



FINAL REPORT



Capture The Flag (CTF) Multiplayer Game



CONTENTS

01.

Project Overview

02.

Live Demo

03.

System Architecture

04.

Core Code Implementation





01

Project Overview



Game Features



1

Real-time Multiplayer

The game supports real-time multiplayer across devices, allowing players to compete in flag capture with smooth interaction.

Key Features

It includes game room matchmaking, automatic team assignment, keyboard-controlled movement, and WebSocket-based state synchronization.

2

3

Tech Stack

The frontend uses HTML5 Canvas and JavaScript, the backend leverages FastAPI and Redis, and deployment is handled with Docker and Nginx.



02

Live Demo





Demo Presentation

The screenshot shows a web browser window with the address bar set to `localhost:3000`. The browser's address bar and tabs are visible at the top. Below the browser, a terminal window is open, displaying the output of a `docker-compose up --build` command. The terminal output shows the build process for a Docker Compose project, including the loading of local bake definitions, the transfer of Dockerfiles, and the building of three services: `nginx`, `python`, and `nginx`. The build process is completed successfully, and the terminal window is closed.

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！ https://aka.ms/PSWindows

PS C:\Users\lea> wsl
wsl: 检测到 localhost 代理配置，但未镜像到 WSL。NAT 模式下的 WSL 不支持 localhost 代理。
lea@DESKTOP-BJ9PTLJ: /mnt/c/Users/lea$ cd /home/lea/Capture-the-Flag
lea@DESKTOP-BJ9PTLJ: ~/Capture-the-Flag$ docker-compose up --build
[+] Building 2.3s (23/23) FINISHED
=> [internal] load local bake definitions                                0.0s
=> => reading from stdin 981B                                          0.0s
=> [client internal] load build definition from Dockerfile             0.1s
=> => transferring dockerfile: 131B                                     0.1s
=> [server internal] load build definition from Dockerfile             0.1s
=> => transferring dockerfile: 210B                                     0.1s
=> [server internal] load metadata for docker.io/library/python:3.11-slim 0.0s
=> [client internal] load metadata for docker.io/library/nginx:alpine  0.0s
=> [backup-server internal] load .dockerignore                        0.0s
=> => transferring context: 2B                                          0.0s
=> [client internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                          0.0s
=> [client internal] load build context                                0.0s
=> => transferring context: 438B                                        0.0s
=> [backup-server internal] load build context                        0.1s
=> => transferring context: 360B                                        0.0s
=> [client 1/3] FROM docker.io/library/nginx:alpine                  0.0s
=> [backup-server 1/5] FROM docker.io/library/python:3.11-slim       0.0s
=> CACHED [client 2/3] COPY . /usr/share/nginx/html                  0.0s
=> CACHED [client 3/3] COPY nginx.conf /etc/nginx/conf.d/default.conf 0.0s
```



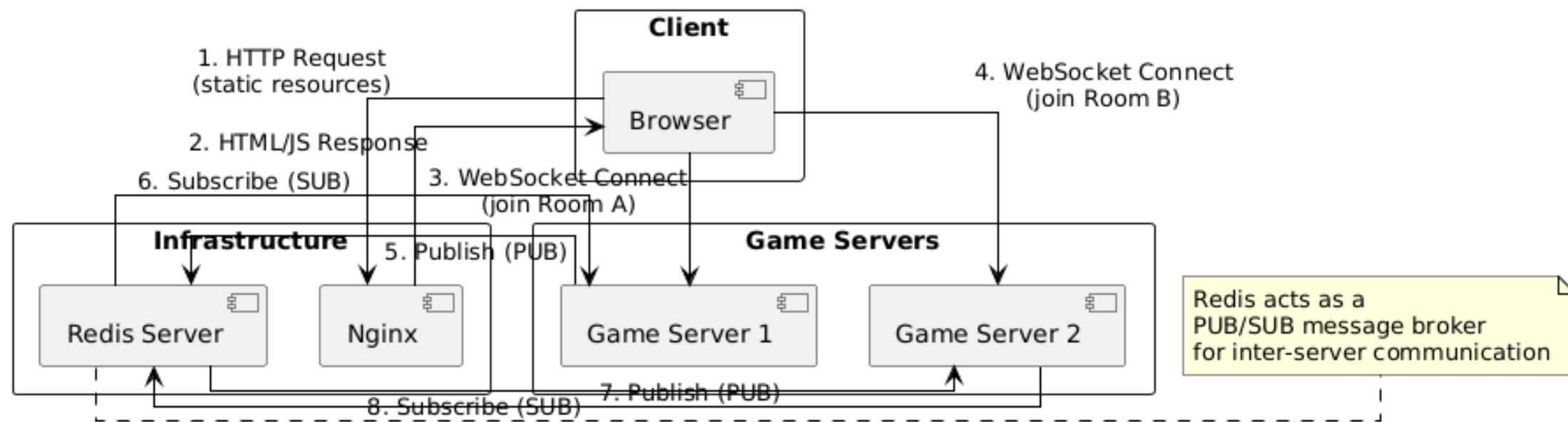
03

System Architecture





Component Diagram



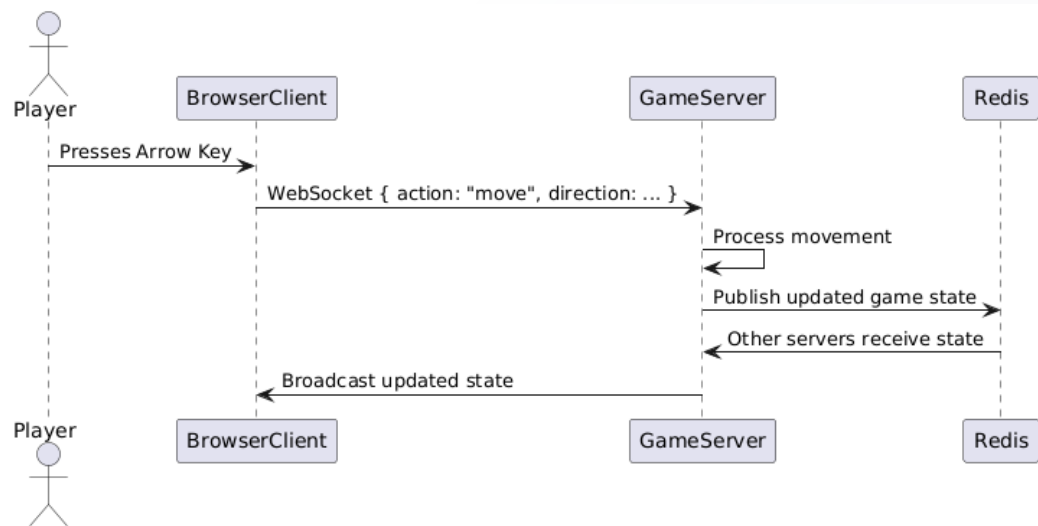


Dynamic Behavior

01

Sequence Diagram

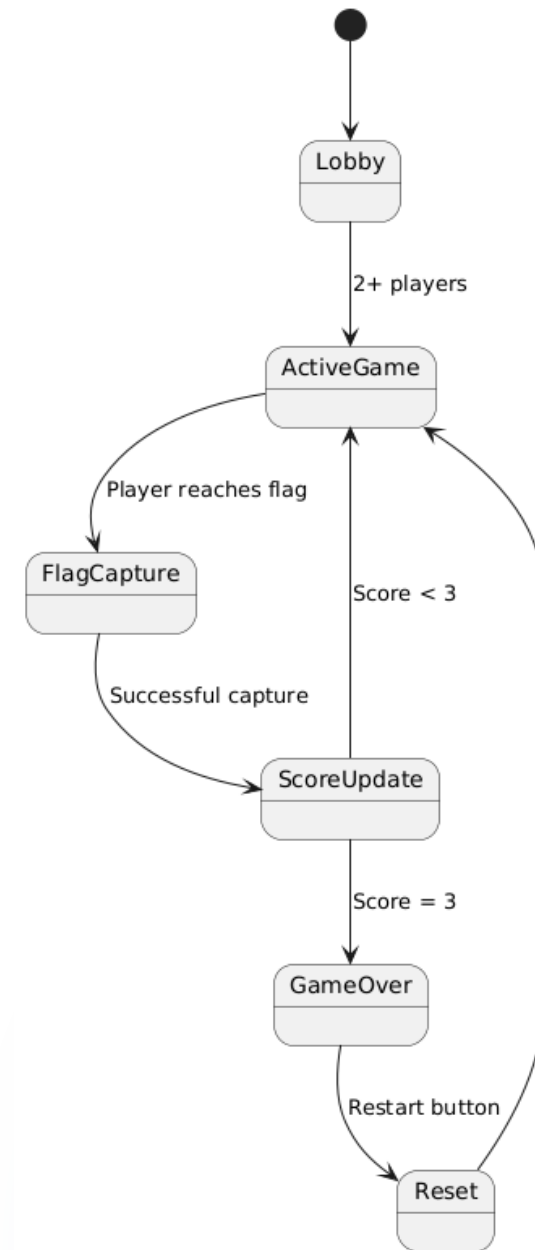
The sequence diagram illustrates the WebSocket message flow from client to server, Redis, and back to other clients.



02

State Diagram

It shows the game session lifecycle: Waiting, Active, Flag Captured, Game Over, and Reset.



Key Architecture Features



Client-Server

Real-time updates are achieved through WebSocket connections between clients and the server.

Fault Tolerance

A backup server maintains synchronized state via Redis for seamless failover.

Room Isolation

Each game room has an isolated state, ensuring independent game sessions.

Stateless Design

The system is designed to be stateless, with no persistent storage beyond session memory.





04

Core Code Implementation





Room Isolation & Player Management

01

Player Addition

Players are added to rooms with unique IDs, ensuring independent data storage for easy management.

02

State Synchronization

WebSocket pushes state updates to clients, ensuring they see the latest game state.





State Management & Updates

01

Broadcast State

The `'broadcast_state'` function publishes game state via Redis and pushes updates to all clients.

02

Consume Updates

The `'consume_game_updates'` function subscribes to Redis messages, filters self-originated updates, and applies changes to the local game state.





THANK YOU!

