Database (Mongodb)

Mongodb is a cross platform document oriented database program, classified as a non sql database program, mongodb uses json like document with optional schema.

We use mongodb to store the game state. We will store game room id, the members of the game, the members inventory(weapons, character, coins collected)

Docker manager

It runs a docker sdk. Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Docker manager helps us to create and manage the various rooms in the game. Each docker instance will run a single instance of the game

Docker manager and database belongs to backend network

Server

It’s a node app container. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. It’s like a middleware between the backend and frontend network. It gives an abstraction between the database and the docker manager. It also manges the user authentication.

Game instance

It’s a node app container. This will be created by the docker manager. It belongs to frontend network. Each game instance docker container runs a gane server which a player can join. It manages the users who had joined, broadcasts any data sent and also create procedural maps for each game.

Proxy(nginx)

All the containers should be available to the outside public through some port. But we have only 65535 ports, among which 2000 is meant for other purpose. Hence we need something that maps the internal webserver to outside world. There is an advantage in docker, all the docker instance can be called using it’s name,rather than the ip. So there will be a proxy that maps the external request to the internal network. We made the proxy to work in such a way that any request with subdomain has the docker instance name will map to that docker instance.

Example: something.domain.com will maps to something container

Client:

This is written in typescript which will run on the browser of the user. It gets the data from the game instance, renders it. Any action (change in state) will be sent to game instance which will be broadcasted there.

Use case:

1. For creating room

Client sent request via proxy to the server. Server sends request to docker manager and also store in the database. After successful creation of docker container (game instance container), a user will be notified. The client can join the room based on the id.

Note: Client joins though the proxy, hence mapping is done here

1. Joining room

Client sent request to room via proxy. Room verifies the credentials from the server, in turn server queries in database. Based on the result, the client will be allowed to the room

1. Normal game play

Any change in the state of the game, result in sending the state to the game instance, which will be then broadcasted to all the client. Since we are using socket.io, broadcasting is possible, without client querying for it.

Implemented Modules:

Client:

We use parcel. Parcel automatically tracks all of the files, configuration, plugins, and dev dependencies that are involved in your build, and granularly invalidates the cache when something changes.

Src/classes/player.ts

Holds all logic for the player. Includes whether the player is the main player, get the inputs from user, make camera to follow the player with smooth animation

Src/listeners/listener.ts

Gets all the events from the socket and emit an event, which any module inside game can listen for.

Socket events are:

Create player – when a user joins, he gets all other user data

Spwan player – when they are playing and someone joins

Player movement – when any other player moves

gotMap – get Procedural map

removePlayer – if someone has left the game

levelcompleted – after completing a level

Game events

newPlayer – send I had joined

playermovement – send my movements

get Map – request for map

levelcomplete – send I had completed the level i.e, I had reached the end room

Src/scenes/Bootscene.ts

Loads all the assets. Till then it will show the loading image

Src/Scnenes/GameScene.ts

Gets all the data from the listeners and shows that to the user. Renders necessary images, add colliders and also initialize other players. Listen for all events from listeners module.

Game server:

Index.ts

Listen for all the socket events and broadcasts the data. Also creates the procedural maps using LevelGenerator file

Room\_generator

It generates a procedural map. It first initialize a 11\*11 matrix, and make the middle point as the start room, and moves randomly in any one of the four direction. It is called as drunker method(Drunkard Walk algorithm). It goes until there is max room generation or all the types of room is created.

Based on that array, we create the room with variable width and height. We will generate the walls of each room. Then we will generate the pathway between the rooms, as given by the algorithm. Also we had the walls. We will add the locations of the colliders and also overlappers. Then return the data in the json format.

Nginx (reverse proxy)

We have return the nginx proxy in such a way that and subdomain.domain is request we map it into the subdomain, here subdomain will be the docker instance name.