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

Game
`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

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

▼

Game State (Authoritative)

GameState
`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

```
Client                    Server
  │                         │
  ├─► connect ──────────────►
  │                         │
  ├─► join({name}) ─────────►
  │                         ├─► addPlaye
  │                         ├─► getNextC
  ◄── welcome({id,color}) ──┤
  │                         │
  [other clients] ◄─────────┤ playerJo
```

Movement Sync Flow

```
Client A        Server        Client B
   │              │              │
   keydown        │              │
   ├─► local move │              │
   ├─► move(x,y) ─►              │
   │              ├─► updatePos() │
   │              ├─► broadcast ──►
   │              │              ├─► up
   │              │              re
```

Collision → Death Flow

```
Client                    Server
  │                         │
  ├─► collisionSystem.update()
  │     └─► result: 'car'
  │
  ├─► handleDeath()
  │     ├─► lives--
  │     ├─► death('car') ──────►
  │     │                     ├─► setAlive(
  │     │                     ├─► broadcast
  │     │                     └─► setTimeou
  │     │                           └─► res
  │     └─► local respawn
```

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

GameData (Client)
- frog: Frog
- lanes: Lane[]
- score: number
- timeRemaining: number
- level: number

Player (Server)
- id: string
- name: string
- color: number
- position: Point
- isAlive: boolean

Lane
- y: number
- type: LaneType
- obstacles: Obstacle[]
- spawnRate, direction
- speed: number

Obstacle (Client)
- id, position
- size: s|m|l|xl
- velocity: number
- type: car|log
- sprite?: SpriteData

Obstacle (Server)
- id, position
- width: number
- velocity: number
- type: car|log

Frog (Client)
- position: Point
- lives: 1-3
- isAlive: boolean
- isOnLog: boolean
- currentLogId?: string

Core/Entry Systems UI Server/Network

Architecture Notes

Server Authority
- Server owns obstacle state
- Server assigns player colors
- 150ms broadcast interval
- Client prediction for movement

Type Mismatch
- Server: obstacle.width
- Client: obstacle.size + sprite
- Potential sync issues
- Sprite data not sent

Tick Rates
- Client: 30ms (33 FPS)
- Server: 150ms (6.6 TPS)
- Client interpolates between

Player Colors
- 8 colors in rotation
- Assigned on join
- Green, Magenta, Cyan...