game.cpp

```
#define ROOM_ROWS 4
#define ROOM_COLS 3
#include <iostream>
#include "game.h"
#include "player.h"
#include "utils.h"

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

Game::Game(string playerName) {
    // Create and allocate rooms
    this->createRooms();

    // Create player instance
    Room startingRoom = this->rooms[0];
    this->player = Player(playerName, &startingRoom);
    cout << this->player.getRoom()->getName() << endl;

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

    // Show story line and help screen before running the main game loop
    this->showStoryLine();
    this->showHelpScreen();
    this->runGameLoop();
}

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

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

```
            cout << "


YOUR INVENTORY


" << endl;
            break;
    }
}

void Game::command() {
    // Anatomy of a command:
    //     command    argument can be multiple words
    //     \_____/    \_____________________________/

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

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

    if (command == "exit") {
        this->gameOver = true;
    } else if (command == "help") {
        this->showHelpScreen();
    } else if (command == "view") {
        this->cycleView();
    } else if (command == "tower") {
        this->view = VIEW_TOWER;
    } else if (command == "room") {
        this->view = VIEW_ROOM;
    } else if (command == "inventory") {
        this->view = VIEW_INVENTORY;
    } 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);
    } else {
        this->command();  // Invalid command. Repeat
    }
}
```

```
// 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();
}

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

void Game::createRooms() {
    // Load and append rooms
    // TODO: Load rooms from text file
    this->rooms.push_back(Room("CONTROL CENTER", "nice place", "XXX"));
    this->rooms.push_back(Room("OFFICE", "nice place", "XXX"));
    this->rooms.push_back(Room("SPA", "nice place", "XXX"));
    this->rooms.push_back(Room("LABORATORY", "nice place", "XXX"));
    this->rooms.push_back(Room("LIBRARY", "nice place", "XXX"));
    this->rooms.push_back(Room("GIFT SHOP", "nice place", "XXX"));
    this->rooms.push_back(Room("CAFETARIA", "nice place", "XXX"));
    this->rooms.push_back(Room("LOBBY", "nice place", "XXX"));
    this->rooms.push_back(Room("TOILET", "nice place", "XXX"));
    this->rooms.push_back(Room("SERVER ROOM", "nice place", "XXX"));
    this->rooms.push_back(Room("CAR PARK", "nice place", "XXX"));
    this->rooms.push_back(Room("PLUMBING ROOM", "nice place", "XXX"));

    // 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]);
            }
        }
    }
}

void Game::renderTower() {
    vector<string> output;

    // Append roof
    output.push_back("   +-------------------------------------------------------------------------------+   ")
    output.push_back("  /                                                                               \  ")
    output.push_back(" /                            B R U M P   T O W E R                              \ ")
    output.push_back("|                                                                                 |")
    output.push_back("+---------------------------------------------------------------------------------+")

    // Append each room floor by floor
    for (int row = 0; row < ROOM_ROWS; row++) {
        for (int col = 0; col < ROOM_COLS; col++) {
        }
    }
}
```

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"
```

```cpp
#include <string>
#include <vector>

enum View {
    VIEW_TOWER,
    VIEW_ROOM,
    VIEW_INVENTORY,
    VIEW_COUNT
};

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

        bool getFoundKiller();

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

        bool gameOver;
        bool foundKiller;
        int view;

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

        void renderView();
        void renderTower();

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

image.cpp

```cpp
#include <string>
#include <iostream>

using namespace std;

int main() {
    string mystr = "hello, world";
    cout << mystr << endl;
    mystr[4] = 'A';
    cout << mystr[4] << endl;
}
```

image.h

```
#include <vector>
#include <string>

class Image() {
    private:
        std::vector<std::string> grid;
    public:
        Image(std::string stringImage);

        std::string string();  // Return image as string
        std::string string();  // Return image as string
}
```

item.cpp

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

item.h

```
/************************************************************
 *
 * 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

```
/************************************************************
 *
 * 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: ");

    // Run the game while keeping track of the time
    int gameStartTimeSeconds = time(0);
    Game game(playerName);
```

```
    bool playerWon = game.getFoundKiller();
    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 << "s" << endl;
    pause();
}

// Display leaderboard. Wait for user before continuing
// TODO: Show leaderboard instead of help screen
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;
}

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

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

```
}

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);

        // Accessor methods
        std::string getName();
        std::vector<Item> getInventory();
        Room *getRoom();

        void move(Direction direction);

    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;

    // 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;
}

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>
#include "suspect.h"
#include "item.h"

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

class Room {
    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;

        // Pointer to neighbouring room in all four direction
        std::map<Direction, Room*> neighbour;

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

        std::string getName();

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

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();
};
```

test.h

```
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();
};
```

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
// TODO think of a better argument name than value
std::string toLower(std::string value);

// TODO: implement a function takes string as input and prints it in a frame
void printFramed(std::string value);
```