ZAPPY: Server

Guillaume Blaizot, Louis Langanay

June 23, 2024

Contents

1 Introduction

This document provides comprehensive documentation for the Network and the game part of the server of the Zappy project.

1.1 Binary Usage

```
./zappy_server -p port -x width -y height -n name1 name2 ...
-c clientsNb -f freq [ -v | --verbose ]
       port            is the port number
       width           is the width of the world
       height          is the height of the world
       nameX           is the name of the team X
       clientsNb       is the number of authorized clients per team
       freq            is the reciprocal of time unit for execution of actions
```

2 Server Protocol

The protocol is divided into three main components:

1. **Client-Side Protocol** (`protocol/client.h`)
2. **Shared Data Structures** (`protocol/data.h`)
3. **Server-Side Protocol** (`protocol/server.h`)

Each component plays a specific role in the overall communication process. Below is a detailed overview of these components, their structures, and their functions.

2.1 Client-Side Protocol

The client-side protocol manages client connections to the server, including creating, maintaining, and closing connections.

2.1.1 Structures

- `protocol_client_t`: Represents a client connected to the server.
  - `fd_set master_read_fds`: Set of file descriptors to monitor for reading.
  - `fd_set master_write_fds`: Set of file descriptors to monitor for writing.
  - `fd_set readfds`: Temporary set of file descriptors for reading.
  - `fd_set writefds`: Temporary set of file descriptors for writing.
  - `int sockfd`: Socket file descriptor.
  - `protocol_network_data_t network_data`: Network data containing server address and socket information.
  - `bool is_connected`: Connection status.
  - `char buffer[DATA_SIZE]`: Buffer for incoming data.

- `TAILQ_ENTRY(protocol_client_s) entries`: Queue entry for client.
- `TAILQ_HEAD(, protocol_payload_s) payloads`: Queue of payloads received from the server.

2.1.2 Functions

- `protocol_client_create`: Creates a new client and connects to the specified IP and port.
- `protocol_client_close`: Closes the client connection.
- `protocol_client_is_connected`: Checks if the client is currently connected.
- `protocol_client_listen`: Listens for incoming packets from the server.
- `protocol_client_send`: Sends a message to the server using a formatted string.

2.2 Shared Data Structures

Shared data structures are used by both client and server to manage network data and payloads.

2.2.1 Structures

- `protocol_network_data_t`: Contains network-related information.
  - `int sockfd`: Socket file descriptor.
  - `struct sockaddr_in server_addr`: Server address structure.
- `protocol_payload_t`: Represents a data packet.
  - `char message[DATA_SIZE]`: Message buffer.
  - `int fd`: File descriptor associated with the payload.
  - `TAILQ_ENTRY(protocol_payload_s) entries`: Queue entry for payload.

2.3 Server-Side Protocol

The server-side protocol manages incoming client connections, sending and receiving messages, and maintaining the server state.

2.3.1 Structures

`protocol_server_t`: Represents the server instance.

- `protocol_network_data_t network_data`: Network data containing server address and socket information.
- `fd_set master_read_fds`: Set of file descriptors to monitor for reading.
- `fd_set master_write_fds`: Set of file descriptors to monitor for writing.
- `fd_set read_fds`: Temporary set of file descriptors for reading.
- `fd_set write_fds`: Temporary set of file descriptors for writing.
- `TAILQ_HEAD(, protocol_client_s) clients`: Queue of connected clients.
- `TAILQ_HEAD(, protocol_payload_s) payloads`: Queue of payloads received from clients.
- `TAILQ_HEAD(, protocol_connection_s) new_connections`: Queue of new connections.
- `TAILQ_HEAD(, protocol_connection_s) lost_connections`: Queue of lost connections.

2.3.2 Functions

- `protocol_server_create`: Creates a new server and starts listening on the specified port.
- `protocol_server_close`: Closes the server.
- `protocol_server_close_client`: Disconnects a specific client.
- `protocol_server_is_open`: Checks if the server is currently open.
- `protocol_server_listen`: Listens for incoming packets from clients.
- `protocol_server_send`: Sends a message to a specific client using a formatted string.

3 Game

The Zappy game is an intricate multiplayer game that relies heavily on networking and server-client communication to function. This documentation focuses on the game logic and server functionalities, which are critical for ensuring a smooth and engaging gaming experience.

3.1 Game Logic

The game logic is implemented primarily in `game.h` and its associated source files. This includes the core functions that update the game state, manage the map, and initialize the game environment. Here's the key functions of the game logic:

3.1.1 Update_game

```
void update_game(zappy_server_t *server);
```

Description: This function is responsible for updating the game state. It handles AI actions, lifespan, and updates the game map periodically.
Parameters: `zappy_server_t *server`, a pointer to the `zappy_server_t` structure representing the game server.

3.1.2 Update_map

```
void update_map(zappy_server_t *server);
```

Description: Updates the game map, managing the resources and handling events such as meteor showers which can replenish resources on the map.
Parameters: `zappy_server_t *server`, a pointer to the `zappy_server_t` structure representing the game server.

3.1.3 init_map

```
bool init_map(zappy_server_t *server);
```

Description: Initializes the game map by allocating memory for it and setting up initial resource densities.
Parameters: `zappy_server_t *server`, a pointer to the `zappy_server_t` structure representing the game server.
Returns: `true`, if the map initialization was successful, `false` otherwise.

3.1.4 Update_ai

```
void update_ai(zappy_server_t *server);
```

Description: Updates the state of AI players, including actions like moving and gathering resources.
Parameters: `zappy_server_t *server`, a pointer to the `zappy_server_t` structure representing the game server.

4 Debugging

For debugging purposes, the `protocol/debug.h` file provides utilities to log and track the events happening within the server. The main structure and function are as follows:

4.1 Debug Structure

- `debug_log_t`: A structure to represent a log message.
  - `char *message`: The message to log.
  - `time_t timestamp`: The time the message was logged.

4.2 Debug Functions

- `void log_message(const char *format, ...)`: Logs a message to the debug output.

5 Conclusion

This documentation aims to cover the critical components of the Zappy project, providing a detailed overview of the server protocol, game logic, and debugging utilities. For further inquiries or issues, please refer to the respective header files and source code.