DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Major Project Pre Final Report - [VIII Sem B.E]

on

"An Online Scalable Low Latency Multiplayer Game Using Docker"

Submitted by

Subramanya G	1SI18CS115
Swaroop K	1SI18CS121
Vishnu Teja S	1SI18CS133
Vishwak Vemuru	1SI18CS135

under the guidance of

Mrs Kavitha M

Assistant Professor

Department of Computer Science and Engineering

Siddaganga Institute of Technology, Tumakuru

(An Autonomous Institute, Affiliated to Visvesvaraya Technological University Belagavi,

Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

B.H Road, Tumakuru-572 103, Karnataka, India.

AY 2021-22

Contents

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's browser and a 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

A multiplayer game is a game 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 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

There are two types of multiplayer games:

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. There are two ways of implementing this:

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 developing, shipping, and running applications. Docker enables us to separate our applications from our infrastructure so we can deliver software quickly. Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow us to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run an application, so we do not have to rely on what is currently installed on the host.

Docker provides tooling and a platform to manage the lifecycle of containers:

- Develop applications and its supporting components using containers.
- The container becomes the unit for distributing and testing applications.
- Deploy applications into a production environment, as a container or an orchestrated service. This works the same whether the production environment is a local data center, 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.

Chapter 2

Literature Survey

Many works and papers were referred and the following things were noted.

Cloud gaming is the future for the gaming industry and it allows users to play games anywhere, anytime and from any device. The paper [1] addresses the main challenges in the cloud gaming namely, resource allocation and ensuring quality of user experience. [1] also elaborates on how to improve the performance of cloud gaming by initializing the cloud gaming package inside Docker containers which will allow the application to be more reliable by secularizing its resource allocation and increasing the overall performance of the cloud gaming system and makes it more reliable while utilizing less resources.

The real-time strategy game StartCraft: Brood War is a commonly used domain for AI research, with a pre-installed collection of AI development tools supporting all the major types of StarCraft bots. [2] presents a dockerized version of StarCraft. The implementation in [2] provides a convenient way to deploy StarCraft AIs on numerous hosts at once and across multiple platforms despite limited OS support of StarCraft. [2] also describes the design of the Docker images and presents a few use cases.

Mobile Edge Computing (MEC) is an emerging network paradigm that provides cloud and IT services at the point of access of the network. Such proximity to the end user translates into ultra-low latency and high bandwidth, while, at the

same time, it alleviates traffic congestion in the network core. Due to the need to run servers on edge nodes, key element of MEC architectures is to ensure server portability and low overhead. [3] proposes a tool that can be used for this purpose - Docker. Docker is a framework that allows easy, fast deployment of Linux containers. [3] focusses on the suitability of Docker in MEC scenarios by quantifying the CPU consumed by Docker when running two different containerized services: multiplayer gaming and video streaming. The results of tests performed by the authors in [3], with varying numbers of clients and servers, yield different results for the two case studies: for the gaming service, the overhead logged by Docker increases only with the number of servers; conversely, for the video streaming case, the overhead is not affected by the number of either clients or servers.

The recent developments in the field of cloud computing allow cloud gaming to provide high-end quality of experience to the users. [4] claims that the fundamental requirement of gaming is to provide maximum quality of gamer experience, however cloud gaming suffers in terms of providing Quality of Experience, because the network transmission of game scenes from cloud game server to gamer device is distant. The author endeavours to minimize latency and increase performance in cloud gaming through this paper. [4] proposes moving the cloud game server to the fog nodes present at the edge network of the player based on Node selection algorithm. [4] also suggests that, to increase the performance of cloud gaming, traditional virtual machine should be replaced by light-weight containers.

HTML5 has provided many developers the opportunity to experiment with the new possibilities for web development. The authors' aim, through [5] is to give an overview of what this means to the game development community. [5] evaluates new HTML5 elements and JavaScript features. It also highlights WebGL, Canvas and WebSockets that have given developers the opportunity to flaunt their creativity by manipulating images, creating 3D environments and providing real-time interaction.

Big Two is an online multiplayer game with imperfect information. In this game, each player plays without knowing the opponent's confidential information. [6]

considers Big Two as an example in elaborating the need for web-based multiplayer games with imperfect information to have real-time communication for handling rapid information changes and the condition of the game state at any time. [6] proposes a new framework for web-based multiplayer games with imperfect information. [6] utilizes an open-source WebSocket, namely Socket.IO, and implements this framework in Big Two as a case study.

Chapter 3

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 real time two-way interactive communication session between the user's browser and a server.
- To enable users to create private and public rooms for game sessions.
- To containerize the game server and reduce the game server overhead.

Chapter 4

Project Design

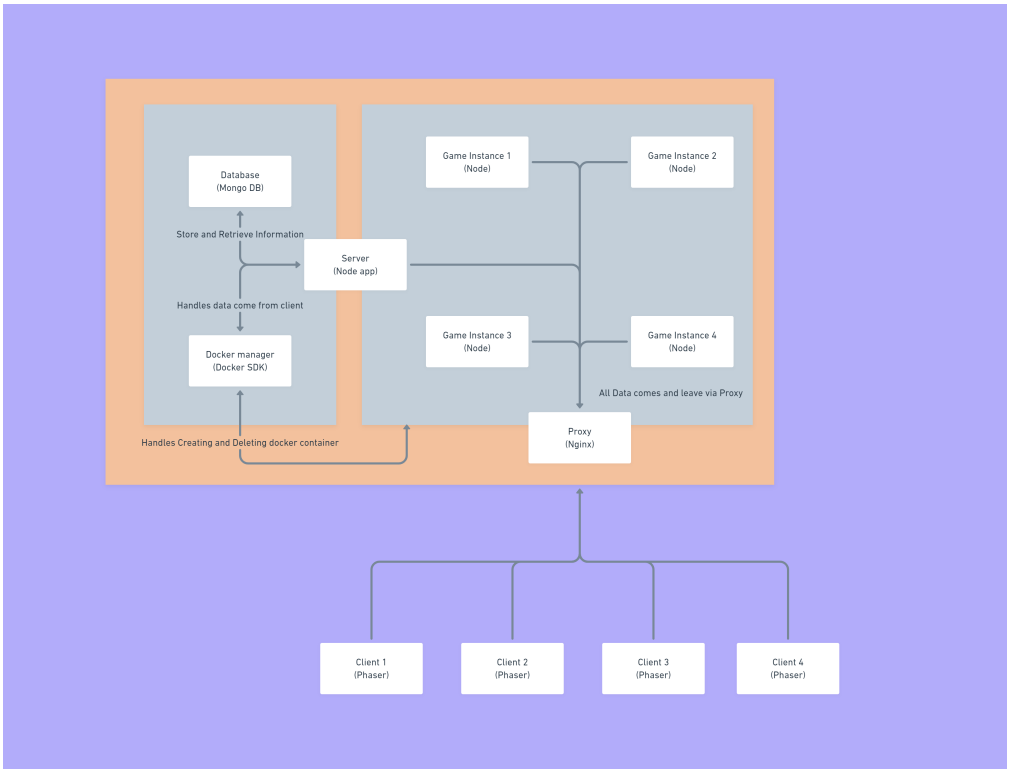


Figure 4.1: An overview of the basic game architecture

The Fig. 4.1 shows the architecture of the game. The top part of the architecture diagram depicts the server. 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 also make use of phaser game framework and MongoDB for the database. We make use of Google Cloud VM instance for the hosting of our game.

The following are the major components of the architecture:

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.

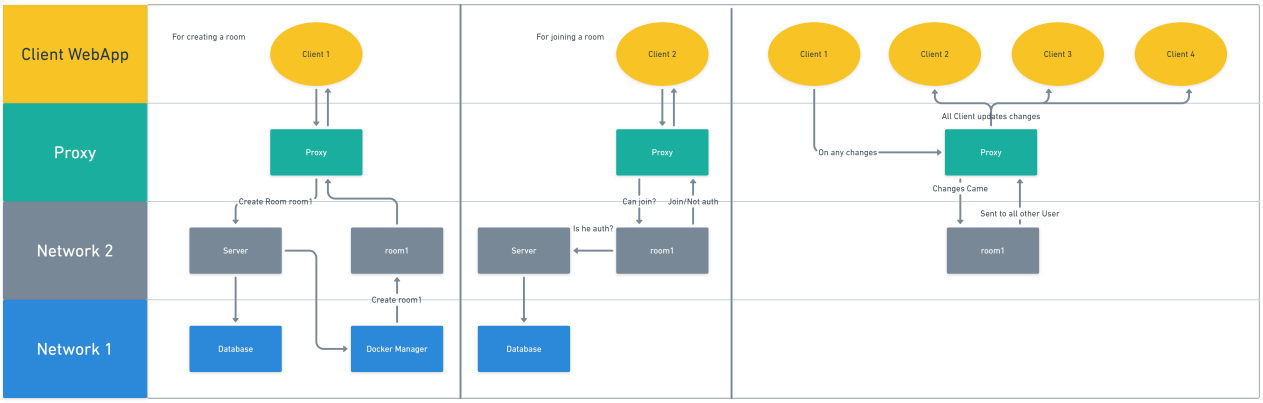


Figure 4.2: Swimlane diagram depicting the basic use cases of the application

The Fig. 4.2 shows the flow of the actions required for some basic use cases of the application. These are briefly described as follows:

- **Creating a room:** A room is created when a new game session is to be started. One of the users sends a request via the proxy to the server. The server then requests the docker manager to create a container for the room and also stores the necessary information in the database. After successful creation of a docker container (game instance container), the user will be notified. The other users can join this room by using the ID. It should be noted here that the initial user joins through the proxy and it is mapped to the created game instance container.

- **Joining a room:** A user who wishes to join a room sends a request to the room via the proxy using the room ID. The room verifies the credentials of the user from the server (The server checks the credentials in the database). After the successful authentication, the user will be allowed to join the room.

- **Normal game play:** The group of users in a room are playing on the same game instance, any change in the state of the game made by one user must be synchronised to all the other users. Thus, every change results in the client sending its state to the game instance, which will then be broadcasted to all the users. We make use of socket.io to enable broadcasting without explicit querying from the client.

Chapter 5

Software and Hardware Requirements

Software Requirements

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 WebSocket
- Nginx – For reverse proxy
- MongoDB – Database

Client Side:

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

Development:

- Docker desktop
- VS Code

Hardware Requirements

As the game is implemented in the form of a web app, any hardware configuration which can run a web browser can effectively run our game. However, a good network connectivity is required.

Chapter 6

Project Plan and Cost Estimation

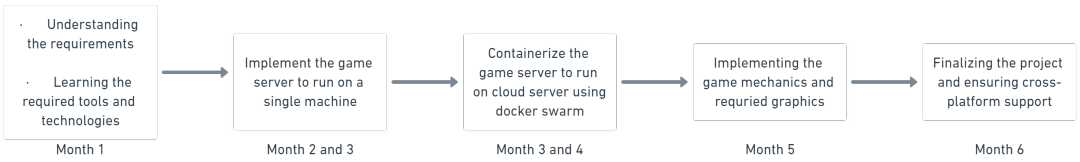


Figure 6.1: An overview of the project plan

Project planning

The Fig. 6.1 shows an overview of the plan for implementing the project. We have divided the implementation of the entire project into five phases spanning across six months. The tasks planned in these months are breifly described as follows:

- **Month 1:** Understanding the requirements: The key to successful software development is that all developers develop a clear and uniform understanding of the application requirements. To understand the requirements, we considered making a web application for the game. We planned on the basic architecture of the game decided on some tools and technologies needed to build the project.
  Learning the required tools and technologies: The following are the tools that we have decided for the implementation of the project.

  - Docker Swarm
  - Docker Manager
  - Phaser
  - TypeScript
  - Nginx

  We will learn these technologies over the course of the project with the help of tutorials, courses and relevant documentations.

- **Month 2 and 3:** Implementing the game server to run on a single machine: We begin the implementation of the game by making the game server run in the local environment. We will make use of some placeholder assets for the game, while testing the game server. The game server would be running on a single machine and multiple users will be able to connect to it. We will make use of 2D Phaser game engine to render the generated level using the placeholder assets and the game engine would also be able to handle the collision and overlap events. We will make use of Drunkard Walk algorithm for procedural generation of levels. We will make use of Socket.io package to handle bi-directional communication between the server and the user.

- **Month 3 and 4:** Containerize the game server to run on cloud server using docker swarm: Docker swarm is a container orchestration tool, meaning that it allows us to manage multiple containers deployed across multiple host machines. In this step of implementation, we containerize the game server, so that it can run in docker containers. We will have to make docker images for the various components of the game such as game server, docker manager and database. We will make use of Nginx for the purpose of reverse proxy. We will then be able to deploy it in the local docker swarm called as stack.

- **Month 5:** Implementing the game mechanics and required graphics In the this step of implementation, we plan to implement the specifics of the game which are:

  - Core game mechanics

  - Various types of enemies
  - Character skins
  - Weapons
  - Other game assets
  - Level progression
  - In-game economy

- **Month 6:** Finalizing the project and ensuring the cross-platform support: We will finalize the project, by packaging the client, server, game server and the docker manager. We will try to optimize the docker containers to improve the performance of the game. We will implement platform specific controls such as touch screen for android, controllers for consoles and so on. We will then be able to deploy the final project to the google console platform.

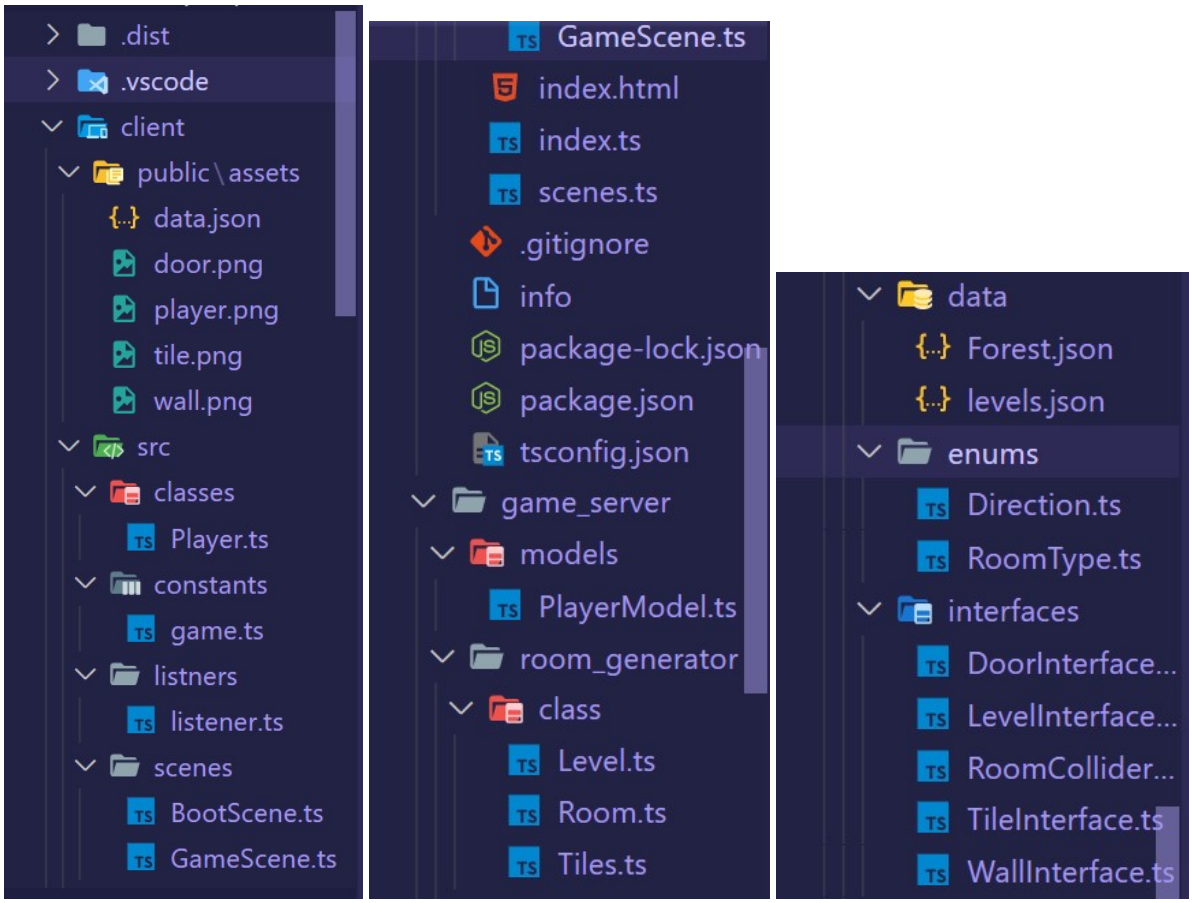
Cost Estimation

Sl. No.	Particulars	Description	Approx. Cost
1	Google Cloud	For hosting the Docker containers	Rs. 5000 p.m
2	Google Cloud Registry	To save and manage the Docker images	Rs. 1000 p.m
3	Software Licenses	Licenses for various proprietary tools we may need to use during the project development	Rs. 5000
4	Game Assets	Various assets required for the game like background music, player skins, weapons etc.	Rs. 3000
5	Courses	Courses required for learning the tools and technologies	Rs. 3000
6	Miscellaneous	Costs for preparing project reports and for stationary requirements	Rs. 5000

Chapter 7

Modules Implemented and Results

The following images (Fig. 7.1) shows the directory structure of the module implemented:



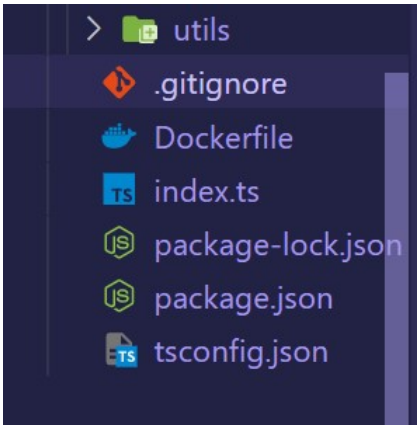


Figure 7.1: The directory structure of the project

The following parts constitute the implemented module:
Client Side

1. **Parcel.js:** We have used the Parcel.js bundler for automatically tracking files and configurations, plugins and dependencies. It allows hot reloading and better diagnostics and also boosts the performance of the application.

2. **src/classes/Player.ts:** This file 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.

3. **src/listners/listener.ts:** This file 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 listeners implemented so far:

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

    - Create 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 this event.

- Spawn 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.
- Player Movement: This event is used to receive the position and movements off all other players.
- Got Map: This event is used to receive the map of a level.
- 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.
- Level Completed: This event is triggered when a level is completed and a new level is loaded for all the players.

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

- 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.
- Player Movement: This event is triggered whenever a user moves his player. The updated position is broadcasted to all the other players.
- Get Map: Requests the server for the map of the level.
- Level Complete: 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.

4. **src/scenes/BootScene.ts:** This file loads all the required assets for the game. It showws a loading image until loading of all the assets is complete.

5. **src/scenes/GameScene.ts:** This file renders the current game state to the user. It renders the necessary images, adds colliders and initializes other players. It obtains all the data from the server using listeners and shows it to the user. It uses the listeners from the *listeners.ts* file.

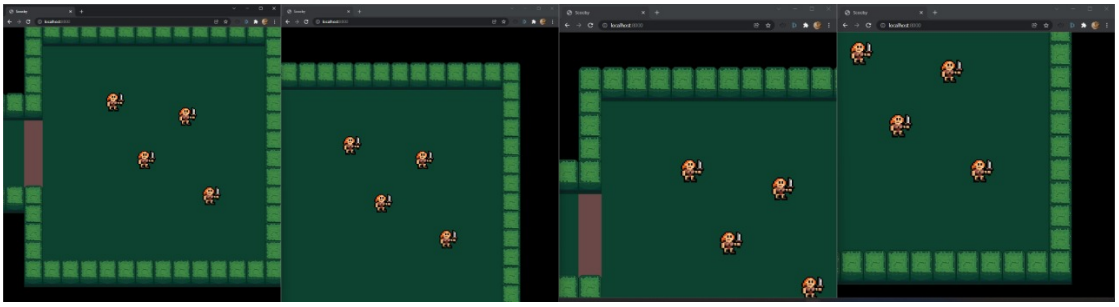
Server Side

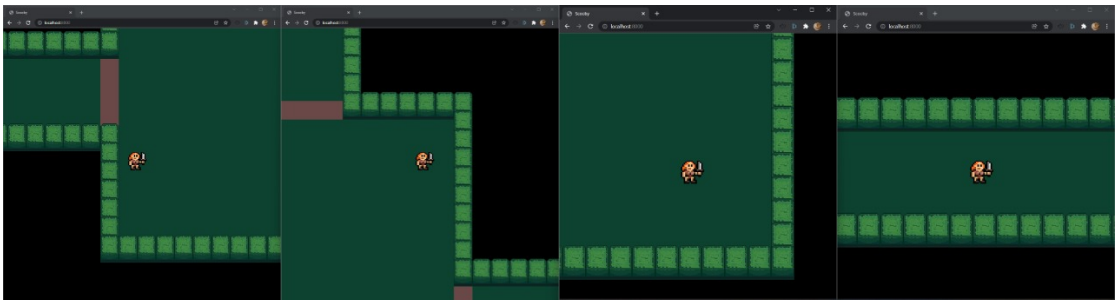
1. **index.ts:** This file listens for the socket events and broadcasts the data. It creates the procedural maps using room_generator.

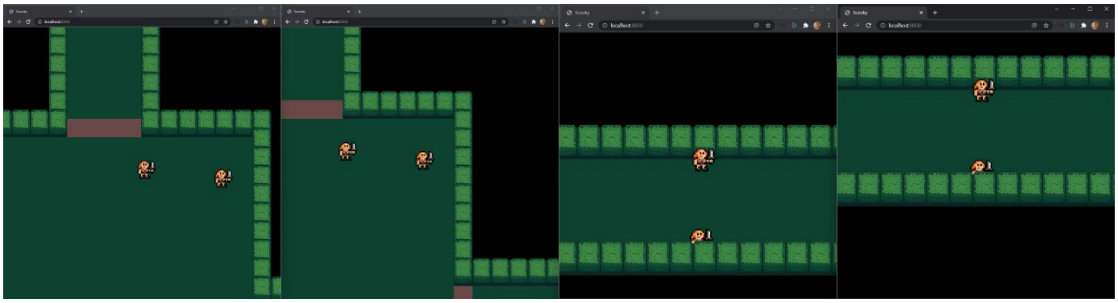
2. **room_generator:** It generates a procedural map for a level. It first initializes a 11*11 matrix, and sets the middle of this matrix as the start room. It uses the Drunkard Walk algorithm to populate the rest of the matrix until the maximum number of rooms for all the types of rooms are created. 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 server.

3. **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.

Results:







Conclusion

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 use docker containers for the game server which allows us to, easily manage, monitor and maintain the game server, allows for deployment on the cloud, reduces 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 propose to build an online multiplayer game that allow users from across the world 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.

Bibliography

[1] Cloud Gaming System in Docker Container Image:
http://norma.ncirl.ie/4233/1/arunpugalendhi.pdf

[2] Multi-platform Version of StarCraft: Brood War in a Docker Container: Technical Report:
https://arxiv.org/pdf/1801.02193

[3] Characterizing Docker Overhead in Mobile Edge Computing Scenarios:
https://dl.acm.org/doi/pdf/10.1145/3094405.3094411

[4] Enhancing Cloud Gaming User Experience through Docker Containers in Fog Nodes:
http://norma.ncirl.ie/4135/1/manojkannan.pdf

[5] The Future of Web and Mobile Game Development:
https://bit.ly/3IPqDBI

[6] The Development and Evaluation of Web-based Multiplayer Games with Imperfect Information using WebSocket:
https://bit.ly/35C9ywK

[7] Phaser game engine documentation:
https://newdocs.phaser.io/docs/3.55.2

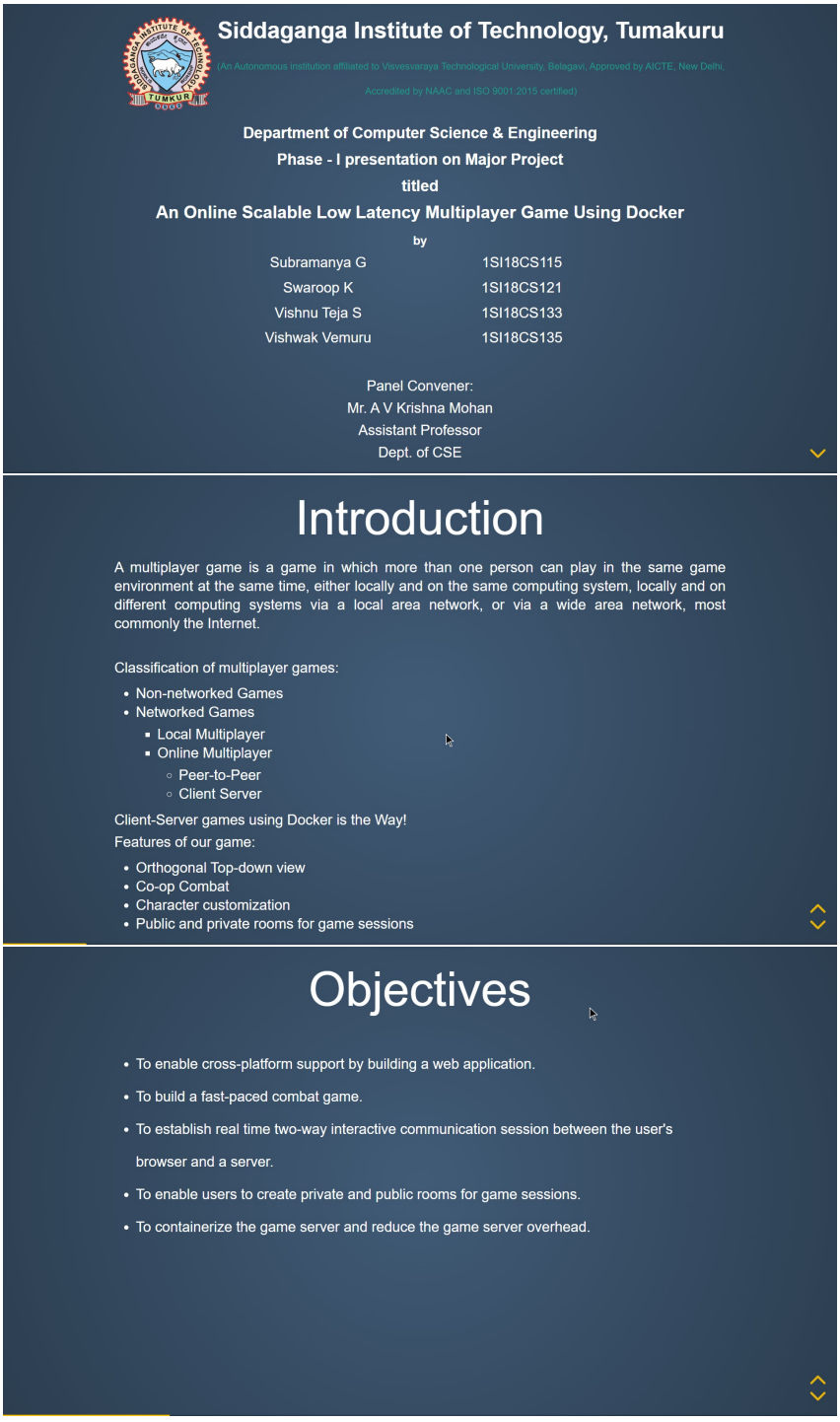
[8] Phaser game engine tutorials:
https://phaser.io/learn

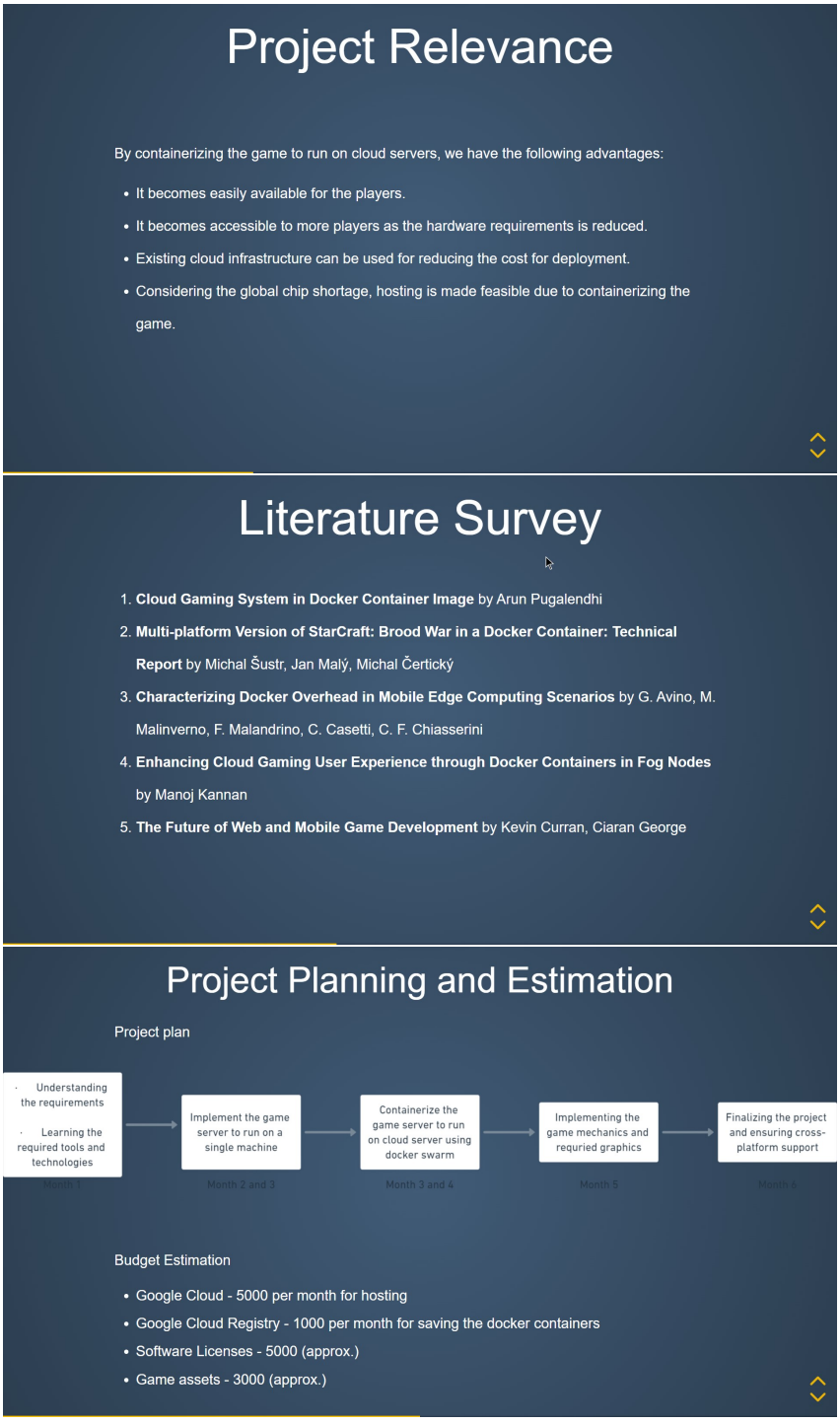
[9] Udemy course on game development using Phaser:
https://bit.ly/3GJTW7d

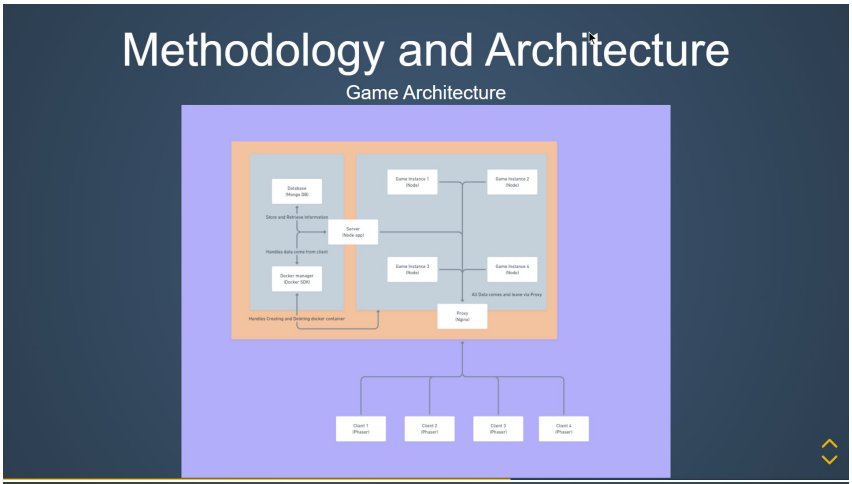
[10] MMORPG academy course in Zenva:
`https://bit.ly/34P7ReU`

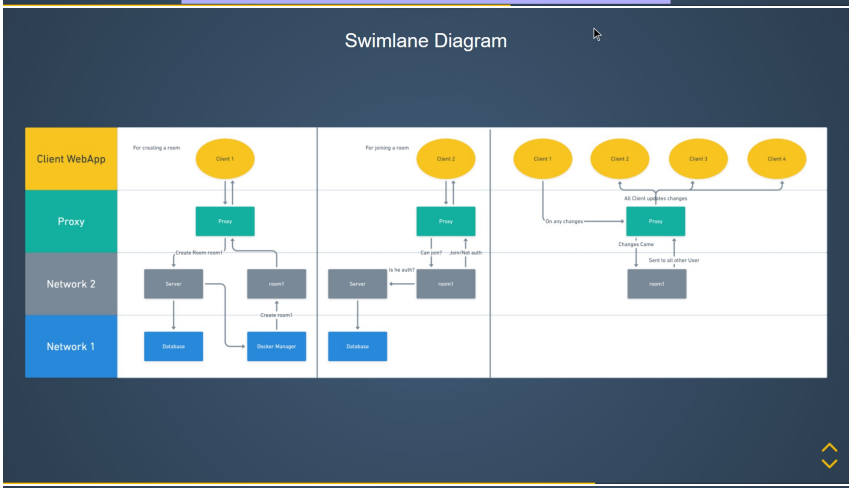
[11] JavaScript and TypeScript tutorials:
`https://youtube.com/c/Ourcadehq`

Phase I Presentation Slides

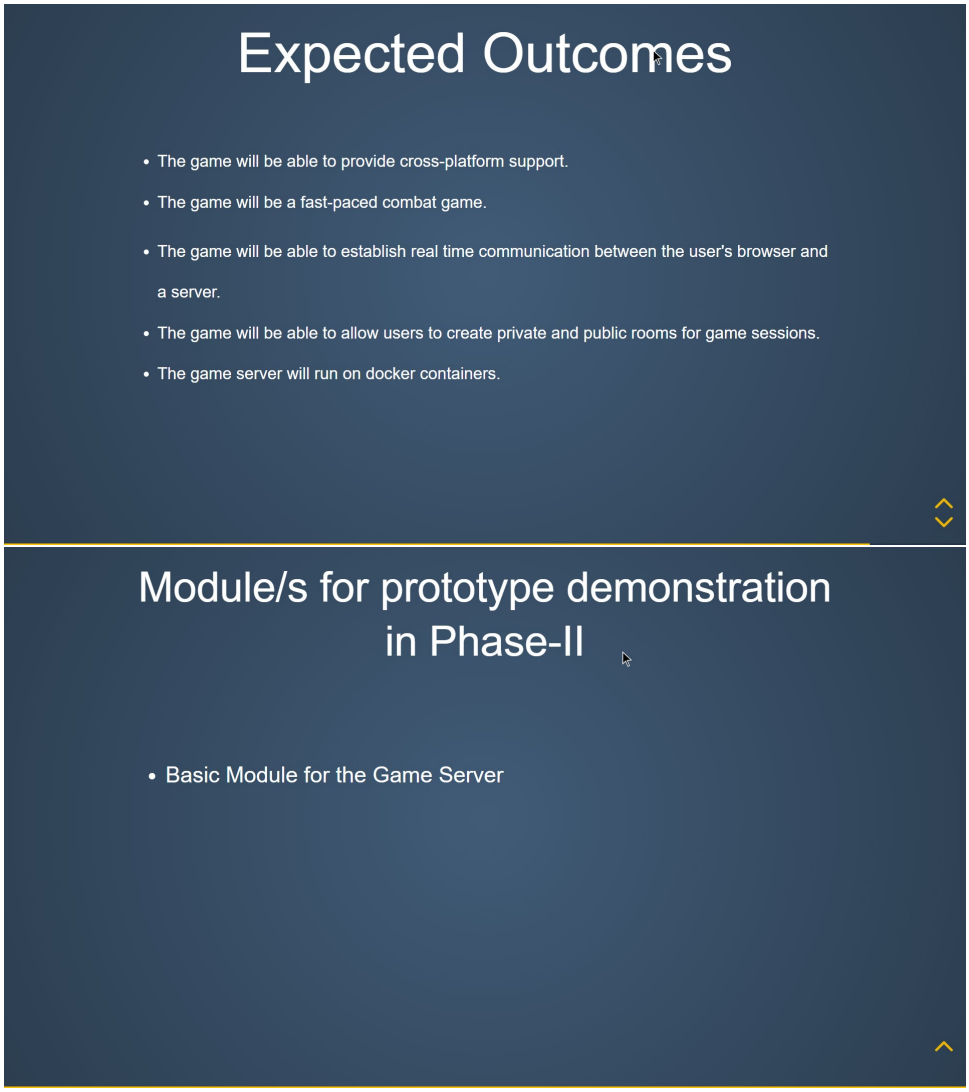



Project Relevance
By containerizing the game to run on cloud servers, we have the following advantages:
• It becomes easily available for the players.
• It becomes accessible to more players as the hardware requirements is reduced.
• Existing cloud infrastructure can be used for reducing the cost for deployment.
• Considering the global chip shortage, hosting is made feasible due to containerizing the game.
Literature Survey
1. Cloud Gaming System in Docker Container Image by Arun Pugalendhi
2. Multi-platform Version of StarCraft: Brood War in a Docker Container: Technical Report by Michal Šustr, Jan Malý, Michal Čertický
3. Characterizing Docker Overhead in Mobile Edge Computing Scenarios by G. Avino, M. Malinverno, F. Malandrino, C. Casetti, C. F. Chiasserini
4. Enhancing Cloud Gaming User Experience through Docker Containers in Fog Nodes by Manoj Kannan
5. The Future of Web and Mobile Game Development by Kevin Curran, Ciaran George
Project Planning and Estimation
Project plan
Understanding the requirements
Learning the required tools and technologies
Implement the game server to run on a single machine
Month 2 and 3
Containerize the game server to run on cloud server using docker swarm
Month 3 and 4
Implementing the game mechanics and requried graphics
Month 5
Finalizing the project and ensuring cross-platform support
Budget Estimation
• Google Cloud - 5000 per month for hosting
• Google Cloud Registry - 1000 per month for saving the docker containers
• Software Licenses - 5000 (approx.)
• Game assets - 3000 (approx.)


Methodology and Architecture
Game Architecture


Swimlane Diagram
Client WebApp
Proxy
Network 2
Network 1


Tools and Technologies
Server Side:
Docker – For containerization
Docker SDK – For managing the container
Node – For server scripting
Express – For server web application framework
Express-ws – For handling the WebSocket
Nginx – For reverse proxy
MongoDB – Database
Client Side:
React – Client-side rendering
Phaser – Game engine
Development:
Docker desktop
VS Code


Expected Outcomes
• The game will be able to provide cross-platform support.
• The game will be a fast-paced combat game.
• The game will be able to establish real time communication between the user's browser and a server.
• The game will be able to allow users to create private and public rooms for game sessions.
• The game server will run on docker containers.
Module/s for prototype demonstration in Phase-II
• Basic Module for the Game Server

Phase II Presentation Slides

Siddaganga Institute of Technology, Tumakuru

(An Autonomous institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi. Accredited by NAAC and ISO 9001:2015 certified)

Department of Computer Science & Engineering Phase - II presentation

on

Major Project titled

An Online Scalable Low Latency Multiplayer Game Using Docker

by

Subramanya G	1SI18CS115
Swaroop K	1SI18CS121
Vishnu Teja S	1SI18CS133
Vishwak Vemuru	1SI18CS135

Under the Guidance of:
Mrs. Kavitha M
Assistant Professor
Dept. of CSE

Introduction

A multiplayer game is a game 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 via a wide area network, most commonly the Internet.

Classification of multiplayer games:

- Non-networked Games
- Networked Games
  - Local Multiplayer
  - Online Multiplayer
    - Peer-to-Peer
    - Client Server

Client-Server games using Docker is the Way!

Features of our game:

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

Module Demo

Details of the Module

Implementing the game server to run on a single machine:

- We begin the implementation of the game by making the game server run in the local environment.
- We have used some placeholder assets for the game while testing the game server.
- The game server runs on a single machine and multiple users are able to connect to it.
- We have used 2D Phaser game engine to render the generated level using the placeholder assets and the game engine is capable of handling collision and overlap events.
- We have used Drunkard Walk algorithm for the procedural generation of levels.
- We have used socket.io package to handle bi-directional communication between the server and the user.

Tools and Technologies

Server Side:

- Docker – For containerization
- Docker SDK – For managing the container
- Node – For server scripting
- Express – For server web application framework
- Socket.IO – For handling the Bidirectional data sending
- Nginx – For reverse proxy
- MongoDB – Database

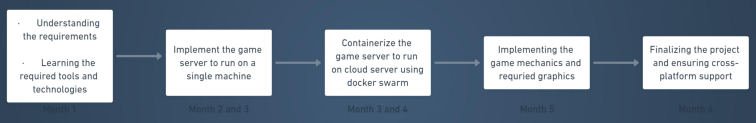
Client Side:

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

Development:

- Docker desktop
- VS Code

Project Planning



Conclusions

- We have built a simple client-side with some placeholder assets.
- Both the client-side and server-side are able to establish real-time communication between the user's browser and a server.
- The server is able to generate the procedural game room based on the Drunkard Walker algorithm.
- We have containerized single instance of the game server which can run on docker.

Module/s for prototype demonstration in Phase-III

- Containerize the game server, so that it can run on multiple docker instances.
- Make docker images for the various components of the game such as game server, docker manager and database.
- Make use of Nginx for the purpose of reverse proxy.
- Deploy it in the local docker called as stack.

Thank you