

# Fractal Frogger Architecture

Full-Stack Multiplayer Game System

- PixiJS 8.16
- TypeScript 5.7
- Vite 6.4
- Express
- Socket.io
- Node.js

## CLIENT (Browser)

### Entry Point

**main.ts**  
src/main.ts

- Bootstrap PixiJS
- Create Game instance
- Load FroggerScene



### Game Engine

**Engine**  
engine/Game.ts

- RAF loop
- Fixed timestep 30ms
- Scene lifecycle

**Renderer**  
engine/Renderer.ts

- PixiJS wrapper
- Grid drawing
- Sprites

**Input**  
engine/Input.ts

- Keyboard events
- Arrow/WASD



### Scene & Systems

**FroggerScene**  
scenes/FroggerScene.ts

- GameState machine
- Composes systems
- Multiplayer sync

**Movement**

- Frog movement
- Log riding

**Collision**

- Hit detection
- Zone checks

**SocketClient**

- Server connection
- Event callbacks

**UI Components****StartScreen****HUD****GameOver****Victory****SERVER (Node.js)****Entry Point**

- `server/src/index.ts`
- Express + HTTP server
  - Socket.io setup
  - CORS config
  - Health endpoint
  - Graceful shutdown

**Game State (Authoritative)**

- `server/src/GameState.ts`
- lanes: Lane[] (20 lanes)
  - players: Map<id, Player>
  - Tick loop (150ms interval)
  - Obstacle spawning
  - Obstacle movement
  - Player management



## Event Handlers

events.ts

server/src/events.ts

- on('connection') - new socket
- on('join') - add player, assign color
- on('move') - update & broadcast position
- on('death') - mark dead, respawn timer
- on('victory') - broadcast win
- on('disconnect') - cleanup player



## Shared Types

types.ts

server/src/types.ts

- Player, Obstacle, Lane
- ClientToServerEvents
- ServerToClientEvents
- PLAYER\_COLORS[]

## WebSocket Communication (Socket.io)

### Client → Server

- join({ name? })
- move({ x, y })
- death({ cause })
- victory()



### Server → Client(s)

- welcome({ playerId, color, players, lanes })
- playerJoined({ playerId, color, name })
- playerLeft({ playerId })
- playerMoved({ playerId, x, y })
- playerDied({ playerId })

- `playerWon({ playerId })`
- `obstacles({ lanes })` – every 150ms

### Client Game Loop

```

requestAnimationFrame(t)
  → dt = t - lastTime
  → accumulator += dt
  → while (acc >= 30ms)
    → scene.update(0.03s)
      → movementSystem
      → collisionSystem
  → renderer.clear()
  → scene.render()

```

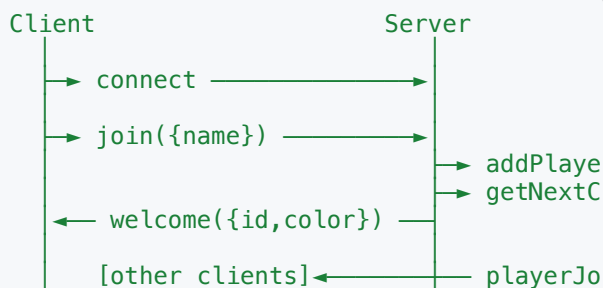
### Server Tick Loop

```

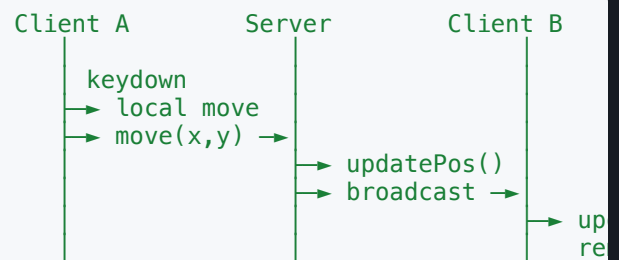
setInterval(150ms)
  → For each lane:
    → Increment spawn counter
    → If counter >= spawnRate:
      → spawnObstacle()
    → Move all obstacles
    → Remove off-screen
  → io.emit('obstacles', lanes)

```

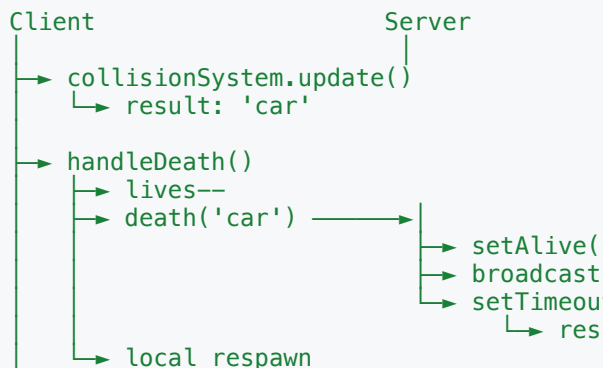
### Player Join Flow



### Movement Sync Flow



### Collision → Death Flow



### Obstacle State Authority

SERVER owns obstacle state:

- Spawning (when, where)
- Movement (velocity × tick)
- Despawning (off-screen)
- Broadcasting (every 150ms)

CLIENT receives & renders:

- `syncLanesFromServer()`
- `renderObstacles()`

Note: Server uses width-based obstacles, client uses sprite-based (potential mismatch)

### Core Data Structures

