🎮 Zappy

Multi-Platform Networked Game Engine

C Server · Python AI · C++ GUI · Network Protocol · Real-time Strategy

Table of Contents

Project Overview

Zappy is a sophisticated networked game engine built as part of the Epitech curriculum. It demonstrates advanced concepts in network programming, artificial intelligence, and real-time graphics rendering across multiple programming languages and platforms.

🎮 Game Concept

Zappy simulates a competitive survival environment where AI-controlled players (Trantorians) compete for resources and advancement. Players must collect various resources, form teams, and perform incantations to level up while navigating a dynamic 2D world.

Key Features

- **Multi-language Architecture:** C server, Python AI clients, C++ GUI
- **Real-time Networking:** Custom protocol with efficient message handling
- **Advanced AI Behaviors:** Team coordination, resource management, strategic planning
- **3D Visualization:** Beautiful Raylib-powered graphics with animations
- **Resource Management:** Complex economy with 7 different resource types
- **Team Dynamics:** Collaborative elevation ceremonies and broadcasts
- **Scalable Design:** Supports multiple teams and configurable world sizes

System Components

Zappy Server

Language: C

Purpose: Core game engine and world simulation

- World generation and resource distribution
- Player lifecycle management
- Command processing and validation
- Real-time game state synchronization
- Team management and slot allocation
- Physics simulation and collision detection

AI Client

Language: Python 3

Purpose: Intelligent autonomous game agents

- Advanced pathfinding algorithms
- Resource collection strategies
- Team coordination protocols
- Elevation ceremony management
- Broadcast communication system
- Adaptive behavior patterns

GUI Visualizer

Language: C++ with Raylib

Purpose: Real-time 3D game visualization

- Dynamic 3D world rendering
- Player animation systems
- Resource visualization
- Real-time statistics dashboard
- Interactive camera controls
- Audio feedback system

Communication Flow

```
┌─────────────┐ TCP/UDP ┌─────────────┐ TCP/UDP ┌─────────────┐ │ │◄───────────────►│ │◄───────────────►│ │ │ AI
Clients │ Commands │ Server │ Game State │ GUI Client │ │ (Python) │ Responses │ (C) │ Updates │ (C++) │ │ │ │
   │ │ │ └─────────────┘ └─────────────┘ └─────────────┘ │ ▼ ┌─────────────┐ │ World State │ │ Management │
                                                    └─────────────┘
```

⚙ Installation Guide

📋 Prerequisites

System Requirements	Development Tools	Graphics Library	Network Libraries
Linux (Ubuntu 18.04+, Mint, etc.)	GCC/G++, Make, Python 3.8+	Raylib 4.0+ (for GUI component)	Standard POSIX sockets

Quick Installation

1

Clone and Enter Directory

```
git clone [repository-url] zappy
cd zappy
```

2

Install Raylib (Required for GUI)

```
# Install Raylib dependencies
sudo apt-get update
sudo apt-get install build-essential git cmake

# Clone and build Raylib
git clone --depth 1 https://github.com/raysan5/raylib.git raylib
cd raylib/src/
make PLATFORM=PLATFORM_DESKTOP RAYLIB_LIBTYPE=SHARED
sudo make install RAYLIB_LIBTYPE=SHARED
cd ../..
```

3

Build All Components

```
# Build everything with single command
make all

# Or build components individually:
make SERVER    # Builds zappy_server
make AI        # Builds zappy_ai
make GUI       # Builds zappy_gui
```

4

Verify Installation

```
# Check if executables were created
ls -la zappy_server zappy_ai zappy_gui

# Test server help
./zappy_server --help
```

🎮 Usage Instructions

🖥 Starting the Server

```
./zappy_server -p [port] -x [width] -y [height] -n [team1] [team2] ... -c [clients_per_team] -f [frequency]
```

Server Parameters

- **-p port:** Network port number (1024-65535)
- **-x width:** World width in tiles (minimum 10)
- **-y height:** World height in tiles (minimum 10)
- **-n teams:** List of team names (space-separated)
- **-c clients:** Maximum clients per team
- **-f frequency:** Game frequency/speed (default: 100)

```
# Example: Start server with 2 teams on 20x20 world
./zappy_server -p 4242 -x 20 -y 20 -n team1 team2 -c 5 -f 100
```

Launching AI Clients

```
./zappy_ai -p [port] -n [team_name] -h [hostname]
```

AI Client Parameters

- **-p port:** Server port number
- **-n name:** Team name (must match server configuration)
- **-h machine:** Server hostname (default: localhost)

Starting the GUI

```
./zappy_gui -p [port] -h [hostname]
```

Complete Game Session

1

Start the Server

```
./zappy_server -p 4242 -x 15 -y 15 -n red blue green -c 3 -f 100
```

2

Launch GUI (Optional but Recommended)

```
./zappy_gui -p 4242 -h localhost
```

3

Connect AI Players

```
# Terminal 1
./zappy_ai -p 4242 -n red

# Terminal 2
./zappy_ai -p 4242 -n blue

# Terminal 3
./zappy_ai -p 4242 -n green
```

🎮 Game Mechanics

🗺️ World Resources

Resource	Icon	Purpose	Rarity
Food		Essential for survival - consumed over time	Common
Linemate		Basic mineral for early level advancement	Common
Deraumere		Precious stone for mid-level incantations	Uncommon
Sibur		Rare crystal for advanced ceremonies	Rare
Mendiane	🟤	Mystical gem for high-level transformations	Rare
Phiras		Ancient artifact for master-level rituals	Very Rare
Thystame	☆	Legendary material for ultimate elevation	Legendary

📈 Level Progression System

```
Level 1 → 2: 1 player,  1 linemate
Level 2 → 3: 2 players, 1 linemate, 1 deraumere, 1 sibur
Level 3 → 4: 2 players, 2 linemate, 1 deraumere, 1 sibur, 2 phiras
Level 4 → 5: 4 players, 1 linemate, 1 deraumere, 2 sibur, 1 mendiane
Level 5 → 6: 4 players, 1 linemate, 2 deraumere, 1 sibur, 3 mendiane
Level 6 → 7: 6 players, 1 linemate, 2 deraumere, 3 sibur, 1 phiras
Level 7 → 8: 6 players, 2 linemate, 2 deraumere, 2 sibur, 2 mendiane, 2 phiras, 1 thystame
```

AI Player Commands

MOVEMENT

`Forward, Left, Right`

Navigate the world and change orientation

INTERACTION

`Take [resource], Set [resource]`

Collect and drop resources on current tile

PERCEPTION

`Look, Inventory`

Examine surroundings and check personal inventory

COMMUNICATION

`Broadcast [message]`

Send messages to all players with directional information

ADVANCED

`Incantation, Fork, Eject`

Level up, reproduce, or expel other players

Team Coordination Features

- **Broadcast System:** Directional messaging between team members
- **Resource Sharing:** Strategic resource distribution
- **Elevation Ceremonies:** Coordinated level-up rituals
- **Leader Election:** Dynamic leadership for complex operations
- **Pathfinding:** Intelligent navigation and obstacle avoidance
- **State Management:** Complex AI behavior trees

🛠 Development Information

Project Structure

```
zappy/
├── 🖥 Serveur/          # C server implementation
│   ├── src/             # Source files
│   ├── include/         # Header files
│   └── Makefile         # Build configuration
├──  AI/                # Python AI client
│   ├── main.py          # Entry point
│   ├── player.py        # Core AI logic
│   ├── server.py        # Network communication
│   └── Makefile         # Build configuration
├──  GUI/               # C++ GUI with Raylib
│   ├── src/             # Source files
│   ├── include/         # Header files
│   ├── ressources/      # 3D models, textures, audio
│   └── Makefile         # Build configuration
├──  Makefile           # Main build file
└── 📋 requirements.txt  # Dependencies
```

Build Commands

Available Make Targets

- **make all:** Build all components
- **make SERVER:** Build zappy_server only
- **make AI:** Build zappy_ai only
- **make GUI:** Build zappy_gui only
- **make clean:** Remove object files
- **make fclean:** Remove executables
- **make re:** Rebuild everything

🏆 Victory Conditions

The game continues until one team achieves dominance through:

- Having 6 players reach the maximum level (Level 8)
- Controlling the majority of high-level positions
- Effective resource monopolization strategies

🎓 Educational Value

This project demonstrates mastery of multiple advanced programming concepts:

Network Programming

- Socket programming with TCP/UDP protocols
- Custom protocol design and implementation
- Concurrent connection handling
- Message parsing and validation
- Network state synchronization

Artificial Intelligence

- Behavior trees and state machines
- Multi-agent coordination algorithms
- Pathfinding and navigation systems
- Strategic decision making
- Adaptive learning patterns

Graphics Programming

- 3D rendering with Raylib
- Animation systems and interpolation
- Real-time visualization techniques
- User interface design
- Audio integration and feedback

System Architecture

- Multi-component system design
- Inter-process communication
- Modular programming principles
- Cross-language integration
- Scalable software architecture

Built with ❤ for the Epitech curriculum

Technologies: C • Python • C++ • Raylib • Network Programming • AI Development