game.cpp

```cpp
#include "game.h"
#include "player.h"
#include "utils.h"
#include <iostream>

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

Game::Game(string playerName) {
    // Create and allocate rooms
    // TODO: Need to add more rooms
    this->rooms.push_back(Room("Lobby", "Nice place to sit", "XXX"));
    this->rooms.push_back(Room("Office", "Get that bread sir", "$$$"));

    // Create player instance
    Room startingRoom = this->rooms[0];
    this->player = Player(playerName, &startingRoom);

    // Initialize variables
    this->gameOver = false;
    this->gameOver = false;
    this->view = VIEW_TOWER;

    // Show help screen when first running the game
    this->showStoryLine();
    this->showHelpScreen();
}

// Render view. Read command. Repeat.
// Return true if the killer is found (player won), and false otherwise
bool Game::runGameLoop() {
    while (!gameOver) {
        this->renderView();
        this->command();
    }
    return this->foundKiller;
}

// Draw image to screen
void Game::renderView() {
    clearScreen();
    switch (this->view) {
        case VIEW_TOWER:
            cout << readFile("assets/tower.txt");
            break;
        case VIEW_ROOM:
            cout << readFile("assets/room.txt");
            break;
        case VIEW_COUNT:
            break;
    }
```

```
}

void Game::command() {
    // Anatomy of a command:
    //
    //     command   argument
    //        V         V
    //     collect    {item}  <-- What the user types

    int index;
    string command = "";
    string argument = "";

    string input = toLower(readInput("//> "));
    if ((index = input.find(' ')) == string::npos) {
        // Single word command
        command = input.substr(0, index);
    } else {
        // Multi word command
        command = input.substr(0, index);
        argument = input.substr(index + 1);
    }

    if (command == "exit") {
        exit(0);
    } else if (command == "help") {
        this->showHelpScreen();
    } else if (command == "view") {
        this->cycleView();
    } else {
        // Invalid command
        this->command();
    }
}

// Cycle through all the different views
void Game::cycleView() {
    if (this->view < VIEW_COUNT - 1) {
        this->view += 1;
    } else {
        this->view = 0;
    }
}

// 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();
}
```

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_COUNT
};

class Game {
    public:
        Game(std::string playerName);

        bool runGameLoop();  // returns true if the player found the killer

    private:
        Player player;
        std::vector<Room> rooms;

        bool gameOver;
        bool foundKiller;
        int view;

        void renderView();
        void cycleView();
        void command();  // Read command from user and do appropriate actions

        void showStoryLine();
        void showHelpScreen();
};
```

item.cpp

```cpp
#include "item.h"
```

item.h

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

#pragma once
#include <string>

class Item {
    private:
        std::string name;
        std::string description;
        std::string image;

    public:
        Item();
};
```

main.cpp

```cpp
/*************************************************************
 *
 * main.cpp
 *
 * Main application file
 *
 * Briano Goestiawan, 31482228
 *
*************************************************************/

#include "game.h"
#include "utils.h"
#include <iostream>
#include <ctime>

using namespace std;

void mainMenu();
void startGame();
void endGame(bool playerWon, int playedTimeSeconds);
void showLeaderboard();

// Function call graph: main -> mainMenu -> startGame
int main() {
```

```
    while (true) {
        mainMenu();
    }
}

// Show list of actions to user, run specific actions based on what the user
// input
void mainMenu() {
    // Display main menu screen
    clearScreen();
    cout << readFile("assets/main_menu.txt");

    // Get option from user. keep asking until get valid option
    string input;
    do {
        input = readInput("Pick one option (1-3): ");
    } while(!isInteger(input) || stoi(input) < 1 || stoi(input) > 3);

    // Call the appropriate functions based on option the user selects
    switch (stoi(input)) {
        case 1:
            startGame();
            break;
        case 2:
            showLeaderboard();
            break;
        case 3:
            exit(0);
    }
}

// Start game
void startGame() {
    string playerName = readInput("Enter player name: ");
    Game game(playerName);

    // Run the game while keeping track of the time
    int gameStartTimeSeconds = time(0);
    bool playerWon = game.runGameLoop();
    int playedTime = time(0) - gameStartTimeSeconds;

    endGame(playerWon, playedTime);
}

// Display end screen congratulating or ridiculing the player
// depending on if they win or lose. Show time played
void endGame(bool playerWon, int playedTimeSeconds) {
    clearScreen();
    if (playerWon) {
        cout << "Congrats you won!";
    } else {
        cout << readFile("assets/game_over.txt");
    }
    cout << "TIME: " << playedTimeSeconds << endl;
```

```
    pause();
}

// Display leaderboard. Wait for user before continuing
void showLeaderboard() {
    clearScreen();
    cout << readFile("assets/help_screen.txt");
    pause();
}
```

player.cpp

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

using namespace std;

Player::Player() {
}

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

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

vector<Item> Player::getInventory() {
    return this->inventory;
}
```

player.h

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

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

class Player {
    public:
        Player();
```

```
        Player(std::string name, Room *startingRoom);

        std::string getName();
        std::vector<Item> getInventory();

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

room.cpp

```
#include "room.h"

using namespace std;

Room::Room(string name, string description, string image) {
    this->name = name;
    this->description = description;
    this->image = image;
}
```

room.h

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

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

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

    private:
        std::string name;  // must be unique
        std::string description;
        std::string image;

        // List of all suspects in the room
```

```
        std::vector<Suspect> suspects;

        // List of all items in the room
        std::vector<Item> items;
};
```

suspect.cpp

```
#include "suspect.h"
```

suspect.h

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

#pragma once
#include <string>

class Suspect {
    private:
        std::string name;
        std::string description;
        std::string image;

    public:
        Suspect();
};
```

utils.cpp

```
#include <fstream>
#include <iostream>

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;
    }

    return content;
}

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

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);

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