game.cpp

```cpp
/*************************************************************
 *
 * game.cpp
 *
 * Implementation of game.h
 *
 * Briano Goestiawan, 31482228
 *
*************************************************************/

// TODO:
// - Show leaderboard
// - Player move count
// - Write a reflection document 300-500 words

#include <iostream>
#include <sstream>
#include <algorithm>
#include "game.h"
#include "player.h"
#include "item.h"
#include "utils.h"
#define ROOM_ROWS 4
#define ROOM_COLS 3
#define ROOM_WIDTH 25    // Not including border
#define WINDOW_WIDTH 77  // Not including border
#define WINDOW_HEIGHT 39 // Not including border

using namespace std;
string readFile(string fileName);
string readInput(string prompt);

/****************************** PUBLIC METHODS ******************************/

Game::Game(string playerName, Difficulty difficulty) {
    // Create and allocate game objects
    this->createRooms();
    this->createItems();
    this->createSuspects();

    // Create player instance
    Room startingRoom = this->rooms[6];
    this->player = Player(playerName, &startingRoom);
    this->inventory = Room("Inventory");

    // Initialize variables
    this->view = VIEW_TOWER;
    this->difficulty = difficulty;
    this->gameOver = false;
    this->foundKiller = false;
}
```

```
// Display help screen. Wait for user before continuing
void Game::showStoryLine() {
    clearScreen();
    cout << readFile("assets/story_line.txt");
    pause();
}

// Display help screen. Wait for user before continuing
void Game::showHelpScreen() {
    clearScreen();
    cout << readFile("assets/help_screen.txt");
    pause();
}

// Draw image to screen
void Game::displayView() {
    clearScreen();
    switch (this->view) {
        case VIEW_TOWER:
            this->displayTower();
            break;
        case VIEW_ROOM:
            this->displayRoom();
            break;
        case VIEW_INVENTORY:
            this->displayInventory();
            break;
    }
}

// Validate and process user input
void Game::command() {
    // Anatomy of a command:
    //     command    argument (can be multiple words)
    //     \_____/    \_____________________________/

    // Read input from user
    string input = toLower(readInput("//> "));
    string command = "";
    string argument = "";
    int spaceIndex = input.find(' ');
    if (spaceIndex == string::npos) {
        command = input.substr(0, spaceIndex);
    } else {
        command = input.substr(0, spaceIndex);
        argument = input.substr(spaceIndex + 1);
    }

    // Game view
    if (command == "view") this->cycleView();
    else if (command == "tower") this->view = VIEW_TOWER;
    else if (command == "room") this->view = VIEW_ROOM;
    else if (command == "inv") this->view = VIEW_INVENTORY;
```

```
    // Movement
    else if (command == "left") this->player.move(DIR_LEFT);
    else if (command == "right") this->player.move(DIR_RIGHT);
    else if (command == "up") this->player.move(DIR_UP);
    else if (command == "down") this->player.move(DIR_DOWN);

    // Item Interactions
    else if (command == "search") this->player.getRoom()->search();
    else if (command == "pickup") this->pickup(argument);
    else if (command == "drop") this->drop(argument);
    else if (command == "inspect") this->inspect(argument);

    // Supect Interactions
    else if (command == "talk") this->talk(argument);
    else if (command == "gather") this->gather();
    else if (command == "accuse") this->accuse(argument);

    // Utility commands
    else if (command == "help") this->showHelpScreen();
    else if (command == "easter") cout << "egg" << endl, this->command();
    else if (command == "quit") this->confirmQuit();
    else this->invalidCommand();
}

bool Game::getFoundKiller() {
    return this->foundKiller;
}

bool Game::getGameOver() {
    return this->gameOver;
}

/****************************** PRIVATE METHODS *******************************/

// Generate room objects
void Game::createRooms() {
    // Create room instances and append to this->rooms
    this->rooms.push_back(Room("CONTROL CENTER", "assets/room_control_center.txt"));
    this->rooms.push_back(Room("OFFICE",         "assets/room_office.txt"));
    this->rooms.push_back(Room("SPA",            "assets/room_spa.txt"));
    this->rooms.push_back(Room("LABORATORY",     "assets/room_laboratory.txt"));
    this->rooms.push_back(Room("LIBRARY",        "assets/room_library.txt"));
    this->rooms.push_back(Room("GIFT SHOP",      "assets/room_gift_shop.txt"));
    this->rooms.push_back(Room("CAFETARIA",      "assets/room_cafetaria.txt"));
    this->rooms.push_back(Room("LOBBY",          "assets/room_lobby.txt"));
    this->rooms.push_back(Room("TOILET",         "assets/room_toilet.txt"));
    this->rooms.push_back(Room("SERVER ROOM",    "assets/room_server_room.txt"));
    this->rooms.push_back(Room("CAR PARK",       "assets/room_car_park.txt"));
    this->rooms.push_back(Room("PLUMBING ROOM",  "assets/room_plumbing_room.txt"));

    // Set one of the room to be the murder room
    this->murderRoom = &this->rooms[rand() % this->rooms.size()];
    this->murderRoom->addBlood();
```

```
    // Set room neighbours. for each room in rooms, set its left, right, up and
    // down neighbouring room if possible (not wall)
    for (int row = 0; row < ROOM_ROWS; row++) {
        for (int col = 0; col < ROOM_COLS; col++) {
            int index = row * ROOM_COLS + col;

            // Set left
            if (col > 0) {
                this->rooms[index].setNeighbour(DIR_LEFT, &this->rooms[index - 1]);
            }

            // Set right
            if (col < ROOM_COLS - 1) {
                this->rooms[index].setNeighbour(DIR_RIGHT, &this->rooms[index + 1]);
            }

            // Set up
            if (row > 0) {
                this->rooms[index].setNeighbour(DIR_UP, &this->rooms[index - ROOM_COLS]);
            }

            // Set down
            if (row < ROOM_ROWS - 1) {
                this->rooms[index].setNeighbour(DIR_DOWN, &this->rooms[index + ROOM_COLS]);
            }
        }
    }
}

// Generate items objects
void Game::createItems() {
    // Create items
    this->items.push_back(Item("Knife", "assets/item_knife.txt"));
    this->items.push_back(Item("Fork", "assets/item_fork.txt"));
    this->items.push_back(Item("Stick", "assets/item_stick.txt"));
    this->items.push_back(Item("Scissors", "assets/item_scissors.txt"));
    this->items.push_back(Item("Bowling ball", "assets/item_bowling_ball.txt"));
    this->items.push_back(Item("Screwdriver", "assets/item_screwdriver.txt"));
    this->items.push_back(Item("Chair", "assets/item_chair.txt"));
    this->items.push_back(Item("Vase", "assets/item_vase.txt"));

    // Set one of the item to be the murder weapon
    this->murderWeapon = &this->items[rand() % this->items.size()];
    this->murderWeapon->addBlood();

    // Put all item in items to a randomly selected room
    for (int i = 0; i < this->items.size(); i++) {
        this->items[i].setRoom(this->getRandomRoom());
    }
}

// Generate suspects objects
void Game::createSuspects() {
    // Create suspects
```

```
    this->suspects.push_back(Suspect("Anna"));
    this->suspects.push_back(Suspect("Bob"));
    this->suspects.push_back(Suspect("Charlie"));
    this->suspects.push_back(Suspect("Daniel"));
    this->suspects.push_back(Suspect("Emma"));
    this->suspects.push_back(Suspect("Felix"));
    this->suspects.push_back(Suspect("George"));

    // Put all suspects to a randomly selected room
    for (int i = 0; i < this->suspects.size(); i++) {
        this->suspects[i].setRoom(this->getRandomRoom());
    }

    // Create a vector of pointer to all suspects (for random picking purpose)
    vector<Suspect*> suspectCollection;
    for (int i = 0; i < this->suspects.size(); i++) {
        suspectCollection.push_back(&this->suspects[i]);
    }

    // Shuffle suspect collection
    for (int i = 0; i < suspectCollection.size(); i++) {
        Suspect *temp = suspectCollection[i];
        int randomIndex = rand() % suspectCollection.size();
        suspectCollection[i] = suspectCollection[randomIndex];
        suspectCollection[randomIndex] = temp;
    }

    // Assign killer
    this->killer = suspectCollection.back();
    this->killer->setType(SUS_KILLER);
    this->killer->setAlibi(suspectCollection[rand() % suspectCollection.size()]);
    suspectCollection.pop_back();

    // Assign victim
    suspectCollection.back()->setType(SUS_VICTIM);
    suspectCollection.pop_back();

    // Set alibi pairs
    while (suspectCollection.size() > 1) {
        Suspect *suspectA = suspectCollection.back();
        suspectCollection.pop_back();
        Suspect *suspectB = suspectCollection.back();
        suspectCollection.pop_back();

        suspectA->setAlibi(suspectB);
        suspectB->setAlibi(suspectA);
    }
}

// Cycle through all the different views
void Game::cycleView() {
    switch (this->view) {
        case VIEW_TOWER:
            this->view = VIEW_ROOM;
```

```
            break;
        case VIEW_ROOM:
            this->view = VIEW_INVENTORY;
            break;
        case VIEW_INVENTORY:
            this->view = VIEW_TOWER;
            break;
    }
}

// Display the tower including the player character where it is located in the tower
void Game::displayTower() {
    // Print tower roof
    cout << "   +-------------------------------------------------------------------------------+   " << endl;
    cout << "  /                                                                               \ " << endl;
    cout << " /                           B R U M P   T O W E R                               \" << endl;
    cout << "|                                                                                 |" << endl;
    cout << "+---------------------------------------------------------------------------------+" << endl;

    // For each floor in the tower
    for (int row = 0; row < ROOM_ROWS; row++) {

        // Status bar on top of each room
        cout << '|';
        for (int col = 0; col < ROOM_COLS; col++) {
            // Display a * for each item in room
            Room *currentRoom = &this->rooms[row * ROOM_COLS + col];
            string itemString = "";
            if (currentRoom->getItemHidden()) {
                itemString = "? ";
            } else {
                for (int i = 0; i < this->items.size(); i++) {
                    if (items[i].getRoom() == currentRoom) {
                        itemString += "* ";
                    }
                }
            }

            // Display the first letter of each suspect in the room
            string suspectString = "";
            for (int i = 0; i < this->suspects.size(); i++) {
                if (this->suspects[i].getRoom() == currentRoom) {
                    suspectString += ' ' + this->suspects[i].getName().substr(0, 1);
                }
            }

            // Example display format: | A B C D E F G * * * * * * |
            int blankCount = ROOM_WIDTH - itemString.length() - suspectString.length();
            cout << fixedWidth(suspectString + fixedWidth("", ' ', blankCount) + itemString, ' ', ROOM_
        }
        cout << '
';

        // Display the main section of the room (where the character might be)
```

```
        for (int i = 0; i < 6; i++) {
            cout << '|';
            for (int col = 0; col < ROOM_COLS; col++) {
                string content = "";

                Room *currentRoom = &this->rooms[row * ROOM_COLS + col];
                if (currentRoom == this->player.getRoom()) {
                    content = this->player.getImage()[i];
                }

                cout << fixedWidth(content, ' ', ROOM_WIDTH) << '|';
            }
            cout << '
';
        }

        // Print names of rooms in the current floor
        cout << '|';
        for (int col = 0; col < ROOM_COLS; col++) {
            string roomName = this->rooms[row * ROOM_COLS + col].getName();
            cout << fixedWidth(" " + roomName, ' ', ROOM_WIDTH) << '|';
        }
        cout << '
';

        cout << "+-------------------------------------------------------------------------------+" << en
    }
}

// Display the room where the player is in
void Game::displayRoom() {
    Room *currentRoom = this->player.getRoom();

    // Display the current room's image and description
    cout << currentRoom->getImage();
    cout << "|                                                                               |" << endl;

    // Display each item that exists in the current room
    string itemsString = "  ITEMS: ";
    if (currentRoom->getItemHidden()) {
        itemsString += '?';
    } else {
        vector<Item *> items;
        for (int i = 0; i < this->items.size(); i++) {
            if (this->items[i].getRoom() == currentRoom) {
                items.push_back(&this->items[i]);
            }
        }
        if (items.size() > 0) {
            itemsString += items[0]->getName();
            for (int i = 1; i < items.size(); i++) {
                itemsString += ", " + items[i]->getName();
            }
        } else {
```

```
            itemsString += '-';
        }
    }
    cout << '|' << fixedWidth(itemsString, ' ', WINDOW_WIDTH) << '|' << endl;
    cout << "|                                                                                |" << endl;

    // Display all suspect in the current room
    string suspectsString = "  SUSPECTS: ";
    vector<Suspect *> suspects;
    for (int i = 0; i < this->suspects.size(); i++) {
        if (this->suspects[i].getRoom() == currentRoom) {
            suspects.push_back(&this->suspects[i]);
        }
    }
    if (suspects.size() > 0) {
        suspectsString += suspects[0]->getName();
        for (int i = 1; i < suspects.size(); i++) {
            suspectsString += ", " + suspects[i]->getName();
        }
    } else {
        suspectsString += '-';
    }
    cout << '|' << fixedWidth(suspectsString, ' ', WINDOW_WIDTH) << '|' << endl;
    cout << "|                                                                                |" << endl;

    cout << "+--------------------------------------------------------------------------------+" << endl;
}

// Display player's inventory full screen
void Game::displayInventory() {
    cout << "+--------------------------------------------------------------------------------+" << endl;

    // Print title
    cout << '|' << fixedWidth("", ' ', WINDOW_WIDTH) << '|' << endl;
    cout << '|' << fixedWidth("  INVENTORY", ' ', WINDOW_WIDTH) << '|' << endl;

    // Print items in player's inventory
    for (int i = 0; i < this->getInventory().size(); i++) {
        cout << '|' << fixedWidth("", ' ', WINDOW_WIDTH) << '|' << endl;
        cout << '|' << fixedWidth("  - " + this->getInventory()[i]->getName(), ' ', WINDOW_WIDTH) << '|
    }

    // Print blank lines
    for (int i = 0; i < WINDOW_HEIGHT - this->getInventory().size() * 2 - 2; i++) {
        if (i == 1 && this->getInventory().size() == 0) {
            // Print empty. if nothing in inventory
            cout << '|' << fixedWidth("  Empty.", ' ', WINDOW_WIDTH) << '|' << endl;
        } else {
            cout << '|' << fixedWidth("", ' ', WINDOW_WIDTH) << '|' << endl;
        }
    }

    cout << "+--------------------------------------------------------------------------------+" << endl;
}
```

```
// Get confirmation from user if they want to quit
void Game::confirmQuit() {
    string answer = readInput("Are you sure? (Y/n) ");
    if (answer == "y" || answer == "Y") {
        this->gameOver = true;
    }
}

// If the user puts an invalid command suggest them to read the help screen
void Game::invalidCommand() {
    cout << "Get some '//> help'
" << endl;
    this->command();  // Prompt for command again. No re-render
}

// Display a response from the suspect named suspectName
void Game::talk(string suspectName) {
    Suspect *suspect = this->searchSuspect(suspectName);
    if (suspect && suspect->getRoom() == this->player.getRoom()) {
        suspect->talk(this->player.getName());
    }
    /* int suspectIndex = this->searchSuspect(suspectName); */

    /* if (suspectIndex >= 0) { */
    /*     if (this->suspects[suspectIndex].getRoom() == this->player.getRoom()) { */
    /*         suspects[suspectIndex].talk(this->player.getName()); */
    /*     } */
    /* } */

    this->command();
}

// Move all suspect to the room where the player is in
void Game::gather() {
    for (int i = 0; i < this->suspects.size(); i++) {
        this->suspects[i].setRoom(this->player.getRoom());
    }
}

// Return a pointer to a random room
Room *Game::getRandomRoom() {
    return &this->rooms[rand() % this->rooms.size()];
}

// Return a pointer to the room named roomName and return NULL if not found
Room *Game::searchRoom(std::string roomName) {
    for (int i = 0; i < this->rooms.size(); i++) {
        if (toLower(this->rooms[i].getName()) == toLower(roomName)) {
            return &this->rooms[i];
        }
    }

    return NULL;
```

```
}

// Return a pointer to the suspect named suspectName and return NULL if not found
Suspect *Game::searchSuspect(string suspectName) {
    for (int i = 0; i < this->suspects.size(); i++) {
        if (toLower(this->suspects[i].getName()) == toLower(suspectName)) {
            return &this->suspects[i];
        }
    }

    return NULL;
}

// Return a pointer to an item named itemName and return NULL if not found
Item *Game::searchItem(string itemName) {
    for (int i = 0; i < this->items.size(); i++) {
        if (toLower(this->items[i].getName()) == toLower(itemName)) {
            return &this->items[i];
        }
    }

    return NULL;
}

// Search for item in room add to inventory if found
void Game::pickup(string itemName) {
    Item *item = this->searchItem(itemName);

    if (item && item->getRoom() == this->player.getRoom()) {
        item->setRoom(&this->inventory);
    } else {
        this->command();
    }
}

// Drop item named itemName to player's room if found in inventory
void Game::drop(string itemName) {
    Item *item = this->searchItem(itemName);

    if (item && item->getRoom() == &this->inventory) {
        item->setRoom(this->player.getRoom());
    } else {
        this->command();
    }
}

// Display item named itemName in player's inventory if exists
void Game::inspect(string itemName) {
    Item *item = this->searchItem(itemName);

    if (item && item->getRoom() == &this->inventory) {
        clearScreen();
        cout << item->getImage();
        cout << "|                                                                                                                    |" << end
```

```
        cout << '|' << fixedWidth("  ITEM NAME: " + item->getName(), ' ', WINDOW_WIDTH) << '|' << endl;
        cout << "|                                                                                |" << endl;
        cout << '|' << fixedWidth("  LOCATION: " + item->getRoom()->getName(), ' ', WINDOW_WIDTH) << '|' << endl;
        cout << "|                                                                                |" << endl;
        cout << "+--------------------------------------------------------------------------------+" << endl;
        pause();
    } else {
        this->command();
    }
}

void Game::accuse(string suspectName) {
    // TODO better behaviour

    Suspect *suspect = this->searchSuspect(suspectName);
    if (suspect) {
        if (suspect == this->killer) {
            cout << "That is infact the killer
" << endl;
            string roomName = readInput("Where dit the murder took place? ");
            Room *room = this->searchRoom(roomName);
            if (room) {
                if (room == this->murderRoom) {
                    string itemName = readInput("What weapon did the murderer used? ");
                    Item *weapon = searchItem(itemName);
                    if (weapon) {
                        if (weapon == this->murderWeapon) {
                            this->foundKiller = true;
                            this->gameOver = true;
                        } else this->gameOver = true;
                    } else {
                        cout << "item named '" << itemName << "' not found
" << endl;
                        this->command();
                    }
                } else this->gameOver = true;
            } else {
                cout << "room named '" << roomName << "' not found
" << endl;
                this->command();
            }
        } else this->gameOver = true;
    } else {
        cout << "suspect named '" << suspectName << "' not found
" << endl;
        this->command();
    }
}

// Return a list of item pointer of items in inventory
vector<Item *> Game::getInventory() {
    vector<Item *> inventoryItems;

    for (int i = 0; i < this->items.size(); i++) {
```

```
        if (items[i].getRoom() == &this->inventory) {
            inventoryItems.push_back(&items[i]);
        }
    }

    return inventoryItems;
}
```

game.h

```
/*************************************************************
 *
 * game.h
 *
 * Holds the game state
 * Controls the flow of the game
 *
*************************************************************/

#pragma once
#include "player.h"
#include "suspect.h"
#include "room.h"
#include "item.h"
#include <string>
#include <vector>

enum View {
    VIEW_TOWER,
    VIEW_ROOM,
    VIEW_INVENTORY
};

enum Difficulty {
    DIFF_EASY,      // Pickup all items as fast as possible. No time limit
    DIFF_MEDIUM,    //
    DIFF_HARD,      //
    DIFF_NIGHTMARE  // 1 minute time limit, pickup weapon, accuse killer in killer room
};

class Game {
    public:
        Game(std::string playerName, Difficulty difficulty);
        void showStoryLine();
        void showHelpScreen();
        void displayView();
        void command();
        bool getFoundKiller();
        bool getGameOver();

    private:
        // Game objects
        Player player;
```

```
        std::vector<Room> rooms;
        std::vector<Item> items;
        std::vector<Suspect> suspects;

        Room inventory;
        View view;
        Difficulty difficulty;
        bool gameOver;
        bool foundKiller;
        Suspect *killer;
        Room *murderRoom;
        Item *murderWeapon;

        // Create game objects
        void createRooms();
        void createItems();
        void createSuspects();

        // Game view methods
        void cycleView();
        void displayTower();
        void displayRoom();
        void displayInventory();

        // Utility commands
        void confirmQuit();
        void invalidCommand();

        void talk(std::string suspectName);
        void gather();
        Room *getRandomRoom();
        Room *searchRoom(std::string roomName);
        Suspect *searchSuspect(std::string suspectName);
        Item *searchItem(std::string itemName);
        std::vector<Item *> getInventory();
        void pickup(std::string itemName);
        void drop(std::string itemName);
        void inspect(std::string itemName);
        void accuse(std::string suspectName);
};
```

item.cpp

```
/*************************************************************
 *
 * item.cpp
 *
*************************************************************/

#include <iostream>
#include "item.h"
#include "utils.h"
#define WINDOW_WIDTH 77  // Not including border
```

```
using namespace std;

Item::Item(string name, string imagePath) {
    this->name = name;
    this->image = readFile(imagePath);
}

string Item::getName() {
    return this->name;
}

Room *Item::getRoom() {
    return this->room;
}

void Item::setRoom(Room *room) {
    this->room = room;
}

string Item::getImage() {
    return this->image;
}

// Add blood stains to item image
void Item::addBlood() {
    string blood = "BLOOD";

    // Replace random column of every other line with "BLOOD"
    for (int i = 1; i <= 32; i++) {
        if (i % 2 == 0) {
            int posible = WINDOW_WIDTH - blood.length() + 1;
            int position = i * 80 + 1 + rand() % posible;
            this->image.replace(position, blood.length(), blood);
        }
    }
}
```

item.h

```
/*************************************************************
 *
 * item.h
 *
 * Represents an item object
 *
*************************************************************/

#pragma once
#include <string>
#include "room.h"

class Item {
```

```
    public:
        Item(std::string name, std::string imagePath);
        std::string getName();
        std::string getImage();
        Room *getRoom();
        void setRoom(Room *room);
        void addBlood();

    private:
        std::string name;
        std::string image;
        Room *room;
};
```

main.cpp

```
/*************************************************************
 *
 * main.cpp
 *
 * Main application file
 * Controls the flow of the application
 * Menu option
 *
 * Briano Goestiawan, 31482228
 *
*************************************************************/

#include <iostream>
#include <ctime>
#include <string>
#include "main.h"
#include "game.h"
#include "utils.h"
#define WINDOW_WIDTH 77   // Not including border
#define WINDOW_HEIGHT 39  // Not including border

using namespace std;

bool hasExit = false;
Difficulty difficulty = DIFF_MEDIUM;

// Function call graph: main -> mainMenu -> startGame
int main() {
    // Seed random with current time
    srand(time(NULL));

    // Run main menu until the user exits
    while (!hasExit) {
        mainMenu();
    }
}
```

```
// Show list of actions to user, run specific actions based on what the user input
void mainMenu() {
    clearScreen();

    // Display main menu screen
    cout << readFile("assets/cover_screen.txt");
    string diff = fixedWidth(difficultyString(), ' ', 14);
    cout << "|                                    |                                                |" << endl;
    cout << "|  1. Start game                     |  3. Show leaderboard                           |" << endl;
    cout << "|                                    |                                                |" << endl;
    cout << "|  2. Change difficulty " << diff << " |  4. Exit                                       |" << endl;
    cout << "|                                    |                                                |" << endl;
    cout << "+------------------------------------------------------------------------------------+" << endl;

    // Get option from user. keep asking until get valid option
    int option;
    do {
        option = readInputInt("Pick one option (1-4): ");
    } while(option < 1 || option > 4);

    // Call the appropriate functions based on option the user selects
    switch (option) {
        case 1:
            runGame();
            break;
        case 2:
            changeDifficulty();
            break;
        case 3:
            showLeaderboard();
            break;
        case 4:
            hasExit = true;
    }
}

// Start game
void runGame() {
    // Get player name from user. Ask again if user put blank
    string playerName;
    do {
        playerName = readInput("Enter player name: ");
    } while (playerName == "");

    // Run the game while keeping track of the time
    int gameStartTimeSeconds = time(NULL);
    Game game(playerName, difficulty);

    // Show story line and help screen before running the main game loop
    game.showStoryLine();
    game.showHelpScreen();

    // Main game loop
    while (!game.getGameOver()) {
```

```
        game.displayView();
        game.command();
    }

    // Display end screen congratulating or ridiculing the player
    // depending on if they win or lose. Show time played
    clearScreen();
    if (game.getFoundKiller()) {
        cout << readFile("assets/you_win.txt");
    } else {
        cout << readFile("assets/game_over.txt");
    }

    int playedTimeSeconds = time(NULL) - gameStartTimeSeconds;
    cout << "|                                                                                |" << endl;
    cout << '|' + fixedWidth("  TIME: " + toHourMinuteSeconds(playedTimeSeconds), ' ', WINDOW_WIDTH) <<
    cout << "|                                                                                |" << endl;
    cout << "+--------------------------------------------------------------------------------+" << endl;
    pause();

    // Add time to leaderboard if win
    if (game.getFoundKiller()) addToLeaderboard(playedTimeSeconds, playerName);
}

// Show the difficulty options to the player than prompt the player to pick one difficulty level
void changeDifficulty() {
    // Display difficulty options
    clearScreen();
    cout << readFile("assets/difficulty_options.txt");

    // Get a valid option from user. Keep asking until user input valid option
    int option;
    do {
        option = readInputInt("Pick an option (1-4):");
    } while (option < 1 || option > 4);

    // Set difficulty based on user input
    difficulty = static_cast<Difficulty>(option - 1);
}

// Display leaderboard. Wait for user before continuing
void showLeaderboard() {
    clearScreen();
    cout << "+--------------------------------------------------------------------------------+" << endl;

    vector<string> leaderboard = stringSplit(readFile(leaderboardFileName()), '
');
    leaderboardFileName();

    cout << "|                                                                                |" << endl;
    cout << '|' << fixedWidth("  LEADERBOARD " + difficultyString(), ' ', WINDOW_WIDTH) << '|' << endl;

    for (int i = 0; i < leaderboard.size(); i++) {
        int spaceIndex = leaderboard[i].find(' ');
```

```
        string time = toHourMinuteSeconds(stoi(leaderboard[i].substr(0, spaceIndex)));
        string playerName = leaderboard[i].substr(spaceIndex + 1);

        int dotCount = WINDOW_WIDTH - playerName.length() - time.length() - 6;
        cout << "|                                                                                |" << end
        cout << "|  " << playerName << ' ' << fixedWidth("", '.', dotCount) << ' ' << time << "  |" << e
    }

    int blankLineCount = WINDOW_HEIGHT - leaderboard.size() * 2 - 2;
    for (int i = 0; i < blankLineCount; i++) {
        cout << "|                                                                                |" << end
    }

    cout << "+--------------------------------------------------------------------------------+" << endl;
    pause();
}

// Add a new entry to leaderboard, insert in correct positon sorted in ascending order by time
void addToLeaderboard(int timeSeconds, string playerName) {
    vector<string> leaderboard = stringSplit(readFile(leaderboardFileName()), '
');
    string entry = to_string(timeSeconds) + ' ' + playerName;

    if (leaderboard.size() > 0) {
        // If leaderboard is not empty, insert entry to correct location
        for (int i = 0; i < leaderboard.size(); i++) {
            int spaceIndex = leaderboard[i].find(' ');
            int time = stoi(leaderboard[i].substr(0, spaceIndex));
            if (timeSeconds < time) {
                leaderboard.insert(leaderboard.begin() + i, entry);
                break;
            }
        }
    } else {
        // Else append to leaderboard as first entry
        leaderboard.push_back(entry);
    }

    writeFile(leaderboardFileName(), stringJoin(leaderboard));
}

string difficultyString() {
    switch (difficulty) {
        case DIFF_EASY:
            return "[EASY]";
        case DIFF_MEDIUM:
            return "[MEDIUM]";
        case DIFF_HARD:
            return "[HARD]";
        case DIFF_NIGHTMARE:
            return "[NIGTHMARE]";
    }
}
```

```
string leaderboardFileName() {
    switch (difficulty) {
        case DIFF_EASY:
           return "leaderboard_easy.txt";
        case DIFF_MEDIUM:
           return "leaderboard_medium.txt";
        case DIFF_HARD:
            return "leaderboard_hard.txt";
        case DIFF_NIGHTMARE:
            return "leaderboard_nigthmare.txt";
    }
}
```

main.h

```
#include <string>
#include "game.h"

void mainMenu();
void runGame();
void changeDifficulty();
void showLeaderboard();
void addToLeaderboard(int timeSeconds, std::string playerName);
std::string difficultyString();
std::string leaderboardFileName();
```

player.cpp

```
#include <iostream>
#include <string>
#include "player.h"
#include "room.h"
#include "utils.h"

using namespace std;

Player::Player() {
    this->name = "";
}

Player::Player(string name, Room *startingRoom) {
    this->name = name;

    this->image.push_back("           ___");
    this->image.push_back("          /@ @\");
    this->image.push_back("          \___/");
    this->image.push_back("          __|__");
    this->image.push_back("            |");
    this->image.push_back("           / \");

    this->room = startingRoom;
    this->room = this->room->getNeighbour(DIR_RIGHT);
}
```

```
string Player::getName() {
    return this->name;
}

vector<string> Player::getImage() {
    return this->image;
}

Room *Player::getRoom() {
    return this->room;
}

void Player::move(Direction direction) {
    Room *destination = this->room->getNeighbour(direction);
    if (destination != NULL) {
        this->room = destination;
    }
}
```

player.h

```
/************************************************************
 *
 * player.h
 *
 * Represents the player (detective)
 *
************************************************************/

#pragma once
#include <string>
#include <vector>
#include "item.h"
#include "room.h"

class Player {
    public:
        Player();
        Player(std::string name, Room *startingRoom);
        std::string getName();
        std::vector<std::string> getImage();
        Room * getRoom();
        void move(Direction direction);

    private:
        std::string name;
        std::vector<std::string> image;
        Room *room;
};
```

room.cpp

```
/*************************************************************
 *
 * room.cpp
 *
*************************************************************/

#include "room.h"
#include "utils.h"
#define WINDOW_WIDTH 77  // Not including border

using namespace std;

Room::Room() { }

Room::Room(string name) {
    this->name = name;
}

Room::Room(string name, string imagePath) {
    // Initialize class variables
    this->name = name;
    this->image = readFile(imagePath);
    this->itemHidden = true;
    /* this->isMurderRoom = false; */

    // Initialize neighbours to NULL
    this->neighbour[DIR_LEFT] = NULL;
    this->neighbour[DIR_RIGHT] = NULL;
    this->neighbour[DIR_UP] = NULL;
    this->neighbour[DIR_DOWN] = NULL;
}

string Room::getName() {
    return this->name;
}

string Room::getImage() {
    return this->image;
}

bool Room::getItemHidden() {
    return this->itemHidden;
}

void Room::search() {
    this->itemHidden = false;
}

// Add blood to room image
void Room::addBlood() {
    string blood = "BLOOD";
```

```
    // Replace random column of every other line with "BLOOD"
    for (int i = 1; i <= 32; i++) {
        if (i % 2 == 0) {
            int posible = WINDOW_WIDTH - blood.length() + 1;
            int position = i * 80 + 1 + rand() % posible;
            this->image.replace(position, blood.length(), blood);
        }
    }
}

void Room::setNeighbour(Direction direction, Room *room) {
    this->neighbour[direction] = room;
}

Room *Room::getNeighbour(Direction direction) {
    return this->neighbour[direction];
}
```

room.h

```
/*************************************************************
 *
 * room.h
 *
 * May containt suspects and items
 *
*************************************************************/

#pragma once
#include <string>
#include <vector>
#include <map>

enum Direction {
    DIR_LEFT,
    DIR_RIGHT,
    DIR_UP,
    DIR_DOWN
};

class Room {
    public:
        Room();
        Room(std::string name);
        Room(std::string name, std::string imagePath);

        std::string getName();
        std::string getImage();
        bool getItemHidden();
        void search();
        void addBlood();

        // Neighbour
```

```
        void setNeighbour(Direction direction, Room *room);
        Room *getNeighbour(Direction direction);

    private:
        std::string name;
        std::string image;
        bool itemHidden;
        std::map<Direction, Room*> neighbour;  // Pointer to neighbouring room in all four direction
};
```

suspect.cpp

```
/*************************************************************
 *
 * suspect.cpp
 *
*************************************************************/

#include <iostream>
#include "suspect.h"
#include "utils.h"

using namespace std;

// Suspect constructor
Suspect::Suspect(string name) {
    this->name = name;
    this->type = SUS_NORMAL;
    this->alibi = NULL;
}

// Return the name of the suspect
string Suspect::getName() {
    return this->name;
}

Room *Suspect::getRoom() {
    return this->room;
}

// Set the location of the suspect
void Suspect::setRoom(Room *room) {
    this->room = room;
}

// Mutator method to set the type of the suspect
void Suspect::setType(SuspectType type) {
    this->type = type;
}

// Set the alibi of the suspect
void Suspect::setAlibi(Suspect *alibi) {
    this->alibi = alibi;
```

```
}

// Display messages from the suspect
void Suspect::talk(string playerName) {
    switch (this->type) {
        case SUS_NORMAL:
        case SUS_KILLER:
            cout << "Hi " << playerName << ", ";

            if (this->alibi == NULL) {
                cout << "I was alone";
            } else {
                cout << "I was with " << this->alibi->getName();
            }

            break;
        case SUS_VICTIM:
            cout << "X_X";
            break;
    }
    cout << '
' << endl;
}
```

suspect.h

```
/*************************************************************
 *
 * suspect.h
 *
 * Represents a suspect
 *
*************************************************************/

#pragma once
#include <string>
#include "room.h"

enum SuspectType {
    SUS_NORMAL,
    SUS_KILLER,
    SUS_VICTIM
};

class Suspect {
    public:
        Suspect(std::string name);
        std::string getName();
        void setType(SuspectType type);
        void setAlibi(Suspect *alibi);
        void talk(std::string playerName);
        void setRoom(Room *room);
        Room *getRoom();
```

```
    private:
        std::string name;
        SuspectType type;
        Suspect *alibi;
        Room *room;
};
```

utils.cpp

```
#include <fstream>
#include <iostream>
#include <vector>
#include <sstream>
#include "utils.h"

using namespace std;

string readFile(string fileName) {
    ifstream file;
    file.open(fileName);

    // If file failed to open
    if (!file.is_open()) {
        cout << "ERROR: cannot open file: " << fileName << endl;
        return "";
    }

    // Append each line in file to content
    string content;
    string line;
    getline(file, content);
    while (!file.eof()) {
        getline(file, line);
        content += '
' + line;
    }
    file.close();

    return content;
}

void writeFile(string fileName, string content) {
    ofstream file;
    file.open(fileName);

    // If file failed to open
    if (!file.is_open()) {
        cout << "ERROR: cannot open file" << fileName << endl;
        return;
    }

    file << content;
```

```
    file.close();
}

vector<string> stringSplit(string content, char character) {
    vector<string> lines;

    // Push to lines when content matches character
    int prev = 0;
    for (int i = 0; i < content.length(); i++) {
        if (content[i] == character) {
            lines.push_back(content.substr(prev, i - prev));
            prev = i + 1;
        }
    }

    // Check if last element if delimitered else ignore
    string end = content.substr(prev);
    if (end.length() > 0) {
        lines.push_back(end);
    }

    return lines;
}

string stringJoin(vector<string> lines, char character) {
    string result = "";

    for (int i = 0; i < lines.size(); i++) {
        result += lines[i] + character;
    }

    return result;
}

string readInput(string prompt) {
    cout << prompt;
    string input;
    getline(cin, input);
    return input;
}

int readInputInt(string prompt) {
    string input;
    do {
        input = readInput("Pick one option (1-4): ");
    } while(!isInteger(input));
    return stoi(input);
}

void pause() {
    readInput("Press Enter to continue ");
}

// Cross platform clear command
```

```
#ifdef _WIN32
#define CLEAR "cls"
#else
#define CLEAR "clear"
#endif
void clearScreen() {
    system(CLEAR);
}

bool isInteger(std::string value) {
    // Empty string is not an integer
    if (value.length() <= 0) {
        return false;
    }

    // Set first index to be checked to allow negative integers
    int firstDigitIndex = 0;
    if (value[0] == '-')
        firstDigitIndex = 1;

    // Check if any characters is not a digit
    for (int i = firstDigitIndex; i < value.length(); i++)
        if (value[i] < '0' or value[i] > '9')
            return false;

    // It survived all the previous tests. It must be an integer
    return true;
}

string toLower(string str) {
    for (int i = 0; i < str.length(); i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            str[i] = str[i] + 'a' - 'A';
        }
    }
    return str;
}

string fixedWidth(string text, char symbol, int width) {
    string output = "";

    for (int i = 0; i < width; i++) {
        if (i < text.length()) {
            output += text[i];
        } else {
            output += symbol;
        }
    }

    return output;
}

string toHourMinuteSeconds(int seconds) {
    int hour = seconds / 60 / 60;
```

```
    int min = seconds / 60 % 60;
    int sec = seconds % 60;
    stringstream output;
    output << hour << ':' << min << ':' << sec;
    return output.str();
}
```

utils.h

```
/*************************************************************
 *
 * utils.h
 *
 * A collection of helper functions
 *
*************************************************************/

#pragma once
#include <string>

// Returns the content of a file specified by fileName
std::string readFile(std::string fileName);

// Write content to file named fileName, overwrite if it exists
void writeFile(std::string fileName, std::string content);

// Split string to vector of strings based on character
std::vector<std::string> stringSplit(std::string content, char character = '
');

// Join vector of string into one string used character
std::string stringJoin(std::vector<std::string> lines, char character = '
');

// Print promp to screen then read and return input line
std::string readInput(std::string prompt = "");

// Print prompt to screen then read input as int. Repeat until user input an int
int readInputInt(std::string prompt = "");

// Pause the control flow until the user press enter
void pause();

// Clear the output screen
void clearScreen();

// Return true if value is an integer string else return false
bool isInteger(std::string value);

// Returns a copy of value with all the uppercase characters replaced with its lowercase equivalent
std::string toLower(std::string value);

// Returns a string with a fixed width and align text to the left
```

```
std::string fixedWidth(std::string text, char symbol, int width);

// Returns seconds time in hour:min:sec string format
std::string toHourMinuteSeconds(int seconds);
```