.org Internet Source	1%
5	Submitted to University of Ulster Student Paper	1%
6	Submitted to Ghana Technology University College Student Paper	1%
7	Submitted to Panipat Institute of Engineering & Technology Student Paper	1%
8	Submitted to University of Greenwich Student Paper	1%

9	wikimili.com Internet Source	1 %
10	Submitted to Kingston University Student Paper	1 %
11	Submitted to CITY College, Affiliated Institute of the University of Sheffield Student Paper	1 %
12	analyticsindiamag.com Internet Source	<1 %
13	Submitted to Midlands State University Student Paper	<1 %
14	Submitted to Universiti Teknologi Petronas Student Paper	<1 %
15	Submitted to Griffth University Student Paper	<1 %
16	documents.mx Internet Source	<1 %
17	Matthew Wong, Ramani Duraiswami. "Shared-Space: Spatial Audio and Video Layouts for Videoconferencing in a Virtual Room", 2021 Immersive and 3D Audio: from Architecture to Automotive (I3DA), 2021 Publication	<1 %
18	Submitted to Spring Branch Independent School District Student Paper	<1 %

19	Submitted to Arab American University - Jenin Student Paper	<1 %
20	Submitted to Liverpool John Moores University Student Paper	<1 %
21	Submitted to University of Bedfordshire Student Paper	<1 %
22	Submitted to Letterkenny Institute of Technology Student Paper	<1 %
23	Submitted to Free University of Bolzano Student Paper	<1 %
24	www.mkcr.cz Internet Source	<1 %
25	Submitted to University of Lincoln Student Paper	<1 %
26	Submitted to Harare Institute of Technology Student Paper	<1 %
27	Submitted to Middle East College of Information Technology Student Paper	<1 %
28	Submitted to City and Islington College, London Student Paper	<1 %
29	eprints.utar.edu.my	

Internet Source

<1%

30 Lee Boonstra. "Chapter 10 Creating Fulfillment Webhooks", Springer Science and Business Media LLC, 2021

Publication

<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		