**Hong Kong Institute of Vocational Education (Chai Wan)**

**HD in Game Software Development (IT114107)**

**ITP4708 Game Servers Design and Implementation**

**EA Project**

****

**Game Title:**

**星辰記憶 • Starry Memories**

**Student Name: Yung Kwan Wai**

**Student Number: 230033193**

**Class: IT114107 - 2A**

Contents

[1. Overview 3](#_Toc185325487)

[*Game Introduction* 3](#_Toc185325488)

[*Technical Architecture* 3](#_Toc185325489)

[*Game Background Story* 3](#_Toc185325490)

[2. Game Play 3](#_Toc185325491)

[*Basic Rules* 3](#_Toc185325492)

[*Victory Conditions* 4](#_Toc185325493)

[3. Communication between clients and server 4](#_Toc185325494)

[*Communication Flow* 4](#_Toc185325495)

[4. Installation and Setup 8](#_Toc185325496)

[*Environmental Requirements* 8](#_Toc185325497)

[*Installation Step* 8](#_Toc185325498)

[*Open Game Step* 8](#_Toc185325499)

[5. Usage Guide 8](#_Toc185325500)

[*Game Operations* 8](#_Toc185325501)

[*Game Interface* 9](#_Toc185325502)

[6. External Libraries and Frameworks 9](#_Toc185325503)

[*Node.js* 9](#_Toc185325504)

[*Express* 10](#_Toc185325505)

[*Mongoose* 10](#_Toc185325506)

[*WebSocket* 10](#_Toc185325507)

[*React* 10](#_Toc185325508)

[*CSS* 10](#_Toc185325509)

[7. Known Issues and Limitations 11](#_Toc185325510)

[*Connection stability* 11](#_Toc185325511)

[*Player limit* 11](#_Toc185325512)

[*Insufficient error handling* 11](#_Toc185325513)

[*Interface responsiveness* 12](#_Toc185325514)

[*Database connection problem* 12](#_Toc185325515)

[*Duplicate name* 12](#_Toc185325516)

[8. Testing Plan 13](#_Toc185325517)

[9. GitHub Repository 16](#_Toc185325518)

[10. Demo Video 16](#_Toc185325519)

# Overview

## *Game Introduction*

"Starry Memories(星辰記憶)" is a tarot-based turn-based multiplayer memory battle game for two players online. The game combines memory and strategy, by matching the same tarot card scores, the player with the highest score will win.

## *Technical Architecture*

* The front-end is developed using JavaScript, HTML, and CSS.
* The backend is implemented by Node.js and WebSocket, which supports real-time client-server communication.

## *Game Background Story*

Among the vast stars, each tarot card contains a mysterious memory. By turning over the cards, can you recall their corresponding positions and solve the mystery of memory before your opponent becomes a guardian of the stars?

# Game Play

## *Basic Rules*

* At the beginning of the game, the system will randomly generate 10 tarot cards arranged in a 5x2 grid.
* Players take turns turning over two tarot cards:
  + - If the two tarot cards are the same, the player gets 1 point and continues to flip.
    - If the two tarot cards are different, the cards will be flipped back to the back, and it will be another player’s turn.
* The game ends when all matched tarot cards are turned over.
* The player with the highest score wins.

## *Victory Conditions*

* At the end of the game, the system will automatically calculate the player’s total score, with the highest scorer winning.
* If the scores are the same, there is a tie.

# Communication between clients and server

## *Communication Flow*

* **Clients’ connection:**

When Client opens the link, it can connect to server.

一張含有 文字, 螢幕擷取畫面, 設備, 設計 的圖片

自動產生的描述

* **Set Player Name:**

Players send their names to the server when they join the game.

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

* **Game Start:**

When two players join the game, the server initializes the game and start the game.

一張含有 文字, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

* **Chat Message:**

When a player sends a message, it is transmitted to the server, which then relays the message back to both players and saves it in MongoDB's message database.

* Player 1 sends the message

一張含有 文字, 軟體, 多媒體軟體, 螢幕擷取畫面 的圖片

自動產生的描述

* Player 2 sends the message

一張含有 文字, 軟體, 多媒體軟體, 螢幕擷取畫面 的圖片

自動產生的描述

* **Update Game State:**

When the player flips the card, the action is sent to the server. The server checks the card match, updates the game status, and calculates the score. It generates an update message including the card status, player score, and next player ID, and broadcasts it to both players. After the client receives the message, it updates the interface to display the latest score and card status.

* When player 1 matches the card successfully, then player 1 can match the card again. Player 1's score will add one score, and player 2's card status will show which card is matched.

一張含有 螢幕擷取畫面, 文字, 設計 的圖片

自動產生的描述

* When player 1 matches the card failed, then the round will give player 2.

一張含有 螢幕擷取畫面, 文字, 設計 的圖片

自動產生的描述

* When player 2 matches the card successfully, then player 2 can match the card again. Player 2's score will add one score, and player 1's card status will show which card is matched.

一張含有 文字, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

* **Game Over:**

After all matches are found, the server will send the game completion screen, the total number of rounds and the winner is announced to both players, then it will save the player records in MongoDB's game records database.

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

* **Clients’ Disconnection:**

When a client closes the link, it will disconnect the server. Then, the server will send a message to tell other clients that someone has disconnected.

一張含有 文字, 螢幕擷取畫面, 圓形, 設計 的圖片

自動產生的描述

# Installation and Setup

## *Environmental Requirements*

* Node.js: Requires at least v18.20.4.
* Browser: Modern browsers that support WebSocket (e.g. Chrome, Firefox).

## *Installation Step*

* Install dependencies:
* npm init -y
* npm install express
* npm install mongoose
* npm install react-dom
* npm install react-scripts
* npm install WebSocket
* text {"build": "react-scripts build"} in scripts with package.json
* text {“server": "node server/server.js"} in scripts with package.json

## *Open Game Step*

* Build The Game:
  + npm run build
* Start The Server:
  + npm run server
* Run The Client:
  + <http://localhost:7101/>

# Usage Guide

## *Game Operations*

* Logging into the Game:
  + Use the chat room, enter the name and click the "Send" button in the chat box.
* Game Process:
  + Click on the card face to flip and attempt to match tarot cards.
  + If a match is successful, the score will update automatically.
  + If the match fails, the cards will flip back, and it will be the opponent's turn.
* Viewing Chat Messages:
  + Enter messages in the chat box and click "Send" to communicate with the opponent.

## *Game Interface*

* Card Area:
  + Displays the layout of the tarot cards.
* Info Area:
  + Show the player’s name, the total round, the real-time scores of both players and the round are who.
* Chat Area:
  + For communication with the opponent.

# External Libraries and Frameworks

## *Node.js*

* Description: Node.js is an open-source JavaScript execution environment that enables developers to run JavaScript on the server side. It makes the development of real-time applications such as online games and chat applications easy.
* Purpose: Used to create servers, handle WebSocket connections and database operations.

## *Express*

* Description: Express is a fast, flexible Node.js web application framework that provides a set of powerful yet simple features for building web applications.
* Purpose: Handle HTTP requests, serve static files, and set up a WebSocket server.

## *Mongoose*

* Description: Mongoose is an object modelling tool for MongoDB that provides an intuitive API for Node.js to operate the database.
* Purpose: Used to define data models (such as messages and game records) and perform database operations.

## *WebSocket*

* Description: WebSocket is a communication protocol that provides a full-duplex communication channel for use in real-time applications.
* Purpose: Allows real-time data transmission between the server and the client, supporting real-time interaction in the game.

## *React*

* Description: React is a JavaScript library for building user interfaces, focusing on building reusable UI components.
* Purpose: Used to build the front-end interface of the game, including cards, chat rooms and game information display.

## *CSS*

* Description: Cascading Style Sheets (CSS) is a style sheet language used to describe the display style of HTML documents.
* Purpose: Used to design the appearance and layout of the game to make the interface more attractive and easier to use.

# Known Issues and Limitations

"Starry Memories" game server, while operational and providing an engaging gaming experience, has certain limitations and areas that could benefit from further refinement:

## *Connection stability*

* Description: In some network environments, WebSocket connections may be unstable, causing delays or unexpected disconnections.
* Impact: Players may encounter connection issues, affecting the smoothness and experience of the game.
* Improvement method: Implement a reconnection mechanism to automatically try to reconnect when WebSocket is disconnected. In addition, connection monitoring can be added on the server side to detect and adjust server load in a timely manner.

## *Player limit*

* Description: The current game only supports two players to participate at the same time.
* Impact: Inability to scale to more players, limiting the game's scalability and social interaction.
* Improvement methods: Consider improving the game architecture to support more players. Grouping functions can be designed to allow multiple groups of players to compete at the same time, and statistics will be collected after the game.

## *Insufficient error handling*

* Description: The system lacks clear error prompts for players' incorrect operations (such as repeated investments or non-turn operations).
* Impact: Players may be confused about the game rules, affecting the smoothness of the game.
* Improvement method: Add error prompts and feedback mechanisms to ensure that clear messages are provided immediately when players perform improper operations to help them understand the cause and solution of the error.

## *Interface responsiveness*

* Description: While the game interface is responsive for most devices, display issues may occur on some small screen devices.
* Impact: User experience on small devices may be poor, impacting game accessibility.
* Improvement methods: Conduct more comprehensive responsive design testing to ensure it displays well on different devices. Use CSS media queries and flexible layout design to improve user experience on small screen devices.

## *Database connection problem*

* Description: MongoDB connections may become bottlenecked under high load.
* Impact: There may be delays in reading and writing game data.
* Improvement method: Implement database clustering or caching mechanisms to improve performance under high load conditions and ensure that the database can respond to requests quickly. In addition, monitor database performance and adjust configurations in time to respond to traffic changes.

## *Duplicate name*

* Description: The current version of the game allows players to use duplicate names, which may cause confusion among players.
* Impact: Players cannot clearly identify each other in the game, which may lead to misunderstandings and unnecessary conflicts, affecting the overall game experience.
* Improvement methods: In future versions, the checking mechanism for input names will be strengthened to ensure that each player's name is unique, thereby reducing confusion and improving the playability of the game.

# Testing Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Feature to Test | Test Steps | Expected Result | Tested Result |
| 001 | WebSocket Connection | * Start the server. * Client connects to the server. | Client connects successfully; server shows "User connected, waiting for name..." | Press |
| 002 | Player Name Set | * Client sends name set request. * Server checks number of connected clients. | If there is a free player slot, server responds with "player\_index" and notifies others with "sys\_c\_connect." | Press |
| 003 | Player Name Limit | * Two clients connect and set names. * Third client connects and tries to set a name. | Server responds with "game\_full" and closes the connection for the third client. | Press |
| 004 | Game Start | * Two players have set names and connected. * Server broadcasts "game\_start" message. | Clients receive "game\_start" message and the game starts. | Press |
| 005 | Card Matching Logic | * Player chooses two cards. * Check if the chosen cards match. | If they match, mark cards as matched, increase scores, and keep the player's turn. If not, switch to the next player. | Press |
| 006 | Chat Functionality | * Player sends a chat message. * Server processes the message and broadcasts it. | All clients receive the message, and it shows in the chat window. | Press |
| 007 | Game Over Condition | * All cards are matched. * Server processes "game\_over" message. | All players receive "game\_over" message and see the winner and scores. | Press |
| 008 | New Game Setup | * After the current game ends, players click "Start New Game." | Server broadcasts "game\_start" message and players start a new game. | Press |
| 009 | Database Message Saving | * Player sends a chat message. * Server saves the message to the database. | Message is saved in MongoDB and shows "Message saved" in the console. | Press |
| 010 | Player Disconnection | * One player disconnects. * Server notifies other players. | All connected clients receive "sys\_c\_disconnect" message and update the player list. | Press |
| 011 | Player Ready for New Game | * After the game ends, all players click "Start New Game." | Server checks if all players are ready and starts a new game. | Press |
| 012 | Invalid Card Selection | * Player selects a matched card. * Check how the server handles this. | Server ignores this request and shows a warning in the console. | Press |
| 013 | Player Score Tracking | * Player successfully matches cards. * Check if scores update correctly. | Player scores increase and are updated on all clients. | Press |
| 014 | Player Turn Logic | * Player makes a choice when it's their turn. * Check if it correctly moves to the next player. | Current player's choice is accepted, and it moves to the next player. | Press |
| 015 | Game Record Saving | * Game ends. * Check if all players' scores and rounds are saved to the database. | All game records are saved in MongoDB, and "Game record saved" shows in the console. | Press |
| 016 | Error Handling | * Simulate WebSocket disconnection. * Check how the client handles this. | Client shows an error message and tries to reconnect. | Press |
| 017 | Chat Message Format | * Player sends an incorrectly formatted message. * Check how the server handles this. | Server ignores invalid messages and shows an error in the console. | Press |
| 018 | Multiple Connections | * Attempt to make multiple connections from the same client. * Check how the server handles this. | Server rejects extra connection requests and returns an error message. | Press |
| 019 | Background Music Toggle | * Start background music in the game. * Check if music can be paused and resumed. | Music starts normally and can be paused and resumed based on user actions. | Press |

# GitHub Repository

<https://github.com/SnowYung/ITP4708_Game-Server.git>

# Demo Video

[..\GSDI\_YungKwanWai\_230033193\_VideoDemo\GSDI\_YungKwanWai\_230033193\_VideoDemo.mp4](../GSDI_YungKwanWai_230033193_VideoDemo/GSDI_YungKwanWai_230033193_VideoDemo.mp4)