Game Recommendation System

MR.PHITTAYANAN TIENCHAROEN
MR.ASHIRA SANSODA
MR.DECHAYUT PANYAWUTOVRAKUL

A PROJECT SUBMITTED IN PROGRAMMING WITH DATA STRUCTURES
CPE 112 OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR OF ENGINEERING(COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI 2024

Project Title File	:	System Simulation
Credits	:	3
Member(s)	:	Mr. Phittayanan Tiencharoen
		Mr. Ashira Sansoda
		Mr. Dechayut Panyawutvorakul
Project Advisor	:	Prof. Natasha Dejdumrong, Proj. Naveed Sultan, Prof. Aye Hninn Khine
Program	:	Bachelor of Engineering
Field of Study	:	Computer Engineering
Department	:	Computer Engineering
Faculty	:	Engineering
Academic Year	:	2024

Abstract

This project is a console-based Game Management and Recommendation System developed in C. It provides separate interfaces for administrators and customers. Administrators can manage the game catalog by adding, editing, deleting, and relating games. Customers can log in to browse available games, maintain a shopping cart, and purchase games. The system keeps track of each user's purchase history and recommends new games based on relationships and genre preferences using a breadth-first search (BFS) approach. Logging and activity tracking are recorded daily in CSV format.

Game Recommendation System

1. Problem Statement

Modern digital game stores must manage large catalogs of games, support efficient search and recommendation systems, and maintain user-friendly features such as shopping carts, login systems, and activity logs. Without a proper system in place, managing game relationships, user purchases, and admin operations becomes error-prone and inefficient.

This project aims to design and implement a console-based game management system that supports both admin and customer roles. Admins can add/edit/delete games and define relationships between games (for recommendation purposes), while customers can search for games, manage their shopping carts, make purchases, and receive recommendations based on their purchase history.

2. Solution & Functionalities

Solution Overview

This system aims to simulate a basic game store platform where admins can manage game data, and users can browse, purchase, and receive personalized game recommendations. It also includes logging features for tracking actions.

1. **Admin Functionalities:**
   - 1.1. Add, edit, delete game entries
   - 1.2. Manage game relationships (related games for recommendation)
   - 1.3. View system logs by date (logging by CSV)
2. **Customer Functionalities:**
   - 2.1. Login & view available games
   - 2.2. Add to cart / Remove from cart / Checkout
   - 2.3. View purchase history (by date and item)
   - 2.4. View Shopping Cart
   - 2.5. Personalized recommendations based on Purchase history and Game relationships

3. **Data persistence is handled via CSV files:**
   - **3.1.** games.csv: stores game details
   - **3.2.** relations.csv: stores game-to-game relationship
   - **3.3.** UserHistory/username.csv: individual user purchase logs
   - 3.4. Logging/logging_YYYY-MM-DD.csv: system activity logs

3. Data Structures and Code Walkthrough

3.1 Main

The main.c file serves as the entry point for the system. It loads game data and relationships from CSV files and presents users with a menu to choose between Admin or Customer login. Based on authentication results, users are directed to their respective interfaces.

```c
int main() {
    loadGame("games.csv"); //load game to hash table
    loadRelations("relations.csv"); //load relation to graph
    int c;
    int choice;
    char username[100];

    while (1) {
        displayMainMenu();
        int input;
        scanf("%d",&input);
        while ((c = getchar()) != '\n' && c != EOF);//clear buffer from scanf
        switch (input) {
            case 1: //login
                switch (login(username)) {
                    case 1:
                        customerMenu(username);
                        break;
                    case 2:
                        adminMenu();
                        break;
                    case 3:
                        printf("\nInvalid username or password.\n");
                        printf("Press Enter to continue...");
                        while ((c = getchar()) != '\n' && c != EOF);//wait for enter
                        break;
                    default:
                        printf("\nInvalid choice.\n");
                        printf("Press Enter to continue...");
                        while ((c = getchar()) != '\n' && c != EOF);
                }
                break;
            case 2: //register
                user_register();
                printf("Press Enter to continue...");
                while ((c = getchar()) != '\n' && c != EOF);
                break;
            case 3: //exit
                printf("\nExiting program...\n");
                return 0;
            default:
                printf("\nInvalid choice.\n");
                printf("Press Enter to continue...");
                while ((c = getchar()) != '\n' && c != EOF);
        }

    }
```

3.2 Login

Login functions are separated for Admins and Customers:

Admin login reads from admin.csv, while

Customer login reads from users.csv.

Both perform credential verification and log login events via the logging_event() function

3.3. Interface

```c
int login(char *username){
    CLEAR_SCREEN();
    printf("\n+-----------------------------+\n");
    printf("|             Login           |\n");
    printf("+-----------------------------+\n");

    char input_user[50];
    char input_pass[50];
    char line[256];

    printf("Enter username: ");//input username
    if (fgets(input_user, sizeof(input_user), stdin) == NULL) {
        printf("Error username.\n");
        return 0;
    }
    input_user[strcspn(input_user, "\n")] = 0;

    printf("Enter password: ");//input password
    if (fgets(input_pass, sizeof(input_pass), stdin) == NULL) {
        printf("Error password.\n");
        return 0;
    }
    input_pass[strcspn(input_pass, "\n")] = 0;

    if(user_login(username,input_user,input_pass)){//check user
        return 1;
    }else if(admin_login(input_user,input_pass)){//check admin
        return 2;
    }else{
        return 3;//error
    }
}
```

```c
int admin_login(char *input_user,char *input_pass) {
    FILE *file = fopen("admin.csv", "r");
    if (file == NULL) {
        printf("file not found");
        return 0;
    }

    char line[256];
    int check = 0;

    fgets(line, sizeof(line), file);//skip first line

    while (fgets(line, sizeof(line), file)) {
        char file_user[50];
        char file_pass[50];
        if (sscanf(line, "%[^,],%s", file_user, file_pass) == 2) {
            //checking is username and password are match with admin.csv
            if (strcmp(input_user, file_user) == 0 && strcmp(input_pass, file_pass) == 0) {
                check = 1;
                //logging
                char event[100];
                snprintf(event, sizeof(event), "Admin %s login", input_user);
                logging_event(event, input_user);
                break;
            }
        }
    }

    fclose(file);
    return check;
}
```

```c
int user_login(char *username,char *input_user,char *input_pass) {
    FILE *file = fopen("users.csv", "r");
    if (file == NULL) {
        printf("file not found");
        return 0;
    }

    int check = 0;
    char line[256];

    fgets(line, sizeof(line), file);//skip first line

    while (fgets(line, sizeof(line), file)) {
        char file_user[50];
        char file_pass[50];
        if (sscanf(line, "%[^,],%s", file_user, file_pass) == 2) {
            //checking is username and password are match with user.csv
            if (strcmp(input_user, file_user) == 0 && strcmp(input_pass, file_pass) == 0) {
                check = 1;
                strcpy(username, input_user);
                //logging
                char event[100];
                snprintf(event, sizeof(event), "User %s login", input_user);
                logging_event(event, input_user);
                loadUserPurchaseHistory(username);
                break;
            }
        }
    }
    fclose(file);
    return check;
}
```

```c
int user_register() {
    CLEAR_SCREEN();
    printf("\n+-----------------------------+\n");
    printf("|       User Registration     |\n");
    printf("+-----------------------------+\n");

    FILE *file = fopen("users.csv", "a+");
    if (file == NULL) {
        printf("Error opening user database.\n");
        return 0;
    }

    char new_user[50];
    char new_pass[50];
    char confirm_pass[50];
    char line[256];

    // Get username
    printf("Enter new username: ");
    if (fgets(new_user, sizeof(new_user), stdin) == NULL) {
        printf("Error username.\n");
        fclose(file);
        return 0;
    }
    new_user[strcspn(new_user, "\n")] = 0;

    // Check if username already exists
    while (fgets(line, sizeof(line), file)) {
        char exist_user[50];
        if (sscanf(line, "%[^,],", exist_user) == 1) {
            if (strcmp(new_user, exist_user) == 0) {
                printf("Username already exists. Registration failed.\n");
                fclose(file);
                return 0;
            }
        }
    }

    // Get password
    printf("Enter new password: ");
    if (fgets(new_pass, sizeof(new_pass), stdin) == NULL) {
        printf("Error password.\n");
        fclose(file);
        return 0;
    }
    new_pass[strcspn(new_pass, "\n")] = 0;

    // Confirm password
    printf("Confirm password: ");
    if (fgets(confirm_pass, sizeof(confirm_pass), stdin) == NULL) {
        printf("Error confirm password.\n");
        fclose(file);
        return 0;
```

Admin Interface: Provides options to manage the game catalog, view logs, and add relationships between games.

Customer Interface: Enables users to search games, manage their shopping cart, make purchases, view purchase history, and receive game recommendations.

```c
void customerMenu( char username[]){
    setCart();//set cart value to default
    int input;
    char name[100];
    int c;
    while(1){
        CLEAR_SCREEN();
        printgamelist();//display game list
        printf("\n+-----------------------------+\n");
        printf("|       Customer Menu         |\n");
        printf("+-----------------------------+\n");
        printf("| [1] Search Games            |\n");
        printf("| [2] Add to Cart             |\n");
        printf("| [3] Delete from Cart        |\n");
        printf("| [4] View Cart               |\n");
        printf("| [5] Checkout                |\n");
        printf("| [6] View Purchase History   |\n");
        printf("| [7] Recommend Game          |\n");
        printf("| [8] Back to Main Menu       |\n");
        printf("+-----------------------------+\n");
        printf("Enter your choice: ");

        char input_str[10];
        if (fgets(input_str, sizeof(input_str), stdin) != NULL) {
            if (sscanf(input_str, "%d", &input) == 1) {

                switch (input)
                {
                case 1://Search game
```

```c
void adminMenu() {
    int choice;
    char name[99];
    char genre[99];
    float price;
    char newGenre[99];
    float newPrice;
    char name1[99],name2[99];
    int c;

    while (1) {
        CLEAR_SCREEN();
        printgamelist(); //display list game

        //print admin menu option
        printf("+-----------------------------+\n");
        printf("|         Admin Menu          |\n");
        printf("+-----------------------------+\n");
        printf("| [1] Add Game                |\n");
        printf("| [2] Edit Game               |\n");
        printf("| [3] Delete Game             |\n");
        printf("| [4] Watch Logging           |\n");
        printf("| [5] Add relation            |\n");
        printf("| [6] Back to Main Menu       |\n");
        printf("+-----------------------------+\n");
        printf("Enter your choice: ");

        //read a choice
        char input_str[10];
        if (fgets(input_str, sizeof(input_str), stdin) != NULL)
        {
```

3.4. Game Catalog Management & Search

Data Structure: Hash Table(Chaining)

This function allows the admin to add, edit, delete, and view games stored in a hash table for fast access. It also provides search functionality, using a case-insensitive and space-insensitive comparison to help users easily find the games they're looking for.

```c
// function add game to hash table
void addGame(char name[], char genre[], float price) {
    unsigned int index = hash(name);
    game *newNode = malloc(sizeof(game));

    if (newNode == NULL) {
        printf("Failed create new game node.\n");
        return;
    }
    strcpy(newNode->name, name);
    strcpy(newNode->genre, genre);
    newNode->price = price;
    newNode->next = NULL;
    newNode->relationcount = 0;

    if (hashIndex[index] == NULL) {
        hashIndex[index] = newNode;
    } else {
        // traverse to the end of the linked list
        game *current = hashIndex[index];
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode; // add the new node at the end of the list
    }
    printf("Game '%s' added successfully.\n", name);

    saveGameToCSV("games.csv"); //call saveGameTOCSV to write game to games.csv file
}
```

```c
// fuction to delete game in hash table
void deleteGame(char name[]) {
    unsigned int index = hash(name);
    game *current = hashIndex[index];
    game *prev = NULL;

    while (current != NULL) {

        if (!strcasecmp(current->name, name)) {

            if (prev == NULL) {

                hashIndex[index] = current->next;
            } else {

                prev->next = current->next;
            }
            free(current);
            printf("Game '%s' deleted successfully.\n", name);
            char deleteGame[100];
            snprintf(deleteGame, sizeof(deleteGame), "Delete %s", name);
            logging_event(deleteGame,"Admin");

            saveGameToCSV("games.csv"); //call fuction saveGameTOCSV to save

            return;
        }
        prev = current;
        current = current->next;
    }
    printf("Game '%s' not found.\n", name);
}
```

```c
case 1://Search game
                    printf("\n--- Search Game ---\n");
                    printf("Enter game to search: ");
                    fgets(name, sizeof(name), stdin);
                    name[strcspn(name, "\n")] = '\0'; //remove newline character
                    game *found = findGame(name);
                    if (found != NULL){
                        printf("\nFound: %s (%s, $%.2f)\n", found->name, found->genre, found->price);
                    } else {
                        printf("\nGame '%s' not found.\n", name);
                    }
                    printf("\nPress Enter to continue...");
                    while ((c = getchar()) != '\n' && c != EOF);
                    break;
```

3.5. Customer Shopping Cart

Data Structure: Linked List

Implements a linked-list-based cart system where customers can add, view, and remove games before checkout. The system calculates the total price and maintains purchase records per user

```c
// Add a game to the shopping cart if it's not already in and cart is not full
void addtoCart(char name[]){
    if(cart.count >= max_cart) {
        printf("Cart is full! Maximum %d items allowed.\n", max_cart);
        return;
    }
    game *found = findGame(name);
    if(found == NULL) {
        printf("Game '%s' not found!\n", name);
        return;
    }

     // Check for duplicates
    CartItem* current = cart.front;
    while (current != NULL) {
        if (compareWithoutspaces(current->game->name, name) == 0) {
            printf("Game '%s' is already in the cart.\n", name);
            return;
        }
        current = current->next;
    }

    // Allocate and append new cart item
    CartItem* newItem = malloc(sizeof(CartItem));
    if (newItem == NULL) {
        printf("fail create new item");
        return;
    }
    newItem->game = found;
    newItem->next = NULL;

    if(cart.front == NULL) {
        cart.front = newItem;
        cart.rear = newItem;
    } else {
        cart.rear->next = newItem;
        cart.rear = newItem;
    }
    cart.count++;
    cart.total+=found->price;

    printf("Added to cart: %s ($%.2f)\n", found->name, found->price);
    printf("Cart now has %d items (Total: $%.2f)\n", cart.count, cart.total);
}
```

```c
// Remove a game from the cart by name
void deletefromCart(char name[]){
    if(cart.count == 0) {
        printf("Cart is empty!\n");
        return;
    }
    CartItem *current=cart.front;
    CartItem *prev=NULL;
    int found = 0;

    // Traverse cart to find the game
    while(current!=NULL){
        if(!compareWithoutspaces(current->game->name,name)){
            // Remove node from linked list
            if(prev==NULL){
                cart.front=current->next;
            }else{
                prev->next=current->next;
            }
            if(current == cart.rear) {
                cart.rear = prev;
            }
            cart.count--;
            cart.total -= current->game->price;
            printf("Removed from cart: %s\n", current->game->name);
            printf("Cart now has %d items (Total: $%.2f)\n", cart.count, cart.total);
            free(current);
            found = 1;
            break;
        }
        prev = current;
        current = current->next;
    }
    if (!found) {
         printf("Game '%s' not found in the cart.\n", name);
    }
}
```

```c
// Display all games currently in the shopping cart
void viewCart(){
    // Clear the screen for clean output
    CLEAR_SCREEN();

    // Head
    printf("\n+-----------------------------------------------------------------+\n");
    printf("|                    Your Shopping Cart                           |\n");
    printf("+-----------------------------+------------------+----------+\n");

    //if cart empty
    if(cart.count == 0) {
        printf("|Your cart is empty                                               |\n");
        printf("+-----------------------------------------------------------------+\n");
        return;
    }

    //table head
    printf("| %-28s | %-18s | %-8s |\n", "Game", "Genre", "Price");
    printf("+-----------------------------+------------------+----------+\n");

    // Display each item in cart
    CartItem* current = cart.front;
    while(current != NULL) {
        printf("| %-28s | %-18s | %-8.2f |\n", current->game->name,current->game->genre,current->game->price);
        current = current->next;
    }

    //Bottom with total Price
    printf("+-----------------------------+------------------+----------+\n");
    printf("|Total: $%-54.2f|\n", cart.total);
    printf("+-----------------------------------------------------------------+\n\n");
}
```

3.6. Relations

Data Structure: Graph

Use addRelation to Establishes links between games that are related or complementary. If writeToFile is true, the relation is saved in relations.csv and logged. Duplicate links are avoided, and bidirectional links are created during admin operations.and use loadRelation to Reads from relations.csv on startup and builds relation pointers between games.

```c
// Add a relation between two games if not already related
void addRelation(char name1[], char name2[], int writeToFile) {
    name1[strcspn(name1, "\n")] = 0;
    name2[strcspn(name2, "\n")] = 0;

    game *game1 = findGame(name1);
    game *game2 = findGame(name2);

    //when one game don't have in games.csv
    if (!game1 || !game2) {
        printf("game1 = %s game2 = %d\n",game1,game2);
        printf(" One or both games not found.\n");
        return;
    }
    //when add same game name
    if (game1 == game2) {
        printf(" Cannot relate a game to itself.\n");
        return;
    }

    // Check for existing relation
    for (int i = 0; i < game1->relationcount; ++i) {
        if (game1->related[i] == game2) {
            if (writeToFile) {
                printf("Relation already exists: %s <-> %s\n", name1, name2);
            }
            return;
        }
    }


    // Add relation if under limit
    if (game1->relationcount < max_relation) {
        game1->related[game1->relationcount++] = game2;

        // Save to CSV and log
        if (writeToFile) {
            FILE *fp = fopen("relations.csv", "a");
            if (fp != NULL) {
                fprintf(fp, "%s,%s\n", name1, name2);
                fclose(fp);
            }

            char logMsg[200];
            snprintf(logMsg, sizeof(logMsg), "Relation added: %s <-> %s", name1, name2);
            logging_event(logMsg, "Admin");

            printf("Relation added: %s <-> %s\n", name1, name2);
        }
    } else {
        printf("Max relation reached for '%s'\n", name1);
    }
}

```

```c
//load relation to graph
void loadRelations(char filename[]) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        printf(" Warning: Could not load game relations from '%s'.\n", filename);
        return;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        char game1_name[100], game2_name[100];
        if (sscanf(line, "%[^,],%[^\n]", game1_name, game2_name) == 2) {
            addRelation(game1_name, game2_name,0);
            addRelation(game2_name, game1_name,0);
        }
    }
    fclose(file);
}
```

3.7. Recommendations

Data Structure: BFS(Graph+Queue)

Recommendations BFS-Based Suggestion Uses Breadth-First Search to traverse the game network based on relationships. Filters out already purchased games. Scores are assigned based on: Distance from purchased games Frequency of genre among past purchases Sorting Logic Recommendations are sorted by genre preference first, then by distance.

```c
// Recommend games based on the user's purchase history and game relations
void recommendBasedOnHistory(const char* username) {
    CLEAR_SCREEN();
    // Load user purchase history if it not already loaded for this user
    if (currentUserPurchase == NULL || strcmp(currentUserPurchase->username, username) != 0) {
        loadUserPurchaseHistory(username);
    }
    // If user has no purchase history, show a message and return
    if (currentUserPurchase->purchaseCount == 0) {
        printf("No recommendation now. Try to buy some games.\n");
        return;
    }
    setVisited();// Reset visited flags for BFS

    // Struct for genre frequency
    typedef struct {
        char genre[99];
        int count;
    } GenreFrequency;

    GenreFrequency genreFreq[20] = {0};
    int genreCount = 0;

    // find genre frequency
    for (int i = 0; i < currentUserPurchase->purchaseCount; i++) {
        int found = 0;
        // Find if the genre is already in the frequency list
        for (int j = 0; j < genreCount; j++) {
            if (strcmp(genreFreq[j].genre, currentUserPurchase->purchasedGames[i]->genre) == 0) {
                genreFreq[j].count++;
                found = 1;
                break;
            }
        }
        // If the genre is not found add it
        if (!found && genreCount < 20) {
            strcpy(genreFreq[genreCount].genre, currentUserPurchase->purchasedGames[i]->genre);
            genreFreq[genreCount].count = 1;
            genreCount++;
        }
    }

    // Sort genres by frequency (bubble sort)
    for (int i = 0; i < genreCount - 1; i++) {
        for (int j = 0; j < genreCount - i - 1; j++) {
            if (genreFreq[j].count < genreFreq[j+1].count) {
                GenreFrequency temp = genreFreq[j];
                genreFreq[j] = genreFreq[j+1];
                genreFreq[j+1] = temp;
            }
        }
    }

```

```c
    // Find recommended games (BFS from all purchased games)
    Queue* front = NULL;
    Queue* rear = NULL;

    // Mark all purchased games as visited (distance 0)
    for (int i = 0; i < currentUserPurchase->purchaseCount; i++) {
        game* purchased = currentUserPurchase->purchasedGames[i];
        purchased->visited = 1;
        enqueue(&front, &rear, purchased);
    }

    // Struct for recommendations with distance and score
    typedef struct {
        game* game;
        int distance;
        int genreScore;
    } Recommendation;

    Recommendation recommendations[max_relation * 2];
    int recCount = 0;

    //BFS
    while (front != NULL) {
        game* current = dequeue(&front, &rear);
        int currentDistance = current->visited - 1;//game that owned distance will be 0

        if (currentDistance > 0) {
            int genreScore = 0;
            for (int i = 0; i < genreCount; i++) {
                if (strcmp(current->genre, genreFreq[i].genre) == 0) {
                    genreScore = genreCount - i; // Higher score for more frequent genres
                    break;
                }
            }

            recommendations[recCount].game = current;
            recommendations[recCount].distance = currentDistance;
            recommendations[recCount].genreScore = genreScore;
            recCount++;
        }
        for (int i = 0; i < current->relationcount && i < max_relation; i++) {
            if (current->related[i] != NULL && !current->related[i]->visited) {
                current->related[i]->visited = current->visited + 1;
                enqueue(&front, &rear, current->related[i]);
            }
        }
    }

    // Sort by genre score (primary) and distance (secondary)
    for (int i = 0; i < recCount - 1; i++) {
        for (int j = 0; j < recCount - i - 1; j++) {
            if (recommendations[j].genreScore < recommendations[j+1].genreScore ||
                (recommendations[j].genreScore == recommendations[j+1].genreScore &&
                 recommendations[j].distance > recommendations[j+1].distance)) {
                Recommendation temp = recommendations[j];
                recommendations[j] = recommendations[j+1];
                recommendations[j+1] = temp;
            }
        }
    }

    // Display
    printf("+-------------------------------------------------------------+\n");
    printf("|                    Game Recommendations                     |\n");
    printf("+--------------------+--------------------+-----------+\n");
    printf("|%-30s|%-20s|%-10s|\n", "Name", "Genre", "Price");
    printf("+--------------------+--------------------+-----------+\n");

    int maxToShow = 5;
    for (int i = 0; i < recCount && i < maxToShow; i++) {
        printf("|%-30s|%-20s|$%-9.2f|\n", recommendations[i].game->name, recommendations[i].game->genre, recommendations[i].game->price);
    }

    if (recCount == 0) {
        printf("No recommendations found.\n");
    }

    printf("+-------------------------------------------------------------+\n");

    char logMessage[200];
    snprintf(logMessage, sizeof(logMessage), "Viewed personalized recommendations");
    logging_event(logMessage, username);
}

//Display user's purchase history
void printPurchaseHistory(const char* username) {
    CLEAR_SCREEN();
    char filename[150];
    snprintf(filename, sizeof(filename), "UserHistory/%s.csv", username);

    FILE* file = fopen(filename, "r");
    if (!file) {
        printf("No purchase history found for %s\n", username);
        return;
    }
    printf("+-------------------------------------------------------------------------+\n");
    printf("|                        Purchase History                                 |\n");
    printf("+--------------------+--------------------+-----------+----------+\n");
    printf("|%-30s|%-20s|%-10s|%-12s|\n", "Game", "Genre", "Price", "Date");
    printf("+--------------------+--------------------+-----------+----------+\n");

    char line[256];
```

3.8. Logging System: Files (CSV)

Data Storage: Files (.csv files in Logging and UserHistory folders).

Logging use Logs are saved to Logging/logging_YYYY-MM-DD.csv. Each entry includes the date, time, event description, and username. and view by display_logging to Allows admins to view activity logs for a specific day in a formatted table.

```c
// The result will be something like "Logging/logging_2025-04-28.csv"
void CreateLogbyDate(char *filename, size_t size, const char *date)
{
    // Format: Logging/logging_<date>.csv
    snprintf(filename, size, LOG_FOLDER"/logging_%s.csv", date);
}
```

```c
//function Display Logging
void display_logging(const char *date)
{

    // Construct the log file path based on the provided date
    char filename[256];
    CreateLogbyDate(filename, sizeof(filename), date);

    // Open the log file for reading
    FILE *fp = fopen(filename, "r");
    if (fp == NULL)
    {
        printf("Can't open file: %s\n", filename);
        return;
    }
    // Print head of logging
    char line[512];
    printf("+-------------------------------------------------------------------------------------+\n");
    printf("|                              System Logging: %s                             |\n", date);
    printf("|----------+-----------------------------------------------------------+----------|\n");
    printf("|%-10s|%-61s|%-10s|\n", "Time", "Activity", "User");
    printf("|----------+-----------------------------------------------------------+----------|\n");

    // Skip the CSV head line
    fgets(line, sizeof(line), fp);

    // Read and parse each log line
    while (fgets(line, sizeof(line), fp))
    {
        char tempdate[10], time[10],activity[100], user[20];

        // Parse the CSV line into individual fields
        if (sscanf(line, "%[^,],%[^,],%[^,],%s",tempdate, time, activity, user) == 4) {

            // Display the parsed log entry in table format
            printf("|%-10s|%-61s|%-10s|\n", time, activity, user);
        }
    }

    // Print closing table border
    printf("+-------------------------------------------------------------------------------------+\n");

    // Close the file
    fclose(fp);
}
```

```c
// Function to log an event with associated user information to a file
void logging_event(const char *event, const char *user)
{

    // Get the current time
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    char today[11];
    snprintf(today, sizeof(today), "%04d-%02d-%02d",
        t->tm_year + 1900,
        t->tm_mon + 1,
        t->tm_mday);

    char filename[256];
    CreateLogbyDate(filename, sizeof(filename),today);

    // Open the logging file in append mode
    FILE *fp = fopen(filename,"a+");
    if(fp == NULL)
    {
        // Print an error if the file cannot be opened
        printf("Can't open file: %s\n",filename);
        return;
    }

    //Check is Empty file or not if Empty Add Head of file
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);

    if (size == 0) {
        fprintf(fp, "Date,Time,Activity,User\n"); //Set Header of CSV
    }

    fseek(fp, 0, SEEK_END);


    // Format and print the log entry to the file
    fprintf(fp, "%04d-%02d-%02d,%02d:%02d:%02d,%s,%s\n",
        t->tm_year + 1900,
        t->tm_mon + 1,
        t->tm_mday,
        t->tm_hour,
        t->tm_min,
        t->tm_sec,
        event,
        user);

    // Close the file
    fclose(fp);
}
```

4. Time Complexity Analysis Summary

1. **Hash Table (Game Catalog):**
   1.1. Average: O(1) for add, find, edit, delete.
   1.2. Worst: O(n) for add, find, edit, delete.
   1.3. View All: O(n).
2. **Linked List (Shopping Cart):**
   2.1. Add: O(1).
   2.2. Delete: O(c) (where c is cart size).
   2.3. View/Checkout: O(c).
3. **Arrays (Relations/History):**
   3.1. Add: O(1) avg, O(r/h) worst (with duplicate check).
   3.2. View History: O(h) (where h is history size).
4. **Queue (for BFS):**
   4.1. Enqueue/Dequeue: O(1).
5. **Recommendation (BFS + Sorting):**
   5.1. Overall: Dominated by BFS O(V + E) and sorting O(k log k) or O(k^2).
6. **Files (Logging/History):**
   6.1. Writing (Append): O(L) (where L is log entry length).
   6.2. Reading (Display): O(S) (where S is file size).

5. Team Member Responsibilities

1. **Phittayanan Tiencharoen:** Admin Interface, Game Catalog Management (Hash Table implementation, Add/Edit/Delete/View Game functionalities)
2. **Ashira Sansoda:** Customer Interface, Shopping Cart (Linked List implementation, Add/Delete/View Cart, Checkout functionalities), Purchase History (Array implementation, Record/View History), Game Recommendations (Queue for BFS, Recommendation logic)
3. **Dechayut Panyawutvorakul:** Logging System (File storage implementation, Logging events, Displaying logs), File-based Data Persistence (Saving/Loading games, history, relations), Relation (Graph implementation)

6. References

- https://www.geeksforgeeks.org/hashing-data-structure/
- https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/