Final Report

Capture The Flag (CTF) Multiplayer Game

Kuang Yi

yi.kuang@studio.unibo.it

July 2025

1. Abstract

The Capture The Flag (CTF) game project is a real-time, multiplayer web-based application developed as an academic and experimental initiative to simulate a competitive flag-capturing scenario. The game design enables players from two teams (red and blue) to connect, move within a bounded arena, and attempt to capture the opposing team's flag. The purpose of the project was not only to build a playable game but also to understand and implement real-time web technologies, multiplayer synchronization, and state management.

2. Concept

Product Type

Web-based multiplayer game with real-time interaction using HTML5 Canvas for rendering.

Use Case Collection

1. Users:

Online gamers seeking competitive team-based gameplay experiences

2. Interaction:

Frequency: Session-based (5-10 minute matches)

Devices: Desktop/laptop browsers with keyboard input

Connection: WebSocket-based real-time communication

3. Data Management:

Game state stored in server memory during sessions

Player positions, scores, and game status maintained in real-time

No persistent user data storage beyond active sessions

4. Roles:

Players: Team members (red/blue)

Game Server: Manages game state and synchronization

These technologies were selected for their simplicity, efficiency, and suitability in small-to-medium-scale real-time web applications.

3. Requirements

Functional Requirements

Real-time player synchronization through WebSocket connections

Team assignment (red/blue) based on connection order

Player movement in four directions (up, down, left, right)

Flag capture detection and scoring mechanics

Game state reset after flag capture

Win condition detection (first to 3 points)

Game state broadcasting to all connected clients

Manual game restart functionality via UI button

Non-Functional Requirements

Low-latency updates (<100ms) for real-time gameplay

Support for at least 10 concurrent players

Responsive controls with smooth character movement

Implementation Requirements

Python with FastAPI for server implementation

WebSocket protocol for real-time communication

HTML5 Canvas for client-side rendering

Docker-based containerization for deployment

Glossary

WebSocket: Full-duplex communication protocol over TCP

Pub/Sub: Publish-subscribe messaging pattern

CTF: Capture the Flag (game objective)

Acceptance Criteria

WebSocket connection established within 1 second

Team assignment alternates with each new player

Character moves smoothly in all directions

Score increments when player reaches opponent flag

All positions reset after capture

Game declares winner at 3 points

All players see state updates simultaneously

Game fully resets when restart button is clicked

4. Design

Architecture

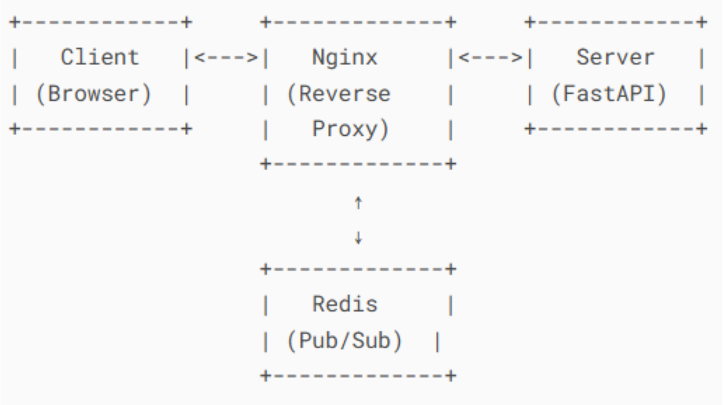
Client-Server Architecture with WebSocket communication:

Ensures authoritative game state management

Simplifies real-time synchronization

Centralizes game logic validation

Infrastructure



```
+------------+     +-------------+     +------------+
|   Client   |<--->|   Nginx     |<--->|   Server   |
| (Browser)  |     | (Reverse    |     | (FastAPI)  |
+------------+     |   Proxy)    |     +------------+
                   +-------------+
                          ↑
                          ↓
                   +-------------+
                   |   Redis     |
                   | (Pub/Sub)  |
                   +-------------+
```

Component Distribution:

All services in same Docker network

Redis for potential pub/sub expansion

Nginx handles WebSocket upgrade routing

Modelling

Domain Entities:

Player (id, position, team)

Flag (position, team)

GameState (players, flags, scores, status)

Domain Events:

PlayerConnected

PlayerMoved

FlagCaptured

GameWon

GameReposition

Interaction

Sequence Diagram

Client -> Server: WebSocket Connect

Server -> Client: Initial GameState

Client -> Server: MoveCommand (direction)

Server -> GameLogic: Process movement

GameLogic -> Server: Updated GameState

Server -> All Clients: Broadcast GameState

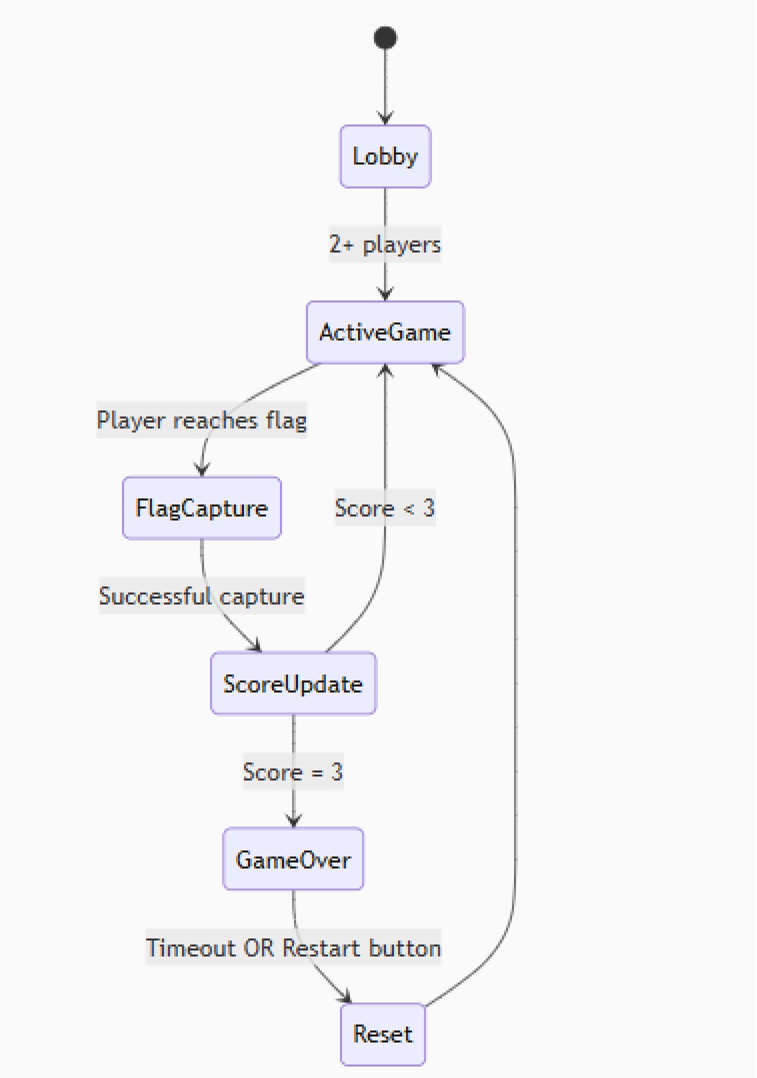
Behaviour

Stateful Components:

GameManager: Maintains game state (players, flags, scores)

Stateless: WebSocket handlers (message routing only)

State Transitions:



Data and Consistency

Data Storage:

Volatile in-memory storage (server process)

Redis used only for pub/sub messaging

No persistent database

Consistency Approach:

Single authoritative game state

Full state broadcast after each change

Eventual consistency through broadcasts

Fault-Tolerance

Failure Handling:

Player disconnect: Automatic removal from game

Server restart: Game state reset

Network issues: WebSocket reconnection

Availability

Approach:

No load balancing in current implementation

Single game instance limits scalability

Client reconnects on disconnect

Security

Current Implementation:

No authentication/authorization

Client-generated player IDs

No encryption (ws:// protocol)

Future Considerations:

JWT-based authentication

Input validation/sanitization

wss:// for encrypted communication

5. Implementation

Technical Implementation

Network Protocol: WebSocket

Data Format: JSON for all messages

Server Framework: FastAPI (ASGI)

Client: Vanilla JavaScript + Canvas API

Containerization: Docker + Docker Compose

Key Code Snippets

WebSocket Handler:

```
@app.websocket("/ws/{player_id}")
async def websocket_endpoint(websocket: WebSocket, player_id: str):
    await websocket.accept()
    clients[player_id] = websocket
    player = game.add_player(player_id)
    try:
        await broadcast_state()
        while True:
            data = await websocket.receive_text()
            if msg.get("action") == "move":
                game.move_player(player_id, direction)
                await broadcast_state()
    except WebSocketDisconnect:
        game.remove_player(player_id)
        del clients[player_id]
        await broadcast_state()
```

broadcast_state:

```
async def broadcast_state():
    state = game.get_state()
    for ws in clients.values():
        try:
            await ws.send_text(json.dumps(state))
        except Exception:
            pass
```

Flag Capture Logic:

```
def check_capture(self, player):
    opponent_flag = self.flags["blue"] if player.team == "red" else self.flags["red"]
    if abs(player.x - opponent_flag["x"]) < 40 and abs(player.y - opponent_flag["y"]) < 40:
        self.scores[player.team] += 1
        if self.scores[player.team] >= self.winning_score:
            self.game_over = True
        self.reset_positions()
```

Technologies Used

Server:	Python 3.11, FastAPI, Uvicorn
Client:	JavaScript, HTML5 Canvas
Infrastructure:	Docker, Docker Compose
Messaging:	Redis (pub/sub)

6. Validation

Automated Testing

Unit Tests:

```
def test_flag_capture():
    game = GameManager()
    player = game.add_player("p1", team="red")
    player.x = 700  # Blue flag position
    player.y = 330
    game.check_capture(player)
    assert game.scores["red"] == 1
    assert game.players["p1"].x == 650  # Reset position
```

Integration Tests:

WebSocket connection establishment

Multi-client state synchronization

Disconnect/reconnect scenarios

Restart button functionality

Test Automation:

Pytest for server tests

Jest for client tests

Docker-based test environment

Acceptance Testing

Manual Test Cases:

Multiplayer connection test (4+ players)

Movement synchronization verification

Flag capture scoring validation

Win condition triggering

Position reset after capture

Manual restart functionality

Why Manual:

Complex real-time interactions

Visual rendering verification

Cross-browser compatibility checks

UI element interaction testing

7. Release

Packaging

Component Modules:

Server: Python package (FastAPI application)

Client: Static web assets (HTML/JS/CSS)

Infrastructure: Docker Compose configuration

Dependency Graph:

client → nginx → server → redis

Versioning:

Semantic versioning (v1.0.0)

Distribution:

Docker Hub for container images

GitHub repository for source code

This Capture The Flag (CTF) multiplayer game demonstrates a working prototype using a modern lightweight stack. While the project lacks some critical mechanics like player blocking, it achieves a functional two-team system with real-time score tracking and state resets.

Cross-browser compatibility checks

8. Deployment

Installation Instructions

Install Docker and Docker Compose

Clone repository: git clone https://github.com/leahkuang/Capture-the-Flag.git

Start services: docker-compose up --build

Access game at: http://localhost:3000

Expected Outcome:

Redis container running

FastAPI server on port 8000

Client accessible on port 3000

9. User Guide

Game Instructions

1. Connecting:

Open game URL in browser

2. Controls:

Arrow keys for movement

Automatic team assignment

3. Objective:

Capture opponent's flag (red/blue)

First team to 3 captures wins

4. Visual Elements:

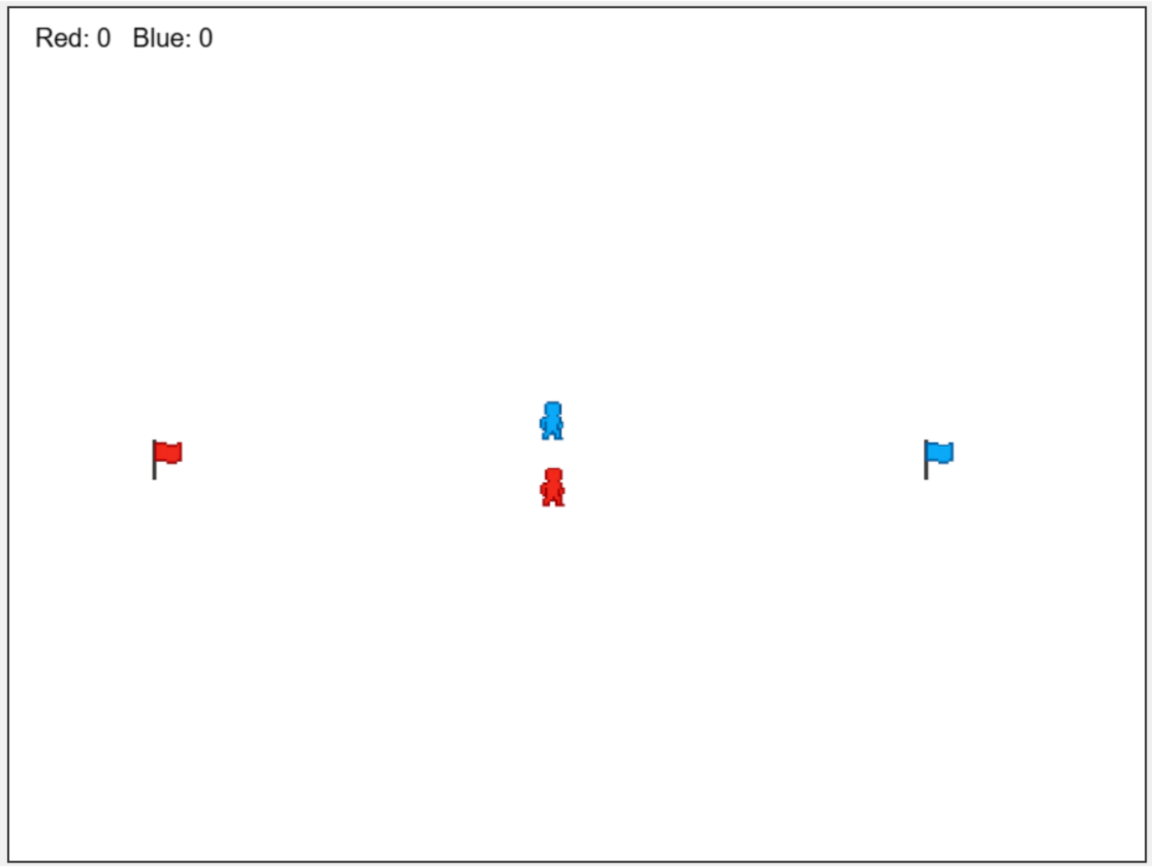
Players: Colored squares (red/blue)

Flags: Stationary flag icons

Score display: Top of screenInstall Docker and Docker Compose

Restart button: Top-right corner

Interface Overview



Troubleshooting

Issue	**Solution**
Connection failed	Check Docker containers
Movement not working	Refresh browser
No other players	Check multiple connections
Game not resetting	Verify server logs