Network Working Group
Request for Comments:
Category: Informational

Enric M. / Candela

July 2024

Jetpack Multiplayer Protocol Specification

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This document specifies the application-level protocol and map format for the Jetpack Multiplayer game, a 2D platformer using a client-server architecture. The protocol enables communication of player states and game objects between clients and a central server.

Table of Contents

1. Introduction

The Jetpack Multiplayer Protocol (JMP) is a text-based application-layer protocol designed for real-time synchronization of game state in a 2D platformer environment. The protocol operates over TCP and supports up to two concurrent players with basic physics simulation and object interaction.

2. Protocol Overview

Communication between clients and server uses newline-delimited ASCII strings with space-separated fields. The protocol follows a simple command-response model:

- Clients send input commands
- Server broadcasts game state updates
- Server maintains authoritative game state
- Clients render state updates

Key features include player position synchronization, coin collection mechanics, and jump physics simulation.

3. Connection Setup

3.1. Server Initialization

Server command format:
./server_app -p <port> -m <map_path>

Example:
./server_app -p 4242 -m maps/basic.map

```
./server_app -p 4242 -m maps/basic.map
```

3.2. Client Connection

Client command format:
./client_app -ip <ipv4> -p <port>

```
./client_app 127.0.0.1 4001
```

On successful connection, server responds with:
ID <player_id>\n

Where <player_id> is 0 or 1, assigned in connection order.

```
Connected to server at 127.0.0.1:4242
Assigned Player ID: 0
```

4.Message Format

4.1. Server-to-Client Messages

ID <player_id>
- Initial assignment message
- Example: ID 0\n

COIN <coin_id> <x> <y>
- Coin state broadcast
- Example: COIN 2 300 150\n

```
Broadcasting coin: 0 at (100, 200)
Broadcasting coin: 1 at (250, 200)
Broadcasting coin: 2 at (400, 200)
Broadcasting coin: 3 at (550, 200)
Broadcasting coin: 4 at (700, 200)
```

MOVE <player_id> <x> <y>
- Player position update (60Hz)
- Example: MOVE 1 120 380\n

4.2. Client-to-Server Messages

INPUT <player_id> [JUMP]
- Client input notification
- Example: INPUT 0 JUMP\n

5. Server Behavior

5.1. Connection Management

- Accepts maximum of 2 concurrent connections
- Maintains active player state:
  * Position (x, y)
  * Vertical velocity
  * Jump state
  * Connection status

5.2. Game Logic

- Physics simulation at 60Hz:
  * Gravity: 400 units/s²
  * Jump power: 1000 units/s
  * Terminal velocity: 400 units/s
- Coin management:
  * Fixed initial positions
  * Periodic state resynchronization
- Boundary enforcement:
  * Window constraints (800x600 units)
* Cube size (50 units)

```
#pragma once

#include <SFML/Graphics.hpp>
#include <iostream>
#include <unordered_map>
#include <mutex>
#include <algorithm>

#include "../CubeState.hpp"
#include "../NetworkClient.hpp"

class Game {
private:
    SharedCubeState *state;
    sf::RenderWindow window;
    sf::RectangleShape cubes[2];

    sf::Texture backgroundTexture;
    sf::Sprite backgroundSprite1;
    sf::Sprite backgroundSprite2;
    float backgroundScrollSpeed = 50.0f;
    float backgroundOffset = 0.0f;
    const float window_height = 600.0f;

    float player_velocities[2] = {[0]=0.0f, [1]=0.0f};

    sf::Clock clock;

    void loadBackground();
    void updateBackground(float dt);
    void handleEvents();
    void update(float dt);
    void render();
    void updateInputs();

    sf::Texture coinTexture;
    sf::Sprite coinSprite;
    sf::CircleShape coinShape;
    bool useCoinShape = false;
public:
    Game(SharedCubeState *state);
    void run();

};
```

5.3. Failure Handling

- On client disconnect:
  * Close corresponding socket
  * Mark player as inactive
  * Pause game simulation

6. Client Behavior

6.1. State Management

- Maintain local copy of:
  * Player positions (0 and 1)
  * Active coin states
- Render game state at received update rate

6.2. Input Handling

- Capture space bar for jump input
- Send INPUT messages at 60Hz interval
- No local physics simulation

7. Security Considerations

This protocol contains no inherent security mechanisms. Implementers should be aware that:

- No authentication is performed

- No message encryption is used
- Game state is vulnerable to injection
- Denial-of-service attacks are possible

Production deployments should implement additional security layers.

8. References

[RFC7322] Bormann, C., "RFC Format Framework", RFC 7322, DOI 10.17487/RFC7322, September 2014, <https://www.rfc-editor.org/info/rfc7322>.