LEUNG Tung Lam (220280643)

Proxy War Game Documentation

ITP4708 Game Servers Design and IMPLEMENTATION PROJECT

Contents

[Overview 1](#_Toc165238057)

[Gameplay 2](#_Toc165238058)

[Communication between clients and server 3](#_Toc165238059)

[Installation and Setup 3](#_Toc165238060)

[Usage Guide 3](#_Toc165238061)

[External Libraries and Frameworks 3](#_Toc165238062)

[Known Issues and Limitations 3](#_Toc165238063)

[Testing 4](#_Toc165238064)

## Overview

The Proxy War game is a web-based multiplayer game that allows players to compete in a strategic turn-based game. The game is built using JavaScript, HTML, and CSS. The game server is implemented using Node.js and WebSocket for real-time communication between the server and the clients.

In the game, each player starts with a certain amount of money. Players invest their money in three attributes: Force, Arms, and Food. The investments in the same attribute are compared, and the player who invested more wins that attribute. The winning player's money is increased by the amount they invested multiplied by the number of attributes they won. The game ends after three rounds, and the player with the most money wins. 一張含有 文字, 電腦遊戲, 數位合成, 遊戲軟體 的圖片

自動產生的描述

## Gameplay

"Proxy War" is a text strategy web game where two players, each with an army of agents, compete against each other. The army has three attributes: troops, armaments, and food.

1. Each player starts with a certain amount of money. The attacker has 100 units, and the defender has 90 units.
2. Players invest their money in the three attributes. The amount invested in each attribute can vary.
3. After both players have invested, the system will broadcast the total amount invested by each player, but not the specific investment items.
4. The investments in the same attribute are compared. The player who invested more wins that attribute.
5. The winning player's money is increased by the amount they invested multiplied by the number of attributes they won.
6. The game ends after three rounds. The player with the most money wins.

There are also special skills you can use:

* National Debt (NB): This skill includes the remaining money in the settlement multiplier.
* Quantitative Easing (QE): This skill ensures the settlement multiplier is at least 1, but no more than 3.
* Intervention Currency (IC): This skill reduces the opponent's settlement by 50%.

## Communication between clients and server

The game uses a series of predefined message types to facilitate communication between the client and the server. These message types are defined in the MESSAGE\_TYPES.js file and include actions such as setting a player's name, updating labels, joining a game, and initiating a battle.

## Installation and Setup

To set up the development environment, you need to have Node.js installed on your machine. After cloning the repository, run **npm install** to install all the dependencies. This command reads the package.json file to determine which dependencies to install. To start the server, use the command node server.js. To view the game, open index.html in your browser.

## Usage Guide

To use the chat application, enter your name and click the "Join Game" button. You can then send messages to other players using the chat input field. To play the game, enter the amount of funding you want to allocate to your force, arms, and food, then click the "Battle" button.

## External Libraries and Frameworks

### Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. We use Mongoose in this project to manage the chat records of the game. Mongoose provides a rich query builder, middleware, and schema-based models, making interactions with MongoDB database more straightforward and intuitive.

* Schema definition: Mongoose allows us to define the structure of our documents, including field names, types, default values, etc.
* Data validation: Mongoose provides built-in data validation features, ensuring that the data saved to the database meets predefined conditions.
* Query building: Mongoose provides a rich query API, making it easy to construct complex queries.
* Middleware: Mongoose's middleware feature allows us to run custom functions before or after certain operations (like saving a document).

### Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that can be used to develop server-side and networking applications. We use Node.js in this project to develop the game server.

### WebSocket

WebSocket is a network communication protocol that allows full-duplex communication over a single TCP connection. We use WebSocket in this project to implement real-time communication between the server and clients.

### jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It simplifies HTML document traversal and manipulation, event handling, animation, and Ajax. We use jQuery in this project for DOM manipulation and event handling.

## Known Issues and Limitations

The Proxy War game server, while operational and providing an engaging gaming experience, has certain limitations and areas that could benefit from further refinement:

### Player Capacity

At present, the game is designed to accommodate only two players simultaneously. This limitation could be addressed by redesigning the game to support more players, enhancing the multiplayer experience.

### Error Management

The current version of the game could benefit from more comprehensive error management, especially in handling invalid user inputs and server errors. Improving this aspect would make the game more robust and user-friendly.

### Database Stability

The server's interaction with the database does not currently account for potential database connection issues. Enhancing this aspect would improve the game's resilience and stability.

### User Interface

The user interface, while functional, is relatively basic. Enhancements to the user interface could provide a more immersive and visually appealing gaming experience.

### Game Mechanics

The game, in its current state, offers a limited set of features. Introducing additional game mechanics, such as power-ups and varied game modes, could make the game more engaging and challenging.

While these limitations do not hinder the basic functionality of the game, addressing them could significantly enhance the user experience and the overall quality of the game.

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Feature to Test | Test Steps | Expected Result | Tested Result |
| TC01 | Static File Serving | Access a static file from the 'public' directory | The server should correctly serve the requested file | Pass |
| TC02 | State Management | Transition the game state from one state to another | The server should correctly update the game state | Pass |
| TC03 | Waiting List Management | Add and remove players from the waiting list | The server should correctly update the waiting list | Pass |
| TC04 | Player Assignment | Assign players to the roles of attacker and defender | The server should correctly assign player roles | Pass |
| TC05 | Troop Management | Update and retrieve information about the troops of the attacker and defender | The server should correctly manage troop information | Pass |
| TC06 | Round Management | Increment the round number | The server should correctly update the round number | Pass |
| TC07 | Delay Increment | Apply a delay of 1 second | The server should correctly apply the delay | Pass |
| TC08 | Attribute Multiplier | Apply an attribute multiplier of 0.5 | The server should correctly apply the multiplier | Pass |
| TC09 | WebSocket Connection | Open and close a WebSocket connection with the client | The server should correctly manage the WebSocket connection | Pass |
| TC10 | Message Handling | Send a message from the client to the server | The server should correctly parse the message and perform the appropriate action | Pass |
| TC11 | Join Game Message | Send a JOIN\_GAME message from the client to the server | The server should correctly process the JOIN\_GAME message | Pass |
| TC12 | Database Connection | 1. Start the server 2. Check the console for database connection status | "Database connected successfully" message should be displayed in the console | Pass |
| TC13 | Save Chat Message | 1. Start the server 2. Send a chat message from the client 3. Check the database for the new chat message | The sent chat message should be saved in the database | Pass |
| TC14 | Retrieve Chat Messages | 1. Start the server 2. Connect a new client 3. Check the chat messages received by the client | The client should receive all the chat messages saved in the database | Pass |
| TC15 | Database Disconnection | 1. Start the server 2. Stop the server 3. Check the console for database disconnection status | "Database disconnected successfully" message should be displayed in the console | Pass |