Zappy Game Protocol Specification

Abstract

This document specifies the Zappy Game Protocol, a network protocol for a multiplayer tile-based resource management game. The protocol defines communication between AI clients, a game server, and a graphical interface. Players compete in teams to achieve elevation through resource collection and ritual incantations on the planet Trantor.

Table of Contents

1. Introduction

The Zappy Game Protocol enables multiple AI clients to control inhabitants of planet Trantor in a competitive resource management game. The protocol supports real-time multiplayer gameplay with server-side state management and optional graphical visualization.

1.1. Requirements Language

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.2. Terminology

- AI Client: Autonomous program controlling a single player
- GUI Client: Graphical interface for game visualization
- Server: Central game state manager
- Player: Game entity controlled by an AI client
- Trantor: The game world planet
- Elevation: The process of advancing player levels
- Time Unit: Basic unit of game time (1/f seconds)

2. Game World Specification

2.1. Geography

The game world is a rectangular grid with wrapping edges. Players exiting one edge appear on the opposite edge, creating a toroidal topology.

World dimensions are specified as width (X) and height (Y) in tiles. Each tile can contain multiple players, resources, and eggs.

2.2. Resources

Six types of stones exist in the game world:

- linemate
- deraumere
- sibur
- mendiane
- phiras
- thystame

Food is also present and required for player survival.

2.3. Resource Distribution

Resources spawn according to density formulas:

Resource	Density
food	**0.5**
linemate	0.3
deraumere	0.15
sibur	0.1
mendiane	0.1
phiras	0.08
thystame	0.05

Total quantity = map_width * map_height * density

Resources respawn every 20 time units following the same distribution.

3. Protocol Overview

3.1. Architecture

The Zappy protocol operates in a client-server model with three types of participants:

- Game Server: Manages world state and enforces rules
- AI Clients: Control individual players autonomously
- GUI Client: Provides real-time visualization (optional)

3.2. Transport Protocol

All communication uses TCP sockets. The server MUST use poll() for socket multiplexing and handle multiple concurrent connections.

3.3. Message Format

All protocol messages are text-based and terminated by a newline character ('\n'). Commands are case-sensitive.

4. Client-Server Communication

4.1. Connection Establishment

Client connection follows this sequence:

1. Client opens TCP connection to server port
2. Server sends: "WELCOME\n"
3. Client sends: "<TEAM_NAME>\n"
4. Server sends: "<CLIENT_NUM>\n"
5. Server sends: "<X> <Y>\n"

Where:

- TEAM_NAME: String identifying the client's team
- CLIENT_NUM: Number of available slots for this team
- X, Y: World dimensions

4.2. Command Buffering

Clients MAY send up to 10 commands without waiting for responses. Commands exceeding this limit are ignored by the server. The server processes commands in first-in, first-out order.

5. AI Client Protocol

5.1. Player Commands

AI clients control players with the following commands:

Command	Syntax	Time	Response
Forward	"Forward\n"	7/f	"ok\n"
Right	"Right\n"	7/f	"ok\n"
Left	"Left\n"	7/f	"ok\n"
Look	"Look\n"	7/f	"[tile1,tile2,...]\n"
Inventory	"Inventory\n"	1/f	"[resource n,...]\n"
Broadcast	"Broadcast <msg>\n"	7/f	"ok\n"
Connect_nbr	"Connect_nbr\n"	-	"<value>\n"
Fork	"Fork\n"	42/f	"ok\n"
Eject	"Eject\n"	7/f	"ok\n"/"ko\n"
Take	"Take <object>\n"	7/f	"ok\n"/"ko\n"

Set	"Set <object>\n"	7/f	"ok\n"/"ko\n"
Incantation	"Incantation\n"	300/f	"Elevation underway\n Current level: <k>\n"/"ko\n"

5.2. Look Command Response Format

The Look command returns vision data in the format: "[player,object1 object2,...,object3,...]"

Vision extends based on player level:

- Level 1: 3 tiles (front center, front left, front right)
- Level n: (2n+1) tiles in a triangular pattern

Tile numbering starts at 0 (current position) and proceeds outward by levels.

5.3. Inventory Response Format

Inventory returns: "[food <n>, linemate <n>, deraumere <n>, sibur <n>, mendiane <n>, phiras <n>, thystame <n>]\n"

Where <n> represents the quantity of each resource.

5.4. Broadcast Messages

When a player broadcasts, all clients receive: "message <K>, <text>\n"

Where K indicates the direction (tile number) from which the sound originates, calculated using the shortest path on the toroidal world.

5.5. Ejection Notifications

When ejected, clients receive: "eject: <K>\n"

Where K indicates the direction from which the ejection occurred.

6. GUI Client Protocol

6.1. Authentication

GUI clients authenticate by sending "GRAPHIC\n" as the team name during connection establishment.

6.2. Server-to-GUI Messages

The server pushes world state updates to GUI clients. The complete GUI protocol specification is provided separately and follows the same TCP transport with newline-terminated messages.

GUI clients receive notifications for:

- Player movements and actions
- Resource spawns and collections
- Elevation rituals
- Player connections and disconnections

7. Game Mechanics

7.1. Player Lifecycle

New players start with:

- 10 food units (1260 time units of life)
- Level 1
- Random position and direction
- Empty inventory except for food

7.2. Elevation Requirements

Elevation requires specific resources and player counts:

Level	Players	linemate	deraumere	sibur	mendiane	phiras	thystame
1->2	1	1	0	0	0	**0**	**0**
2->3	2	1	1	1	0	**0**	**0**
3->4	2	2	0	1	0	**2**	**0**
4->5	4	1	1	2	0	**1**	**0**
5->6	4	1	2	1	3	**0**	**0**
6->7	6	1	2	3	0	**1**	**0**
7->8	6	2	2	2	2	**2**	**1**

7.3. Incantation Process

1. Player initiates with "Incantation\n"
2. Server verifies initial conditions
3. All participating players freeze for 300/f time units
4. Server re-verifies conditions at the end
5. On success: players advance level, resources consumed
6. On failure: "ko\n" response, no state change

7.4. Player Reproduction

The Fork command creates an egg on the current tile and adds a team slot. When a new client connects, a random team egg hatches, spawning the new player.

8. Time Management

8.1. Time Unit Definition

The basic time unit is 1/f seconds, where f is the frequency parameter (default: 100).

8.2. Action Timing

Each command has an associated execution time. Players are blocked from additional actions until the current command completes.

8.3. Life Decay

Players consume 1 food unit every 126 time units. Players die when food reaches zero, receiving the message "dead\n".

8.4. Resource Respawning

Resources respawn every 20 time units according to the density distribution specified in Section 2.3.

9. Error Handling

9.1. Invalid Commands

Unknown or malformed commands result in "ko\n" response.

9.2. Impossible Actions

Actions that cannot be performed (e.g., taking non-existent objects) return "ko\n".

9.3. Disconnection Handling

Client disconnections result in immediate removal of the player from the game world. Associated eggs are destroyed.

9.4. Buffer Overflow

Commands exceeding the 10-command buffer limit are silently rejected.

10. Security Considerations

This protocol is designed for controlled environments and does not include authentication or encryption mechanisms. Implementations should consider:

- Input validation for all commands
- Rate limiting to prevent resource exhaustion
- Bounds checking for world coordinates
- Buffer overflow protection

11. IANA Considerations

This document requires no IANA action. Port numbers are configurable and not standardized.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

12.2. Informative References

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, September 2014.