Magic Maze Online

מגיש: תומר דרור

תז: 328363163

שם החלופה: תכנון ותכנות מערכות

מורה: ניר סליקטר

תאריך הגשה 30.5.24

תוכן עניינים

מבוא

הפרויקט שלי הוא משחק רב משתמשים real time בו 1-8 שחקנים ללא יכולת תקשור צריכים לעבוד ביחד כדי לגלות את מפת הקניון, לגנוב מהחנויות ולברוח לפני שהזמן נגמר, במשחק אין תורות ואפשר לפעול מתי שרוצים על כל אחת מארבעת הדמויות, אך רק בדרכי התנועה הייחודיים הניתנים לך מתחרים נגד הזמן. זה משחק מתסכל שדורש עבודת צוות תחת לחץ זמן ועכשיו יצרתי לו גרסה אונליין.

המשחק

משחק "מאג'יק מייז" (Magic Maze) הוא משחק לוח שיתופי, שבו השחקנים משתפים פעולה כדי להוביל ארבע דמויות - קוסם, לוחם, גנב ואלף - דרך קניון קסום למציאת וגניבת הציוד שלהם מבלי להיתפס על ידי השומרים. הנה תיאור כללי של המשחק:

התחלת המשחק: כל שחקן מקבל קלף פעולה אחד או יותר, המאפשר לו לבצע פעולה מסוימת (לנוע צפונה, דרומה, מזרחה, מערבה, להשתמש בגרמי מדרגות וכו'). כל הפעולות מתבצעות בשיתוף פעולה בין השחקנים, אך אין תקשורת מילולית או סימנים פיזיים מותרים בזמן המשחק. המשחק מתחיל בלוח התחלתי אחד, ועל השחקנים להוסיף עוד חלקי לוח כדי לגלות את הקניון המלא.

מטרת המשחק: למצוא את הפריטים הדרושים לכל דמות ולהוציא אותם מהקניון דרך היציאות המיועדות לפני שנגמר הזמן.

הזמן: המשחק מתנהל תחת מגבלת זמן שמוגדרת מראש. במרכז השולחן יש שעון חול, וכשנגמר הזמן יש לבצע פעולת "עצירה" שבה השחקנים יכולים לתקשר ולתכנן את המהלכים הבאים.

מהלך המשחק: כל שחקן, בתורו, מבצע פעולה המותרת לו לפי קלף הפעולה שלו, על ידי הזזת דמות אחת בלוח. על השחקנים לשתף פעולה ולתכנן את המהלכים בצורה מושכלת כדי לאפשר לכל דמות להגיע אל הפריטים שלה ולברוח מהקניון בזמן.

סיום המשחק: המשחק נגמר כאשר כל הדמויות הצליחו לאסוף את הפריטים שלהן ולצאת מהקניון, או אם הזמן נגמר לפני שהצליחו.

מבני נתונים

במשחק המגרש כל פעם משתנה נפתח ומורכב אחרת לכן מבנה הנתונים שלי צריך להיות מותאם לכל אופציה, הדרך הפשוטה ביותר היא פשוט לשמור את כל המשבצות השונות במערך ענק פסלתי את האופציה הזאת בגלל כמות המקום בזיכרון, יש במשחק עד איזה 30 חלקי מגרש שכל אחד מורכב מ 4X4 משבצות והמערך צריך להכיל אופציה להיפתח לארבעה כיוונים מה שאומר שאני צריך מערך באורך 240X240 שזה 57600 משבצות שזה דרישה גבוהה מאוד מהזיכרון במיוחד אם צריך לשנות בזריזות ולקבל תגובה במהרה. לכן עברתי למבנה נתונים אחר המבנה הכללי הוא מערך דו כיווני דו מימדי (לא מלא) בשביל המשבצות. משבצת היא ריבוע, מכילה מצביע לכל אחד מהמשבצות שאפשר לעבור אליהן מהמשבצת הזו, כמו כן המשבצת מכילה מידע על מה אפשר לעשות במשבצת/ מה קורה כשדורכים עליה.

ארבעת הדמויות של כלי המשחק ממוקמות על משבצות ולכל אחד מצביע למשבצת עליה הוא נמצא. כיוון שיש רק ארבעה דמויות וכיוון שצריך לנהל את איזה חלק מגרש צריך להיפתח יש גם אובייקט של מגרש שמכיל את הדמויות, את החלקים הפתוחים ,את החלקים שעוד צריכים להיפתח וכל הפורטלים שיש לכל אחד מהדמויות. האובייקט הזה במבנה של סינגלטון.

ישנה בעייה גדולה הנובעת מהמבנה נתונים, דמיין ויש לי שמונה חלקי מגרש(ה 4X4) במעגל ונפתח חלק מגרש נוסף במרכז אני צריך לדעת לחבר אותו גם לשאר חלקי המגרש וצריך למצוא את המשבצות אליהן לחבר לכן אני שומר גם את חלקי המגרש כדי להקל על החיפוש למשבצת הנפתחת. לגבי מציאת החלקים לחיפוש אני חיפשתי מיגוון דרכים לחיפוש אך האופציה הכי טובה שמצאתי בלי לשנות את מבנה הנתונים הייתה בסיבוכיות זמן של O(nlogn) וסיבוכיות מקום של O(n) ולכן

הבנתי שיהיה יותר פשוט לשנות את מבנה הנתונים ולשמור רשימה של החלקי מגרש שנפתחו. כפי מה שאמרתי יש אובייקט של חלק מגרש הוא מכיל את המיקום שלו בשביל החיבור עליו הוסבר קודם, את חלקי המגרש הסמוכים אליו, ורשימת הפתיחות שמתחברות לחלקים אחרים. בשביל הבנייה של החלק מגרש אני משתמש בטבלת גיבוב כדי להתאים בין xy זמני ויחסי לחלק מגרש למשבצת וככה אני יודע שאני לא יוצר עותקים של חלק מגרש. בעייה נוספת היא בעית הסיבוב, את החלקים צריך לחבר בכל אוריינטציה לכן צריך להיות אופציה לסובב את החלק מגרש, אם חלק המגרש כבר קיים לסובב אותו זה בעייה לכן יש מבנה נתונים נוסף שזה קדם חלק מגרש שמאחסן מערך דו מימדי של מספרים המאחסנים מידע על המשבצות כגון מה המשבצת עושה ואיפה נמצאים הקירות.

ראה נספח בשביל תמונה של מודל של המבנה נתונים

תקשורת

התקשורת מורכבת משני חלקים, תקשורת בין השרת לצרכן ובין הצרכן לממשק משתמש.

התקשורת ב-Winsock לשרת רב משתמשים היא חיונית לניהול יעיל של חיבורים מרובים בו-זמנית. Winsock, ממשק תכנות יישומים (API) עבור תקשורת ברשתות במערכות הפעלה של Windows, מאפשר לשרת לקבל ולנהל חיבורים מרובים מלקוחות שונים באמצעות מנגנונים כמו sockets. שרת רב משתמשים משתמש ב-Winsock כדי ליצור socket ולהאזין לחיבורים נכנסים. כאשר מתקבל חיבור חדש, השרת יכול להקצות thread נפרד או להשתמש במנגנון אסינכרוני לטיפול בכל חיבור, מה שמאפשר ניהול חיבורים מרובים ביעילות גבוהה יותר.

IPC (תקשורת בין תהליכים) באמצעות זיכרון משותף (shared memory) בין הלקוח לבין הפרונטאנד היא טכניקה נפוצה להעברת נתונים בצורה מהירה ויעילה בתוך מערכת הפעלה אחת. באמצעות shared memory, תהליכים שונים יכולים לגשת לאותו אזור זיכרון ולקרוא או לכתוב נתונים בו ללא צורך במנגנוני תקשורת מסובכים כמו sockets או pipes. זה משפר את ביצועי התקשורת בין הלקוח לפרונטאנד, מכיוון שהעברת הנתונים נעשית ישירות בזיכרון ללא צורך במעברי הקשר נוספים, ומפחיתה את זמני ההשהיה ואת העומס על המעבד.

ראה נספח בשביל תמונה של מודל התקשורת

ממשק משתמש

הממשק משתמש מאפשר גישה לשחקן אין לי הרבה מה להרחיב מעבר להגיד שעשיתי אותו בpygame כי פחות אכפת לי מהמהירות של זה להתעדכן כי אם זה שוגה זה רק במסך שלך ולא משפיע על הרצת המשחק.

ראה נספח בשביל תמונה של חלק מגרש וגם את זרימת האפליקציה .

פרוטוקולים:

פקודות קיימות:

- Login
- Register
- Start
- Move
- getCharacter
- Get
- Open
- CommunicationError

פקודות תוך משחקיות:

פקודות המועברות דרך השרת לכל שאר המשתתפים, פקודות אלו מייצגות פעולה במשחק שעל הדמיות לעשות.

לכל הפקודות יש את המבנה הבא:

Players Movement	variables(might be	command

Ability	(multiple	

Command : אחת מהפקודות הבאות

- Move
- getCharacter
- Get
- Open

Variables : משתנה מפקודה לפקודה ראה למטה לכל אחד

Players Movement Ability : היכולת תנועה של השחקן מבצע הפעולה, מפורק ככה

canOpenFieldPiece	canUseEscalators	canUsePortals	canMoveLeft	canMoveRight	canMoveDown	canMoveUp

Move - בקשה להזיז דמות למיקום ספציפי

משמעות	variables	Move
צבע הדמות המוזזת	color	

ה id של הtile	pos	

דוגמא

move&green$12$1$0$1$0$1$0$1

מבקש להזיז את הדמות הירוקה לtile שהid שלו הוא 12

getCharacter- בקשה לקבלת הid של דמות. בקשה זו נשלחת רק מהFE לBE

משמעות	variables	getCharacter
צבע הדמות	color	

דוגמא

getCharacter&green$1$0$1$0$1$0$1

מבקש לקבל את מיקום הדמות הירוקה

get- בקשה לקבלת צבע של דמות במיקום מסוים. בקשה זו נשלחת רק מהFE לBE

משמעות	variables	getCharacter
Id של ה tile	פסד	

דוגמא

get$12$1$0$1$0$1$0$1

מבקש לקבל את mcg הדמות במיקום 12

Open - בקשה לפתיחת חלק מגרש חדש

משמעות	variables	open
צבע הדמות שפותחת את החלק	color	

דוגמא

open&green$1$0$1$0$1$0$1

מענה על דרישות

תכנות מונחה עצמים:

קיים בקוד המחלקות הבאות: main, field, fieldPiece, Character, Player, movementAbility,Tile,Constants,Utils, PreFieldPiece, Server.

תקשורת

יש שרת מרבה לקוחות המחבר בין כל המשתמשים

יש תקשורת memory share IPC לתקשר בין הbackend וfrontend (מוסבר בטכנולוגיות)

מערכת הפעלה

יש שימוש בthread בשביל שתוכל לקבל מהשרת פקודות תוך כדי שיש תקשורת עם frontEnd

והרצת המשחק עצמו ושימוש בthreads בשרת בשביל השרת המרובה לקוחות

אבטחה

כדי להבטיח את אבטחת המידע המועבר על ידי שילוב של AES ו-RSA בהבטחת נתונים.

AES (Advanced Encryption Standard) AES הוא אלגוריתם הצפנה סימטרי, כלומר אותו מפתח משמש גם להצפנה וגם לפענוח של הנתונים. AES הוא מהיר ויעיל לשימוש עם כמויות גדולות של

נתונים. הוא נחשב לבטוח מאוד כאשר משתמשים בו עם מפתחות באורך מתאים (128, 192, או 256 סיביות).

RSA (Rivest-Shamir-Adleman) RSA הוא אלגוריתם הצפנה אסימטרי, כלומר הוא משתמש בזוג מפתחות: מפתח ציבורי להצפנה ומפתח פרטי לפענוח. RSA מבוסס על הבעייתיות של פירוק מספרים גדולים לגורמים ראשוניים, והוא איטי יחסית לאלגוריתמים סימטריים כמו AES, במיוחד כשמדובר בכמויות גדולות של נתונים. השילוב של AES ו-RSA

ממשק משתמש

במשחק יש ממשק משתמש בpygame כדי לתת למשתמש דרך להפעיל את המשחק.

טכנולוגיות

Front End:

Pygame:

Pygame היא ספרייה פופולרית ב-Python לפיתוח משחקי מחשב ויישומים אינטראקטיביים. היא מספקת כלים קלים לשימוש לעבודה עם גרפיקה, צלילים ואירועים. Pygame נבנית על ספריית SDL (Simple DirectMedia Layer) ומאפשרת יצירה של יישומים גרפיים עם ממשק פשוט ונוח.

IPC memory share:

תקשורת בין תהליכים (IPC) הוא מנגנון המאפשר לתוכנות שונות לתקשר זה עם זה ולסנכרן בתוך אותו מכשיר.memory share היא אחת השיטות המשמשות עבור IPC. השיטה הזו מקצה את אותו קטע זיכרון מוגדר מראש לכמה תוכנות.

Backend

C++:

C++ היא שפת תכנות כללית בעלת יכולות מתקדמות המאפשרות תכנות מונחה עצמים, תכנות פרוצדורלי ותכנות גנרי. היא התפתחה מתוך שפת C ונשמרת התאימות שלה עם C במידה רבה.

C++ נמצאת בשימוש נרחב לפיתוח מערכות תוכנה מורכבות, כולל מערכות הפעלה, דפדפני אינטרנט, משחקים, תוכנות למכשירים משובצים ועוד.

הסיבה שאני משתמש בC++ היא כהשפה מהירה ביותר וחשוב לי במשחק הזמן תגובה ויפעל חלק בהשוואה ל python שימוש בC++ מהיר פי 10-100 בשימוש בכלי בסיס של השפה.

Cpp winsock:

Winsock (Windows Sockets) הוא ממשק תכנות יישומים (API) המשמש לתקשורת רשת במערכות הפעלה Windows. זהו סט של פונקציות וספריות המאפשרות לתוכנות לתקשר דרך פרוטוקולים כמו TCP/IP ו-UDP.

Database - SQL:

הוא מנוע מסד נתונים קל משקל, פשוט לשימוש ובעל יכולות רבות. הוא מאוד פופולרי בעיקר בעקבות היכולת שלו להיות מוטמע באפליקציות קטנות ובעקבות זאת הוא משמש רבות פעמים כמסד נתונים עבור אפליקציות ניידות וכלי אינטרנט. SQLite3 נתמך על מגוון רחב של פלטפורמות ומערכות הפעלה, והוא מצוי בקוד המקור של פייתון, בין היתר, כדי לתמוך במסדי נתונים מוטמעים. המאפיינים הבולטים של SQLite3 כוללים קלות בהתקנה ושימוש, תמיכה ב-SQL, אוטומטיות וניידות. בקיצור, SQLite3 הוא פתרון מוצלח עבור פרויקטים קטנים ואפליקציות ניידות שדורשות מסד נתונים קל משקל ונייד.

ביבליוגרפיה:

https://chatgpt.com/

https://www.w3schools.com/cpp/cpp_class_methods.asp

https://www.w3schools.com/cpp/cpp_output.asp

https://www.w3schools.com/cpp/cpp_classes.asp

https://www.w3schools.com/cpp/cpp_oop.asp

https://www.w3schools.com/cpp/cpp_inheritance_multiple.asp

https://www.w3schools.com/cpp/cpp_arrays.asp

https://www.w3schools.com/cpp/cpp_inheritance_multiple.asp

https://www.w3schools.com/cpp/cpp_strings.asp

https://www.w3schools.com/cpp/cpp_structs.asp

https://www.w3schools.com/cpp/cpp_constructors.asp

https://www.w3schools.com/cpp/cpp_booleans.asp

https://www.educba.com/c-plus-plus-thread-sleep/

https://www.codementor.io/@hbendali/c-c-macro-bit-operations-ztrat0et6

https://stackoverflow.com/questions/18559028/undefined-reference-to-imp-wsacleanup

https://www.geeksforgeeks.org/typedef-in-cpp/

https://stackoverflow.com/questions/612328/difference-between-struct-and-typedef-struct-in-c

https://www.scaler.com/topics/cpp-sleep/

https://www.geeksforgeeks.org/sql-using-c-c-and-sqlite/

https://www.geeksforgeeks.org/scope-resolution-operator-in-c/

https://www.javatpoint.com/cpp-date-and-time

https://www.quora.com/How-do-I-use-terminal-commands-in-C++

https://www.geeksforgeeks.org/multithreading-in-cpp/

רפלקציה

הפרויקט שלי עוסק בפיתוח משחק רב-משתתפים בזמן אמת, שבו השחקנים צריכים לשתף פעולה כדי לגלות מפת קניון, לגנוב מחנויות ולברוח לפני תום הזמן. בניית המשחק ותהליכי הפיתוח שלו הסתיימו בהצלחה, אך הם היו מלווים באתגרים מסוימים. תהליך העבודה היה משובח, כאשר נתקלת בקשיים במציאת דרך יעילה לתקשר בין הלקוח לשרת. כדי לפתור את הבעיה, החלטת להשתמש ב-memory share IPC, אשר התברר כאמצעי תקשורת נוח ויעיל. בנוסף, בתהליך זה למדת בעומק תכנות בשפת c++ ופיתוח אפליקציות בעזרת pygame, המהווים כלי חשובים שיעזרו לך גם בעתיד. למרות האתגרים, תהליך הלמידה היה מרתק ומועיל.

קוד*:

*חשוב לי לציין כאן שהקוד הזה מובנה עם תלות בספריות וקבצים במיקומים מסויימים לכן ניסיון הרצה ואפילו קומפילציה יכשל, אני מאוד מצתער.

Field.h

```
#ifndef FIELD_H
#define FIELD_H
class Tile;
class Character;
class FieldPiece;
class PreFieldPiece;


#include "FieldPiece.h"
#include "PreFieldPiece.h"
#include "Character.h"
#include "Tile.h"
#include <map>


#include <queue>
```

```
class Field
{
public:
    static Field *getInstance();
    bool isTileVacated(Tile *tile);
    Character *getGreenCharacter();
    Character *getPurpleCharacter();
    Character *getYellowCharacter();
    Character *getOrangeCharacter();
    FieldPiece *getCenterPiece();
    std::vector<Tile*> greenPortals;
    std::vector<Tile*> purplePortals;
    std::vector<Tile*> yellowPortals;
    std::vector<Tile*> orangePortals;
    std::queue<int> futureFieldPieces;
    PreFieldPiece *allFieldPieces[10] =
{nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr,nullptr};
    Field();
    std::vector<FieldPiece*> openedFieldPieces;
```

```
// private:
    // Field(FieldPiece *\fieldPiece);
    static Field *instance ;
    FieldPiece *centerPiece;
    Character *greenCharacter;
    Character *purpleCharacter;
    Character *yellowCharacter;
    Character *orangeCharacter;
};
// Field* Field::instance = nullptr;



#endif
```

Field.cpp

```
#include "FieldPiece.h"
FieldPiece::FieldPiece(int x, int y, Field *playingField, PreFieldPiece *preFieldPiece)
{
    this->x = x;
    this->y = y;
    // std::cout << "x " << this->x << ", y " << this->y;
    int size = preFieldPiece->getSize();
```

```
int **field = preFieldPiece->getData();
std::unordered_map<int, Tile *> umap;
bool isFirst = true;
for (int y = 0; y < size; y++)
{
    for (int x = 0; x < size; x++)
    {
        // std::cout<<x<<", "<<y <<" "<<field[y][x] <<"\n";
        if (field[y][x] != 0)
        {
            Tile *currTile;



            if (umap.find(10 * y + x) == umap.end())
            {
                // std::cout<<" creating tile ";



                currTile = new Tile(this, field[y][x]);
                umap[10 * y + x] = currTile;
                if (isFirst)
                {
                    isFirst = false;
```

```
            this->tile = currTile;
        }
    }
    else
    {
        currTile = umap[10 * y + x];
    }
    if (Utils::getTileFeature(currTile->tileType) == "opening" ||
Utils::getTileFeature(currTile->tileType) == "entrance")
    {
        this->openings.push_back(currTile);
        // std::cout << "\n openingss " << openings.size();
    }
    // std::cout << "\nright\n";
    this->locateTile(&(currTile->tileToRight), x + 1, y, field, size, &umap,
Utils::tileBlockedMoveRight(currTile->tileType));
    // std::cout << "left\n";
    this->locateTile(&(currTile->tileToLeft), x - 1, y, field, size, &umap,
Utils::tileBlockedMoveLeft(currTile->tileType));
    // std::cout << "down\n";
    this->locateTile(&(currTile->tileBellow), x, y + 1, field, size, &umap,
Utils::tileBlockedMoveDown(currTile->tileType));
    // std::cout << "up\n";
```

```
        this->locateTile(&(currTile->tileAbove), x, y - 1, field, size, &umap,
Utils::tileBlockedMoveUp(currTile->tileType));



        if (Utils::getTileFeature(field[y][x]) == "entrance")
        {
            this->entrance = currTile;
        }
        // std::cout << Utils::getTileFeature(field[y][x])<<" " <<" "<<x<<","<<y<< "\n";



        if (Utils::getEscalatorDirectionBitwise(field[y][x]) != 0)
        {



            int direction = Utils::getEscalatorDirectionBitwise(field[y][x]);
            // std::cout << " escalator : " << direction << " " << field[y][x] ;
            int escalatorX = x;
            int escalatorY = y;
            switch (direction)
            {
            case 1: // clock 1
                escalatorX += 1;
```

```
    escalatorY -= 2;
    break;
case 2: // clock 2
    escalatorX += 2;
    escalatorY -= 1;
    break;
case 4: // clock 4
    escalatorX += 2;
    escalatorY += 1;
    break;
case 5: // clock 5
    escalatorX += 1;
    escalatorY += 2;
    break;
case 7: // clock 7
    escalatorX -= 1;
    escalatorY += 2;
    break;
case 8: // clock 8
    escalatorX -= 2;
    escalatorY += 1;
    break;
case 10: // clock 10
```

```
    escalatorX -= 2;
    escalatorY -= 1;
    break;
case 11: // clock 11
    escalatorX -= 1;
    escalatorY -= 2;
    break;
case 12: // up right 12
    escalatorX += 1;
    escalatorY -= 1;
    break;
case 13: // down right 13
    escalatorX += 1;
    escalatorY += 1;
    break;
case 14: // down left 14
    escalatorX -= 1;
    escalatorY += 1;
    break;
case 15: // downup right 15
    escalatorX -= 1;
    escalatorY -= 1;
    break;
```

```
        default:
            std::cout << "escalator error: ";
            break;
        }



// std::cout<<" tjt";
        Tile *escalatorTile;
        // std::cout<<"esc x,y "<< escalatorX<<","<<escalatorY<<"/n";
        if (umap.find(10 * (escalatorY) + escalatorX) == umap.end())
        {



            // std::cout<<"not ok ok";
            escalatorTile = new Tile(this, field[escalatorY][escalatorX]);
            umap[10 * (escalatorY) + escalatorX] = escalatorTile;
        }
        else
        {
            // std::cout<<"ok ok";
            // std::cout << "\n3\n";
```

```
        // std::cout << " ,con esc";
        escalatorTile = umap[10 * (escalatorY) + escalatorX];
    }
    currTile->escalatorTo = escalatorTile;
}



// std::cout<<"type "<< currTile->tileType<<" type ";
// std::cout<<" yay1 ";
if (Utils::getTileFeature(field[y][x]) == "portal")
{
    // std::cout<<" yay2 ";
    if (Utils::getTileColor(field[y][x]) == "green")
    {
        // std::cout<<" yay3" ;
        playingField->greenPortals.push_back(currTile);
    }
    if (Utils::getTileColor(field[y][x]) == "purple")
    {
        playingField->purplePortals.push_back(currTile);
    }
    if (Utils::getTileColor(field[y][x]) == "orange")
    {
```

```
                playingField->orangePortals.push_back(currTile);
            }
            if (Utils::getTileColor(field[y][x]) == "yellow")
            {
                playingField->yellowPortals.push_back(currTile);
            }
        }
    }
    // std::cout<<"\n";
  }
}
// std::cout<<"x "<<this->x<<", y "<<this->y;
// std::cout << "openings length " << this->openings.size() << "\n";
}
void FieldPiece::locateTile(Tile **returnTile, int x, int y, int **field, int size,
std::unordered_map<int, Tile *> *umap_ptr, bool blockedInDirection)
{
  // if (x >= 0 && y >= 0 && x<=4 &&y<=4)
    // std::cout<<"asfadf"<<x <<" "<<y;
    // std::cout << " blocked in direction " <<!blockedInDirection;
  if (x < size && x >= 0 && y < size && y >= 0 && field[y][x] != 0 && !blockedInDirection)
  {
```

```
        // std::cout << " "< x << ", " << y <<" "<<(*returnTile)->tileType<< "   " << field[y][x] <<
"\n";
        Tile *tileInDirection;
        if ((*umap_ptr).find(10 * (y) + x) == (*umap_ptr).end())
        {
            // std::cout<<" ,cr up";



            tileInDirection = new Tile(this, field[y][x]);
            (*umap_ptr)[10 * (y) + x] = tileInDirection;
        }
        else
        {
            // std::cout<<" ,con up";
            tileInDirection = (*umap_ptr)[10 * (y) + x];
        }
        // std::cout<<x<<", "<< y<< " "<< tileInDirection->tileType <<"\n";
        (*returnTile) = tileInDirection;
    }
    else
    {
        (*returnTile) = nullptr;
    }
```

```
}
```

FieldPiece.h

```
#ifndef FIELDPIECE
#define FIELDPIECE


class Tile;
class PreFieldPiece;
class Field;
#include "Utils.h"
#include <iostream>
#include "PreFieldPiece.h"
#include "Tile.h"
#include <unordered_map>
#include <stdexcept>
#include "Field.h"
#include <vector>


class FieldPiece
{
```

```
public:
    int x;
    int y;
    Tile *tile;
    Tile *entrance;
    std::vector<Tile *> openings;



    // FieldPiece(PreFieldPiece *preFieldPiece);
    FieldPiece(int x, int y,Field *field, PreFieldPiece *preFieldPiece);



    FieldPiece *leftPiece;
    FieldPiece *rightPiece;
    FieldPiece *upPiece;
    FieldPiece *downPiece;



private:
    void locateTile(Tile **returnTile, int x, int y, int **field,int size, std::unordered_map<int, Tile
*> *umap_ptr,bool blockedInDirection);
};
#endif
```

FieldPiece.cpp

```
#include "FieldPiece.h"

FieldPiece::FieldPiece(int x, int y, Field *playingField, PreFieldPiece *preFieldPiece)

{

    this->x = x;

    this->y = y;

    // std::cout << "x " << this->x << ", y " << this->y;

    int size = preFieldPiece->getSize();

    int **field = preFieldPiece->getData();

    std::unordered_map<int, Tile *> umap;

    bool isFirst = true;

    for (int y = 0; y < size; y++)

    {

        for (int x = 0; x < size; x++)

        {

            // std::cout<<x<<", "<<y <<" "<<field[y][x] <<"\n";

            if (field[y][x] != 0)

            {

                Tile *currTile;
```

```
            if (umap.find(10 * y + x) == umap.end())
            {
                // std::cout<<" creating tile ";



                currTile = new Tile(this, field[y][x]);
                umap[10 * y + x] = currTile;
                if (isFirst)
                {
                    isFirst = false;
                    this->tile = currTile;
                }
            }
            else
            {
                currTile = umap[10 * y + x];
            }
            if (Utils::getTileFeature(currTile->tileType) == "opening" ||
Utils::getTileFeature(currTile->tileType) == "entrance")
            {
                this->openings.push_back(currTile);
                // std::cout << "\n openingss " << openings.size();
            }
```

```
        // std::cout << "\nright\n";
        this->locateTile(&(currTile->tileToRight), x + 1, y, field, size, &umap,
Utils::tileBlockedMoveRight(currTile->tileType));
        // std::cout << "left\n";
        this->locateTile(&(currTile->tileToLeft), x - 1, y, field, size, &umap,
Utils::tileBlockedMoveLeft(currTile->tileType));
        // std::cout << "down\n";
        this->locateTile(&(currTile->tileBellow), x, y + 1, field, size, &umap,
Utils::tileBlockedMoveDown(currTile->tileType));
        // std::cout << "up\n";
        this->locateTile(&(currTile->tileAbove), x, y - 1, field, size, &umap,
Utils::tileBlockedMoveUp(currTile->tileType));



        if (Utils::getTileFeature(field[y][x]) == "entrance")
        {
            this->entrance = currTile;
        }
        // std::cout << Utils::getTileFeature(field[y][x])<<" " <<" "<<x<<","<<y<< "\n";



        if (Utils::getEscalatorDirectionBitwise(field[y][x]) != 0)
        {
```

```
int direction = Utils::getEscalatorDirectionBitwise(field[y][x]);
// std::cout << " escalator : " << direction << " " << field[y][x] ;
int escalatorX = x;
int escalatorY = y;
switch (direction)
{
case 1: // clock 1
    escalatorX += 1;
    escalatorY -= 2;
    break;
case 2: // clock 2
    escalatorX += 2;
    escalatorY -= 1;
    break;
case 4: // clock 4
    escalatorX += 2;
    escalatorY += 1;
    break;
case 5: // clock 5
    escalatorX += 1;
    escalatorY += 2;
```

```
    break;
case 7: // clock 7
    escalatorX -= 1;
    escalatorY += 2;
    break;
case 8: // clock 8
    escalatorX -= 2;
    escalatorY += 1;
    break;
case 10: // clock 10
    escalatorX -= 2;
    escalatorY -= 1;
    break;
case 11: // clock 11
    escalatorX -= 1;
    escalatorY -= 2;
    break;
case 12: // up right 12
    escalatorX += 1;
    escalatorY -= 1;
    break;
case 13: // down right 13
    escalatorX += 1;
```

```
            escalatorY += 1;
            break;
        case 14: // down left 14
            escalatorX -= 1;
            escalatorY += 1;
            break;
        case 15: // downup right 15
            escalatorX -= 1;
            escalatorY -= 1;
            break;



        default:
            std::cout << "escalator error: ";
            break;
        }



// std::cout<<" tjt";
        Tile *escalatorTile;
        // std::cout<<"esc x,y "<< escalatorX<<","<<escalatorY<<"/n";
        if (umap.find(10 * (escalatorY) + escalatorX) == umap.end())
        {
```

```
        // std::cout<<"not ok ok";
        escalatorTile = new Tile(this, field[escalatorY][escalatorX]);
        umap[10 * (escalatorY) + escalatorX] = escalatorTile;
    }
    else
    {
        // std::cout<<"ok ok";
        // std::cout << "\n3\n";
        // std::cout << " ,con esc";
        escalatorTile = umap[10 * (escalatorY) + escalatorX];
    }
    currTile->escalatorTo = escalatorTile;
}



// std::cout<<"type "<< currTile->tileType<<" type ";
// std::cout<<" yay1 ";
if (Utils::getTileFeature(field[y][x]) == "portal")
{
    // std::cout<<" yay2 ";
    if (Utils::getTileColor(field[y][x]) == "green")
```

```
                {
                    // std::cout<<" yay3" ;
                    playingField->greenPortals.push_back(currTile);
                }
                if (Utils::getTileColor(field[y][x]) == "purple")
                {
                    playingField->purplePortals.push_back(currTile);
                }
                if (Utils::getTileColor(field[y][x]) == "orange")
                {
                    playingField->orangePortals.push_back(currTile);
                }
                if (Utils::getTileColor(field[y][x]) == "yellow")
                {
                    playingField->yellowPortals.push_back(currTile);
                }
            }
        }
        // std::cout<<"\n";
    }
}
// std::cout<<"x "<<this->x<<", y "<<this->y;
// std::cout << "openings length " << this->openings.size() << "\n";
```

```
}
void FieldPiece::locateTile(Tile **returnTile, int x, int y, int **field, int size,
std::unordered_map<int, Tile *> *umap_ptr, bool blockedInDirection)
{
   // if (x >= 0 && y >= 0 && x<=4 &&y<=4)
      // std::cout<<"asfadf"<<x <<" "<<y;
      // std::cout << " blocked in direction " <<!blockedInDirection;
   if (x < size && x >= 0 && y < size && y >= 0 && field[y][x] != 0 && !blockedInDirection)
   {
      // std::cout << " "< x << ", " << y <<" "<<(*returnTile)->tileType<< "   " << field[y][x] <<
"\n";
      Tile *tileInDirection;
      if ((*umap_ptr).find(10 * (y) + x) == (*umap_ptr).end())
      {
         // std::cout<<" ,cr up";



         tileInDirection = new Tile(this, field[y][x]);
         (*umap_ptr)[10 * (y) + x] = tileInDirection;
      }
      else
      {
         // std::cout<<" ,con up";
```

```
            tileInDirection = (*umap_ptr)[10 * (y) + x];
        }
        // std::cout<<x<<", "<< y<< " "<< tileInDirection->tileType <<"\n";
        (*returnTile) = tileInDirection;
    }
    else
    {
        (*returnTile) = nullptr;
    }
}
```

PreFieldPiece.h

```
#ifndef PREFIELDPIECE_H
#define PREFIELDPIECE_H
#include "Constants.h"
#include "Utils.h"
class PreFieldPiece {
private:
    int** tiles;



    int getTileFeature(int tileValue);
```

```
    int getTileColor(int tileValue);


    bool canTileMoveUp(int tileValue);
    bool canTileMoveDown(int tileValue);
    bool canTileMoveLeft(int tileValue);
    bool canTileMoveRight(int tileValue);



public:
    int size;
    PreFieldPiece(int** tiles, int size);
    // PreFieldPiece();
    void rotateLeft();
    void rotateRight();
    int** getData();
    int getSize();

};



#endif
```

PreFieldPiece.cpp

```
#include "PreFieldPiece.h"


// PreFieldPiece::PreFieldPiece(){
//     size =0;
//     std::cout<<"error! please contact Tomer ";
// }
PreFieldPiece::PreFieldPiece(int **tiles, int size)
{
    // std::cout << "fasdf";
    this->size = size;
    this->tiles = new int *[size];
    // std::cout << "wow";
    // std::cout <<" \n";
    for (int i = 0; i < size; i++)
    {
        this->tiles[i] = new int[size];
        for (int j = 0; j < size; j++)
        {
            // std::cout <<" "<< tiles[i][j];
            this->tiles[i][j] = tiles[i][j];
        }
```

```
        // std::cout <<" \n";
    }
}


void PreFieldPiece::rotateLeft()
{
    int **newTiles = new int *[size];
    for (int i = 0; i < size; i++)
    {
        newTiles[i] = new int[size];
        for (int j = 0; j < size; j++)
        {
            newTiles[i][j] = (tiles[j][size - 1 - i]);
            Utils::rotateDirectionLeft(&newTiles[i][j]);
        }
    }


    // Deallocate the old memory


    // Assign the new array
```

```
    tiles = newTiles;
}



void PreFieldPiece::rotateRight()
{
    rotateLeft();
    rotateLeft();
    rotateLeft();
}



int **PreFieldPiece::getData()
{
    return this->tiles;
}
int PreFieldPiece::getSize()
{
    return this->size;
}
// unsigned int PreFieldPiece::extractBits(unsigned int num, unsigned int start, unsigned int end)
// {
//     unsigned int mask = (1 << (end - start + 1)) - 1;
```

```
//     mask <<= start;
//     return (num & mask) >> start;
// }
```

Tiles.h

```
#ifndef TILE_H
#define TILE_H


#include "Player.h"
#include <vector>
#include <iostream>
#include "Field.h"
#include "FieldPiece.h"
class Tile
{
public:
    bool isStandable;
    Tile *escalatorTo;
    Tile *tileAbove;
```

```
    Tile *tileBellow;
    Tile *tileToRight;
    Tile *tileToLeft;
    int tileType;
    FieldPiece *fieldPieceOn;
    Tile(FieldPiece *fieldPiece, int tileType);



    std::vector<Tile *> getPlausibleTargetTiles(MovementAbility *movementAbility);



    int getTileFeature();
    int getTileColor();
};



#endif
```

Tiles.cpp

```
#include "Tile.h"
```

```
Tile::Tile(FieldPiece *fieldPiece, int tileType){

    this->fieldPieceOn = fieldPiece;
    this->tileType = tileType;
    this->tileAbove = nullptr;
    this->tileToRight = nullptr;
    this->tileToLeft = nullptr;
    this->tileBellow = nullptr;
    this->escalatorTo = nullptr; // Using nullptr instead of null
}


std::vector<Tile *> Tile::getPlausibleTargetTiles(MovementAbility *movementAbility)
{
    Tile *temp;
    std::vector<Tile *> plausibleTiles;
```

```
    if (movementAbility->canMoveDown && this->tileBellow != nullptr &&
Field::getInstance()->isTileVacated(this->tileBellow) && !Utils::tileBlockedMoveDown(
this->tileType))
    {
        plausibleTiles.push_back(this->tileBellow);
        temp = this->tileBellow;



        while (temp->tileBellow != nullptr &&
Field::getInstance()->isTileVacated(temp->tileBellow) &&
!Utils::tileBlockedMoveDown(temp->tileType))
        {
            plausibleTiles.push_back(temp->tileBellow);
            temp = temp->tileBellow;
        }
    }



    if (movementAbility->canMoveUp && this->tileAbove != nullptr &&
Field::getInstance()->isTileVacated(this->tileAbove)&& !Utils::tileBlockedMoveUp(
this->tileType))
    {
```

```
        plausibleTiles.push_back(this->tileAbove);
        temp = this->tileAbove;
        while (temp->tileAbove != nullptr &&
Field::getInstance()->isTileVacated(temp->tileAbove)&&
!Utils::tileBlockedMoveUp(temp->tileType))
        {
            plausibleTiles.push_back(temp->tileAbove);
            temp = temp->tileAbove;
        }
    }
    if (movementAbility->canMoveRight && this->tileToRight != nullptr &&
Field::getInstance()->isTileVacated(this->tileToRight)&& !Utils::tileBlockedMoveRight(
this->tileType))
    {



        plausibleTiles.push_back(this->tileToRight);
        temp = this->tileToRight;
        while (temp->tileToRight != nullptr &&
Field::getInstance()->isTileVacated(temp->tileToRight)&&
!Utils::tileBlockedMoveRight(temp->tileType))
        {
```

```
            plausibleTiles.push_back(temp->tileToRight);
            temp = temp->tileToRight;
        }
    }
    // bool a =(this->tileToLeft != nullptr);
    // std::cout<<"can move left "<<movementAbility->canMoveLeft<<" tile to left is null "<<
a<< " is vacated" << Field::getInstance()->isTileVacated(this->tileToLeft) <<"tile blocked" <<
!Utils::tileBlockedMoveLeft( this->tileType)<<"\n";
    if (movementAbility->canMoveLeft && this->tileToLeft != nullptr &&
Field::getInstance()->isTileVacated(this->tileToLeft)&& !Utils::tileBlockedMoveLeft(
this->tileType))
    {
        // std::cout<< this->tileToLeft->tileType <<"hi\n";
        plausibleTiles.push_back(this->tileToLeft);
        temp = this->tileToLeft;
        while (temp->tileToLeft != nullptr &&
Field::getInstance()->isTileVacated(temp->tileToLeft)&&
!Utils::tileBlockedMoveLeft(temp->tileType))
        {
            plausibleTiles.push_back(temp->tileToLeft);
            temp = temp->tileToLeft;
        }
    }
```

```
    if (movementAbility->canUseEscalator && this->escalatorTo != nullptr &&
Field::getInstance()->isTileVacated(this->escalatorTo)){
        plausibleTiles.push_back(this->escalatorTo);
    }
    return plausibleTiles;
}
```

Character.h

```
    #ifndef CHARACTER_H
    #define CHARACTER_H
    #include "PreFieldPiece.h"
    #include "FieldPiece.h"
    #include "Tile.h"
    #include "Player.h"
    #include "MovementAbility.h"
    #include <string>
    #include <vector>


    class Character{
```

```
    public:



        std::string name;
        FieldPiece *fieldPieceOn;
        Tile *tileOn;
        Character(std::string, FieldPiece *startingPieceField,Tile *startingTile);
        std::vector<Tile*> getPlausibleTargetTiles(MovementAbility *playerMovementAbility);
        void move(Tile* target,MovementAbility *playerMovementAbility);
        void openFieldPiece();
        bool isFieldPieceAlreadyExist(std::string direction, FieldPiece *fieldPiece);
private:
    void connectAdjacentFieldPieces(FieldPiece *newFieldPiece);



};



#endif
```

Character.cpp

```
#include "Character.h"

Character::Character(std::string name, FieldPiece *startingFieldPiece, Tile *startingTile)

{

    this->name = name;

    this->fieldPieceOn = startingFieldPiece;

    this->tileOn = startingTile;

}



std::vector<Tile *> Character::getPlausibleTargetTiles(MovementAbility

*playerMovementAbility)

{

    std::vector<Tile *> plausibleTargetTiles;

    if (this->tileOn != nullptr)

    {

        plausibleTargetTiles = (tileOn)->getPlausibleTargetTiles(playerMovementAbility);

    }



    std::vector<Tile *> portals;

    if (this->name == "green")
```

```
        portals = Field::getInstance()->greenPortals;
    if (this->name == "purple")
        portals = Field::getInstance()->purplePortals;
    if (this->name == "orange")
        portals = Field::getInstance()->orangePortals;
    if (this->name == "yellow")
        portals = Field::getInstance()->yellowPortals;
    for (Tile *tile : portals)
    {
        // if()
        if (tile != nullptr && playerMovementAbility->canUsePortals &&
Field::getInstance()->isTileVacated(tile))
        {
            plausibleTargetTiles.push_back(tile);
        }
        //
    }
    return plausibleTargetTiles;
}



// void Character::move(Tile *tile,MovementAbility *playerMovementAbility){
//     std::vector<Tile*> plausibleTiles = getPlausibleTargetTiles(playerMovementAbility);
```

```
//     if(std::find(plausibleTiles.begin(), plausibleTiles.end(), tile)!= plausibleTiles.end()){
//         this->tileOn = tile;
//     }
// }



void Character::move(Tile *tile, MovementAbility *playerMovementAbility)
{
    // Check if the tile is a plausible target tile
    std::vector<Tile *> plausibleTargetTiles = getPlausibleTargetTiles(playerMovementAbility);
    bool isValidMove = false;
    for (Tile *targetTile : plausibleTargetTiles)
    {
        if (targetTile == tile)
        {
            isValidMove = true;
            break;
        }
    }
    if (isValidMove)
    {
        this->tileOn = tile;
    }
```

```
}
void Character::openFieldPiece()
{ // TODO
  //  if (Utils::getTileFeature(this->tileOn->tileType) == "opening" && this->name ==
Utils::getTileColor(this->tileOn->tileType)  &&
isFieldPieceAlreadyExist(Utils::getDirection(this->tileOn->tileType),
this->tileOn->fieldPieceOn ))//this->tileOn->fieldPieceOn->upPiece) // add check that there isnt
a fp in that direction TODo
  //  {



    if (!Field::getInstance()->futureFieldPieces.empty())
    {
        FieldPiece *newFieldPiece;
        std::string direction = Utils::getDirection(this->tileOn->tileType);
        // std::cout << direction;
        int x = this->tileOn->fieldPieceOn->x;
        int y = this->tileOn->fieldPieceOn->y;
        PreFieldPiece preFieldPiece =
*(Field::getInstance()->allFieldPieces[Field::getInstance()->futureFieldPieces.front()]);
        if (direction == "up")
        {
            newFieldPiece = new FieldPiece(x, y + 1, Field::getInstance(), &preFieldPiece);
```

```
    Field::getInstance()->futureFieldPieces.pop();
    Field::getInstance()->openedFieldPieces.push_back(newFieldPiece);
}
else if (direction == "down")
{
    preFieldPiece.rotateLeft();
    preFieldPiece.rotateLeft();



    newFieldPiece = new FieldPiece(x, y - 1, Field::getInstance(), &preFieldPiece);
    Field::getInstance()->futureFieldPieces.pop();
    Field::getInstance()->openedFieldPieces.push_back(newFieldPiece);
}
else if (direction == "right")
{
    preFieldPiece.rotateRight();



    newFieldPiece = new FieldPiece(x + 1, y, Field::getInstance(), &preFieldPiece);
    Field::getInstance()->futureFieldPieces.pop();
    Field::getInstance()->openedFieldPieces.push_back(newFieldPiece);
}
else if (direction == "left")
```

```
{
    preFieldPiece.rotateLeft();



    newFieldPiece = new FieldPiece(x - 1, y, Field::getInstance(), &preFieldPiece);
    Field::getInstance()->futureFieldPieces.pop();
    Field::getInstance()->openedFieldPieces.push_back(newFieldPiece);
}
else
{



    std::cout << "invalid direction";
    return;
}
connectAdjacentFieldPieces(newFieldPiece);
// connect fp
// connect tiles



// TODO rotate fp
```

```
        // change tile type
    }
    // }
}
bool Character::isFieldPieceAlreadyExist(std::string direction, FieldPiece *fieldPiece)
{
    if (direction == "up")
    {
        return fieldPiece->upPiece != nullptr;
    }
    else if (direction == "down")
    {
        return fieldPiece->downPiece != nullptr;
    }
    else if (direction == "right")
    {
        return fieldPiece->rightPiece != nullptr;
    }
    else if (direction == "left")
    {
        return fieldPiece->leftPiece != nullptr;
    }
```

```
    std::cout << "invalid direction";
    return false;
}



void Character::connectAdjacentFieldPieces(FieldPiece *newFieldPiece)
{
    for (FieldPiece *fieldPiece : Field::getInstance()->openedFieldPieces)
    {
        if (newFieldPiece->x == fieldPiece->x && newFieldPiece->y == fieldPiece->y + 1)
        { // opening above



            for (Tile *openingInNewFieldPiece : newFieldPiece->openings)
            {
                for (Tile *openingInFieldPiece : fieldPiece->openings)
                {
                    // std::cout << " new tile " <<
Utils::getDirection(openingInNewFieldPiece->tileType) << " old tile " <<
Utils::getDirection(openingInFieldPiece->tileType) << "\n";
                    if (Utils::getDirection(openingInNewFieldPiece->tileType) == "down" &&
Utils::getDirection(openingInFieldPiece->tileType) == "up")
```

```
                {
                    // std::cout << "KA KA";
                    newFieldPiece->downPiece = fieldPiece;
                    fieldPiece->upPiece = newFieldPiece;
                    openingInNewFieldPiece->tileBellow = openingInFieldPiece;
                    openingInFieldPiece->tileAbove = openingInNewFieldPiece;
                }
            }
        }
    }
    if (newFieldPiece->x == fieldPiece->x && newFieldPiece->y == fieldPiece->y - 1)
    { // opening bellow
        for (Tile *openingInNewFieldPiece : newFieldPiece->openings)
        {
            for (Tile *openingInFieldPiece : fieldPiece->openings)
            {
                // std::cout << " new tile " <<
Utils::getDirection(openingInNewFieldPiece->tileType) << " old tile " <<
Utils::getDirection(openingInFieldPiece->tileType) << "\n";
                if (Utils::getDirection(openingInNewFieldPiece->tileType) == "up" &&
Utils::getDirection(openingInFieldPiece->tileType) == "down")
                {
                    newFieldPiece->upPiece = fieldPiece;
```

```
                fieldPiece->downPiece = newFieldPiece;
                openingInNewFieldPiece->tileAbove = openingInFieldPiece;
                openingInFieldPiece->tileBellow = openingInNewFieldPiece;
            }
        }
    }
}
if (newFieldPiece->x == fieldPiece->x + 1 && newFieldPiece->y == fieldPiece->y)
{ // opening to right



    for (Tile *openingInNewFieldPiece : newFieldPiece->openings)
    {
        for (Tile *openingInFieldPiece : fieldPiece->openings)
        {
            // std::cout << "right new tile " <<
Utils::getDirection(openingInNewFieldPiece->tileType) << " old tile " <<
Utils::getDirection(openingInFieldPiece->tileType) << "\n";
            if (Utils::getDirection(openingInNewFieldPiece->tileType) == "left" &&
Utils::getDirection(openingInFieldPiece->tileType) == "right")
            {
                // std::cout << "KA KA" << openingInNewFieldPiece->tileType / 1000000 <<
openingInFieldPiece->tileType / 1000000;
```

```
                newFieldPiece->leftPiece = fieldPiece;
                fieldPiece->rightPiece = newFieldPiece;
                openingInNewFieldPiece->tileToLeft = openingInFieldPiece;
                openingInFieldPiece->tileToRight = openingInNewFieldPiece;
            }
        }
    }
}
if (newFieldPiece->x == fieldPiece->x - 1 && newFieldPiece->y == fieldPiece->y)
{
    for (Tile *openingInNewFieldPiece : newFieldPiece->openings)
    {
        for (Tile *openingInFieldPiece : fieldPiece->openings)
        {
            // std::cout << "left new tile " <<
Utils::getDirection(openingInNewFieldPiece->tileType) << " old tile " <<
Utils::getDirection(openingInFieldPiece->tileType) << "\n";
            if (Utils::getDirection(openingInNewFieldPiece->tileType) == "right" &&
Utils::getDirection(openingInFieldPiece->tileType) == "left")
            {
                // std::cout << "KA KA" << openingInNewFieldPiece->tileType / 1000000 <<"
"<< openingInFieldPiece->tileType / 1000000<<" ";
```

```
                // std::cout <<  openingInNewFieldPiece->tileType<<" right "  <<
openingInFieldPiece->tileType <<"\n";
                newFieldPiece->rightPiece = fieldPiece;
                fieldPiece->leftPiece = newFieldPiece;
                openingInNewFieldPiece->tileToRight = openingInFieldPiece;
                openingInFieldPiece->tileToLeft = openingInNewFieldPiece;
            }
        }
    }
  }
 }
}
```

Utils.h

```
#ifndef UTILS_H
#define UTILS_H



#include <string>
#include <queue>
#include "Constants.h"
#include <iostream>
```

```
#include <map>

#include <fstream>

#include <sstream>





class Utils

{

public:

    static void setTileFeature(int *tileValue, int feature);

    static void rotateDirectionLeft(int *tileValue);



    static void setEscalatorDirection(int *tileValue, int directionBitwise);

    static int getEscalatorDirectionBitwise(int tileValue);

    static std::string getEscalatorDirection(int tileValue);



    static std::string getTileFeature(int tileValue);

    static std::string getTileColor(int tileValue);

    static int getDirectionBitwise(int tileValue);

    static std::string getDirection(int tileValue);
```

```
    static void setDirection(int *tileValue, int directionBitwise);

    static bool tileBlockedMoveUp(int tileValue);

    static bool tileBlockedMoveDown(int tileValue);

    static bool tileBlockedMoveLeft(int tileValue);

    static bool tileBlockedMoveRight(int tileValue);

    void static setTileBlockedMoveUp(int *tileValue, bool blockedUp);

    void static setTileBlockedMoveDown(int *tileValue, bool blockedDown);

    void static setTileBlockedMoveLeft(int *tileValue, bool blockedLeft);

    void static setTileBlockedMoveRight(int *tileValue, bool blockedRight);

    int static setBits(int num, int value, int start, int end);

    static int extractBits(int num, int start, int end);

    static std::queue<int> createQueueFromVector(const std::vector<int> &vec);

    static  int ** readDatabaseFromFile(const std::string &filePath,int targetID) ;



    };



#endif /* UTILS_H */
```

Utils.cpp

```
#include "Utils.h"

std::string Utils::getTileFeature(int tileValue)

{

    int feature = extractBits(tileValue % 1000000, Constants::tileTypeFeatureStart,

Constants::tileTypeFeatureEnd);

    switch (feature)

    {

    case 0:

        return "inaccessable";

        break;

    case 1:

        return "empty";

        break;

    case 2:

        return "exit";

        break;

    case 3:

        return "store";

        break;

    case 4:

        return "opening";

        break;
```

```
    case 5:
        return "portal";
        break;
    case 6:
        return "escalator";
        break;
    case 7:
        return "time-flip";
        break;
    case 8:
        return "entrance";
        break;
    default:
        return "function error";
        break;
    }
}
std::string Utils::getTileColor(int tileValue)
{
    int color = extractBits(tileValue % 1000000, Constants::tileTypeColorStart,
Constants::tileTypeColorEnd);
    if (color == 0)
    {
```

```
        return "green";
    }
    if (color == 1)
    {
        return "purple";
    }
    if (color == 2)
    {
        return "orange";
    }
    if (color == 3)
    {
        return "yellow";
    }
    return "no-color";
}



int Utils::getDirectionBitwise(int tileValue)
{
    return extractBits(tileValue % 1000000, Constants::tileTypeDirectionStart,
Constants::tileTypeDirectionEnd);
}
```

```
std::string Utils::getDirection(int tileValue)
{
    int bitwiseDirection = Utils::getDirectionBitwise(tileValue);
    switch (bitwiseDirection)
    {
    case 0000:
        return "none";
        break;
    case 1: // 0001
        return "right";
    case 2: // 0010
        return "left";
    case 4: // 0100
        return "down";
    case 5: // 0101
        return "down-right";
    case 6: // 0110
        return "down-left";
    case 8: // 1000
        return "up";
    case 9: // 1001
        return "up-right";
    case 10: // 1010
```

```
        return "up-left";



    default:
        return "invalid direction";
        break;
    }
}



void Utils::setDirection(int *tileValue, int directionBitwise)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, directionBitwise,
Constants::tileTypeDirectionStart, Constants::tileTypeDirectionEnd);
}
void Utils::setTileFeature(int *tileValue, int feature)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, feature,
Constants::tileTypeFeatureStart, Constants::tileTypeFeatureEnd);
}
```

```
int Utils::getEscalatorDirectionBitwise(int tileValue)
{
    return extractBits(tileValue % 1000000, Constants::tileTypeEscalatorDirectionStart,
Constants::tileTypeEscalatorDirectionEnd);
}
std::string Utils::getEscalatorDirection(int tileValue)
{
    int bitwiseDirection = Utils::getEscalatorDirectionBitwise(tileValue);
    switch (bitwiseDirection)
    {
    case 0000:
        return "none";
        break;
    case 1: // 0001
        return "clock 1";
    case 2: // 0010
        return "clock 2";
    case 4: // 0100
        return "clock 4";
    case 5: // 0101
        return "clock 5";
    case 7: // 0110
```

```
        return "clock 7";
    case 8: // 1000
        return "clock 8";
    case 10: // 1001
        return "clock 10";
    case 11: // 1010
        return "clock 11";
    case 12:
        return "up-right";
    case 13:
        return "down-right";
    case 14:
        return "down-left";
    case 15:
        return "up-left";



    default:
        return "invalid direction";
        break;
    }
}
```

```
void Utils::setEscalatorDirection(int *tileValue, int directionBitwise)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, directionBitwise,
Constants::tileTypeEscalatorDirectionStart, Constants::tileTypeEscalatorDirectionEnd);
}


void Utils::rotateDirectionLeft(int *tileValue)
{
    int bitwise = Utils::getDirectionBitwise(*tileValue);
    int right = bitwise % 2;


    bitwise = Utils::setBits(bitwise, (bitwise / 4) % 2, 1, 1); // set right from down
    bitwise = Utils::setBits(bitwise, (bitwise / 2) % 2, 3, 3); // set down from left
    bitwise = Utils::setBits(bitwise, (bitwise / 8) % 2, 2, 2); // set left from up
    bitwise = Utils::setBits(bitwise, right, 4, 4);             // set up from right
    Utils::setDirection(tileValue, bitwise);


    bitwise = Utils::getEscalatorDirectionBitwise(*tileValue);
```

```
// clock rotation
if (bitwise > 0 && bitwise <= 11)
{
    Utils::setEscalatorDirection(tileValue, (bitwise + 9) % 12);
}
// corner rotation
if (bitwise >= 12)
{
    Utils::setEscalatorDirection(tileValue, (bitwise - 11) % 4 + 12);
}



int canMoveRight = Utils::tileBlockedMoveRight(*tileValue);
// std::cout<<(*tileValue)<<"\n";
// std::cout << " right " << Utils::tileBlockedMoveRight(*tileValue);
Utils::setTileBlockedMoveRight(tileValue, Utils::tileBlockedMoveDown(*tileValue));
// std::cout << "  " << Utils::tileBlockedMoveRight(*tileValue);



// std::cout << " down " << Utils::tileBlockedMoveDown(*tileValue);
Utils::setTileBlockedMoveDown(tileValue, Utils::tileBlockedMoveLeft(*tileValue));
// std::cout << "  " << Utils::tileBlockedMoveDown(*tileValue);
```

```
    // std::cout << " left " << Utils::tileBlockedMoveLeft(*tileValue);
    Utils::setTileBlockedMoveLeft(tileValue, Utils::tileBlockedMoveUp(*tileValue));
    // std::cout << "  " << Utils::tileBlockedMoveLeft(*tileValue);



    // std::cout << " up " << Utils::tileBlockedMoveUp(*tileValue);
    Utils::setTileBlockedMoveUp(tileValue, canMoveRight);
    // std::cout << "  " << Utils::tileBlockedMoveUp(*tileValue)<<"\n";
    // std::cout<<(*tileValue)<<"\n";
    // std::cout<<"\n";
}



void Utils::setTileBlockedMoveUp(int *tileValue, bool blockedUp)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, !blockedUp,
Constants::tileTypeBlockedMoveUp, Constants::tileTypeBlockedMoveUp);
}
bool Utils::tileBlockedMoveUp(int tileValue)
{
```

```
    return !Utils::extractBits(tileValue % 1000000, Constants::tileTypeBlockedMoveUp,
Constants::tileTypeBlockedMoveUp);
}



void Utils::setTileBlockedMoveDown(int *tileValue, bool blockedDown)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, !blockedDown,
Constants::tileTypeBlockedMoveDown, Constants::tileTypeBlockedMoveDown);
}
bool Utils::tileBlockedMoveDown(int tileValue)
{
    return !Utils::extractBits(tileValue % 1000000, Constants::tileTypeBlockedMoveDown,
Constants::tileTypeBlockedMoveDown);
}



void Utils::setTileBlockedMoveLeft(int *tileValue, bool blockedLeft)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, !blockedLeft,
Constants::tileTypeBlockedMoveLeft, Constants::tileTypeBlockedMoveLeft);
```

```
}
bool Utils::tileBlockedMoveLeft(int tileValue)
{
    return !Utils::extractBits(tileValue % 1000000, Constants::tileTypeBlockedMoveLeft,
Constants::tileTypeBlockedMoveLeft);
}



void Utils::setTileBlockedMoveRight(int *tileValue, bool blockedRight)
{
    int temp = *tileValue / 1000000;
    *tileValue = temp * 1000000 + Utils::setBits(*tileValue % 1000000, !blockedRight,
Constants::tileTypeBlockedMoveRight, Constants::tileTypeBlockedMoveRight);
}



bool Utils::tileBlockedMoveRight(int tileValue)
{
    return !Utils::extractBits(tileValue % 1000000, Constants::tileTypeBlockedMoveRight,
Constants::tileTypeBlockedMoveRight);
}
```

```
int Utils::setBits(int num, int value, int start, int end)
{
    // Create a mask to set the specified bits to 1
    int mask = ((1 << (end - start + 1)) - 1) << (start - 1);



    // Clear the bits in num that we're going to set
    num &= ~mask;



    // Shift and mask the value to fit in the specified range
    value = (value << (start - 1)) & mask;



    // Set the bits in num to the masked value
    return num | value;
}



int Utils::extractBits(int num, int start, int end)
{
    // start -=1;
    int mask = ((1 << (end - start + 1)) - 1) << (start - 1);
```

```
    return (num & mask) >> (start - 1);
}


std::queue<int> Utils::createQueueFromVector(const std::vector<int> &vec)
{
    std::queue<int> q;
    for (const auto &element : vec)
    {
        q.push(element);
    }
    return q;
}


int **Utils::readDatabaseFromFile(const std::string &filePath, int targetID)
{
    std::ifstream file(filePath);
    if (!file.is_open())
    {
        std::cerr << "Unable to open file" << std::endl;
        return nullptr;
    }
```

```
std::string line;
while (std::getline(file, line))
{
    std::istringstream headerStream(line);
    int id, width, length;
    headerStream >> id >> width >> length;



    if (id == targetID)
    {
        int **matrix = new int *[length];
        for (int i = 0; i < length; ++i)
        {
            matrix[i] = new int[width];
        }



        for (int i = 0; i < length; ++i)
        {
            if (!std::getline(file, line))
            {
```

```
            std::cerr << "Error reading matrix data" << std::endl;
            return nullptr;
        }
        std::istringstream rowStream(line);
        for (int j = 0; j < width; ++j)
        {
            // std::cout<<matrix[i][j];
            rowStream >> matrix[i][j];
        }
    }
    file.close();
    return matrix;
  }
  else
  {
    // Skip the matrix lines
    for (int i = 0; i < length; ++i)
    {
        std::getline(file, line);
    }
  }
}
```

```
    file.close();
    std::cerr << "ID not found" << std::endl;
    return nullptr;
}
```

Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
```

```
class Constants {
public:
    static const int tileTypeFeatureStart = 5 ;
    static const int tileTypeFeatureEnd = 8;
    static const int tileTypeColorStart = 9;
    static const int tileTypeColorEnd = 10;
    static const int tileTypeDirectionStart  = 11;
```

```
    static const int tileTypeDirectionEnd = 14;
    static const int tileTypeEscalatorDirectionStart =15;
    static const int tileTypeEscalatorDirectionEnd = 18;
    static const int tileTypeBlockedMoveUp = 1;
    static const int tileTypeBlockedMoveDown = 2;
    static const int tileTypeBlockedMoveLeft = 3;
    static const int tileTypeBlockedMoveRight =4;





};



#endif
```

Player.h

```
#ifndef PLAYER_H
#define PLAYER_H



#include "MovementAbility.h" // Include MovementAbility header file
```

```
class Player {
public:
    MovementAbility* movementAbility;



    // Constructor
    Player(bool canMoveUp,
           bool canMoveDown,
           bool canMoveLeft,
           bool canMoveRight,
           bool canUseEscalator,
           bool canUsePortals,
           bool canOpenFieldPiece);
};



#endif
```

Player.cpp

```
#include "Player.h"



Player::Player(bool canMoveUp,
      bool canMoveDown,
      bool canMoveLeft,
      bool canMoveRight,
      bool canUseEscalator,
      bool canUsePortals,
      bool canOpenFieldPiece)
{
   this->movementAbility = new MovementAbility();
   this->movementAbility->canMoveUp = canMoveUp;
   this->movementAbility->canMoveDown = canMoveDown;
   this->movementAbility->canMoveLeft = canMoveLeft;
   this->movementAbility->canMoveRight = canMoveRight;
   this->movementAbility->canUseEscalator = canUseEscalator;
   this->movementAbility->canUsePortals = canUsePortals;
   this->movementAbility->canOpenFieldPiece = canOpenFieldPiece;
```

```
}
```

MovementAbility.h

```
#ifndef MOVEMENT_ABILITY_H
#define MOVEMENT_ABILITY_H
typedef struct
{
    bool canMoveUp;
    bool canMoveDown;
    bool canMoveLeft;
    bool canMoveRight;
    bool canUseEscalator;
    bool canUsePortals;
    bool canOpenFieldPiece;

} MovementAbility;
```

```
#endif
```

Main.cpp

```
#include <iostream>

#include <WinSock2.h>

#include <thread>

#include <algorithm>

#include "Character.h"

#include "PreFieldPiece.h"

#include "FieldPiece.h"

#include "Player.h"

#include "Tile.h"

#include <iostream>

#include <fstream>



#include <queue>



bool stringToBool(const std::string &str);
```

```
void printPossibleMoves(MovementAbility *movementAbility);
void printPossibleMoves(Character *character, MovementAbility *movementAbility);
std::string movementAbilityToCmdStr(MovementAbility *movementAbility);
std::queue<std::string> splitString(const char input[]);
MovementAbility *queueToMovementAbility(std::queue<std::string> *cmd, MovementAbility
*movementAbility);
void handleCmd(std::queue<std::string> *cmd);
bool canOpenFieldPiece(Character *character);
void openFieldPiece(Character *character);


const char *SERVER_IP = "127.0.0.1"; // Change this to the server's IP address
const int SERVER_PORT = 27015;
const int BUFFER_SIZE = 1024;
Player firstPlayer(1, 1, 1, 1, 1, 1, 1);
bool started = false;
bool printToText = false;


std::string EncryptAES(const std::string& plaintext, const std::string& key) ;
std::string DecryptAES(const std::string& ciphertext, const std::string& key) ;
void GenerateRSAKeys(RSA::PublicKey &publicKey, RSA::PrivateKey &privateKey) ;
```

```
std::string EncryptRSA(const std::string &message, const RSA::PublicKey &publicKey) ;
std::string DecryptRSA(const std::string &ciphertext, const RSA::PrivateKey &privateKey) ;


std::string EncryptAES(const std::string& plaintext, const std::string& key) {
    CryptoPP::AES::Encryption aesEncryption((byte*)key.c_str(),
CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Encryption cbcEncryption(aesEncryption,
(byte*)key.c_str());


    std::string ciphertext;
    CryptoPP::StringSource(plaintext, true, new
CryptoPP::StreamTransformationFilter(cbcEncryption, new CryptoPP::StringSink(ciphertext)));
    return ciphertext;
}


// AES decryption function
```

```
std::string DecryptAES(const std::string& ciphertext, const std::string& key) {
    CryptoPP::AES::Decryption aesDecryption((byte*)key.c_str(),
CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Decryption cbcDecryption(aesDecryption,
(byte*)key.c_str());



    std::string decryptedtext;
    CryptoPP::StringSource(ciphertext, true, new
CryptoPP::StreamTransformationFilter(cbcDecryption, new
CryptoPP::StringSink(decryptedtext)));
    return decryptedtext;
}





using namespace CryptoPP;



// Function to generate RSA key pair
void GenerateRSAKeys(RSA::PublicKey &publicKey, RSA::PrivateKey &privateKey) {
    AutoSeededRandomPool rng;
```

```
    privateKey.GenerateRandomWithKeySize(rng, 2048);
    RSA::PrivateKey pk = privateKey;
    publicKey.AssignFrom(pk);
}


// Function to encrypt a message with RSA
std::string EncryptRSA(const std::string &message, const RSA::PublicKey &publicKey) {
    AutoSeededRandomPool rng;
    RSAES_OAEP_SHA_Encryptor encryptor(publicKey);
    std::string ciphertext;
    StringSource(message, true, new PK_EncryptorFilter(rng, encryptor, new
StringSink(ciphertext)));
    return ciphertext;
}


// Function to decrypt a message with RSA
std::string DecryptRSA(const std::string &ciphertext, const RSA::PrivateKey &privateKey) {
    AutoSeededRandomPool rng;
    RSAES_OAEP_SHA_Decryptor decryptor(privateKey);
    std::string decrypted;
```

```
    StringSource(ciphertext, true, new PK_DecryptorFilter(rng, decryptor, new
StringSink(decrypted)));
    return decrypted;
}


std::string switchKeys(){
        RSA::PublicKey clientPublicKey, serverPublicKey;
    RSA::PrivateKey clientPrivateKey, serverPrivateKey;
    GenerateRSAKeys(clientPublicKey, clientPrivateKey);
    GenerateRSAKeys(serverPublicKey, serverPrivateKey);


    // Client generates AES key and IV
    AutoSeededRandomPool rng;
    SecByteBlock aesKey(AES::DEFAULT_KEYLENGTH), iv(AES::BLOCKSIZE);
    rng.GenerateBlock(aesKey, aesKey.size());
    rng.GenerateBlock(iv, iv.size());
```

```
    std::string aesKeyStr((const char*)aesKey.data(), aesKey.size());
    std::string ivStr((const char*)iv.data(), iv.size());


    // Client encrypts AES key with server's RSA public key
    std::string encryptedAESKey = EncryptRSA(aesKeyStr, serverPublicKey);
    std::cout << "Encrypted AES Key (Hex): " << HexEncode(encryptedAESKey) << std::endl;


    // Server decrypts AES key with its RSA private key
    std::string decryptedAESKey = DecryptRSA(encryptedAESKey, serverPrivateKey);
    std::cout << "Decrypted AES Key: " << HexEncode(decryptedAESKey) << std::endl;


    // Check if the decrypted AES key matches the original AES key
    if (aesKeyStr == decryptedAESKey) {
        std::cout << "AES key exchange successful!" << std::endl;
    } else {
        std::cout << "AES key exchange failed!" << std::endl;
    }
}
```

```
void HandleServerMessages(SOCKET serverSocket)
{
    char buffer[BUFFER_SIZE];
    int bytesReceived;



    // printPossibleMoves(firstPlayer.movementAbility);
    while (true)
    {
        bytesReceived = recv(serverSocket, buffer, BUFFER_SIZE, 0);
        if (bytesReceived == SOCKET_ERROR || bytesReceived == 0)
        {
            std::cerr << "Error receiving from server. Connection closed." << std::endl;
            break;
        }



        buffer[bytesReceived] = '\0';
        std::cout << "Received from server: " << buffer << std::endl;
        std::queue<std::string> cmd = splitString(buffer);
```

```
        handleCmd(&cmd);
        // printPossibleMoves(firstPlayer.movementAbility);
    }
}



void ClientMain()
{
    WSADATA wsaData;
    int result = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0)
    {
        std::cerr << "WSAStartup failed: " << result << std::endl;
        return;
    }



    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET)
    {
        std::cerr << "Error creating client socket: " << WSAGetLastError() << std::endl;
        WSACleanup();
        return;
```

```
    }



    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP);
    serverAddr.sin_port = htons(SERVER_PORT);



    result = connect(clientSocket, reinterpret_cast<sockaddr *>(&serverAddr),
sizeof(serverAddr));
    if (result == SOCKET_ERROR)
    {
        std::cerr << "Failed to connect to server: " << WSAGetLastError() << std::endl;
        closesocket(clientSocket);
        WSACleanup();
        return;
    }
    std::cout << "Connected to server.";
    std::sting key = switchKeys();
    std::thread serverThread(HandleServerMessages, clientSocket);
```

```
// // Start a thread to handle server messages
// if (exitCode == 0)
// {
//     std::cout << "Python script executed successfully." << std::endl;
// }
// else
// {
//     std::cerr << "Error: Python script execution failed." << std::endl;
//     return ; // Return a non-zero value to indicate failure
// }
// Main loop to send commands to the server
while (true)
{
    std::string cmdStr;
    std::string color;
    // int number = 0;



    // std::cout << "Enter a command: ";
    // std::getline(std::cin, cmdStr);
    if (printToText)
    {
```

```
std::ifstream
MyReadFile("C://Users//tomer//Documents//school//cpProject//src//extras//toCpp.txt");



if (MyReadFile.is_open())
{                           // Check if file is open
   getline(MyReadFile, cmdStr); // Read a line from the file into cmdStr
   if (cmdStr.empty())
   {
      MyReadFile.close(); // Close the file
   }
   else
   {



      // std::cout << cmdStr;
      MyReadFile.close(); // Close the file



      std::ofstream
file("C://Users//tomer//Documents//school//cpProject//src//extras//toCpp.txt", std::ofstream::out |
std::ofstream::trunc);
```

```
            if (printToText && file.is_open())
            {
                file.close();
                //     // std::cout << "File contents deleted successfully." << std::endl;
            }
            // else
            // {
            //     std::cerr << "Unable to open file!";
            //     return;
            // }
        }
    }
}
else
{
    // std::cout << "Enter a command: ";
    std::getline(std::cin, cmdStr);
}
// std::cout << cmdStr << " " ;
std::replace(cmdStr.begin(), cmdStr.end(), ' ', '$');
std::queue<std::string> cmd = splitString(cmdStr.c_str());
bool sendToOthers = false;
```

```
    if (started && cmd.front() == "getCharacter")
    {
        cmd.pop();
        if (cmd.front() == "green")
        {
            // std::cout<<"greeeeen";
            printPossibleMoves(Field::getInstance()->getGreenCharacter(),
firstPlayer.movementAbility);
        }
        if (cmd.front() == "purple")
        {
            printPossibleMoves(Field::getInstance()->getPurpleCharacter(),
firstPlayer.movementAbility);
        }
        if (cmd.front() == "orange")
        {
            printPossibleMoves(Field::getInstance()->getOrangeCharacter(),
firstPlayer.movementAbility);
        }
        if (cmd.front() == "yellow")
        {
            printPossibleMoves(Field::getInstance()->getYellowCharacter(),
firstPlayer.movementAbility);
```

```
        }
    }
    else if (started && cmd.front() == "get")
    {
        cmd.pop();
        if (std::stoi(cmd.front()) == Field::getInstance()->getGreenCharacter()->tileOn->tileType
/ 1000000)
        {
            std::cout << "green character\n";
            printPossibleMoves(Field::getInstance()->getGreenCharacter(),
firstPlayer.movementAbility);
        }
        else if (std::stoi(cmd.front()) ==
Field::getInstance()->getPurpleCharacter()->tileOn->tileType / 1000000)
        {
            std::cout << "purple character\n";
            printPossibleMoves(Field::getInstance()->getPurpleCharacter(),
firstPlayer.movementAbility);
        }
        else if (std::stoi(cmd.front()) ==
Field::getInstance()->getOrangeCharacter()->tileOn->tileType / 1000000)
        {
            std::cout << "orange character\n";
```

```
            printPossibleMoves(Field::getInstance()->getOrangeCharacter(),
firstPlayer.movementAbility);
        }
        else if (std::stoi(cmd.front()) ==
Field::getInstance()->getYellowCharacter()->tileOn->tileType / 1000000)
        {
            std::cout << "yellow character\n";
            printPossibleMoves(Field::getInstance()->getYellowCharacter(),
firstPlayer.movementAbility);
        }
        else
        {
            std::cout << "no character on squere " << cmd.front();
            printPossibleMoves(Field::getInstance()->getYellowCharacter(),
firstPlayer.movementAbility);
        }
        std::cout << "\n";
    }



    else if (cmd.front() == "quit")
    {
        break;
```

```
}
else if (!started && cmd.front() == "start")
{



    // cmd.pop();
    // std::vector<int> myLinkedList;



    // if (!cmd.empty())
    // {



    //     for (int i = 0; i < stoi(cmd.front()); i++)
    //     {
    //         cmd.pop();
    //         myLinkedList.push_back(std::stoi(cmd.front()));
    //         // cmdStr.append("");
    //     }
    // }



    // Field::getInstance().futureFieldPieces = Utils::createQueueFromVector(myLinkedList);
```

```
            // started = true;
            sendToOthers = true;
        }
        else if (started && (cmd.front() == "move" || cmd.front() == "open"))
        {
            // std::cout << "moving";
            cmdStr.append(movementAbilityToCmdStr(firstPlayer.movementAbility));
            sendToOthers = true;
        }



        if (sendToOthers && send(clientSocket, cmdStr.c_str(), cmdStr.length(), 0) ==
SOCKET_ERROR)
        {
            std::cerr << "Error sending message to server: " << WSAGetLastError() << std::endl;
            break;
        }



        // std::this_thread::sleep_for(std::chrono::milliseconds(2000)); // Sleep for 100 milliseconds
        // Close the socket and join the server thread
    }
    closesocket(clientSocket);
```

```
    serverThread.join();



    WSACleanup();
}



int main()
{
    std::thread clientThread(ClientMain);
    clientThread.join();



    return 0;
}



std::string movementAbilityToCmdStr(MovementAbility *movementAbility)
{
    return "$" + std::to_string(movementAbility->canMoveUp) +
            "$" + std::to_string(movementAbility->canMoveDown) +
            "$" + std::to_string(movementAbility->canMoveLeft) +
            "$" + std::to_string(movementAbility->canMoveRight) +
```

```
        "$" + std::to_string(movementAbility->canUseEscalator) +
        "$" + std::to_string(movementAbility->canUsePortals) +
        "$" + std::to_string(movementAbility->canOpenFieldPiece);
}



std::queue<std::string> splitString(const char input[])
{
    std::queue<std::string> result;
    size_t len = strlen(input);
    size_t start = 0;



    for (size_t i = 0; i <= len; ++i)
    {
        if (input[i] == '$' || input[i] == '\0')
        {
            result.push(std::string(input + start, i - start));
            start = i + 1;
        }
    }
```

```
    return result;
}


void printPossibleMoves(MovementAbility *movementAbility)
{
    printPossibleMoves(Field::getInstance()->getGreenCharacter(), movementAbility);
    printPossibleMoves(Field::getInstance()->getPurpleCharacter(), movementAbility);
    printPossibleMoves(Field::getInstance()->getYellowCharacter(), movementAbility);
    printPossibleMoves(Field::getInstance()->getOrangeCharacter(), movementAbility);
    // printPossibleMoves(getInstance().getGreenCharacter(),movementAbility);
}


void printPossibleMoves(Character *character, MovementAbility *movementAbility)
{
    std::cout << "possible " << (character->name) << " tiles ";
    std::vector<Tile *> possibleTiles = character->getPlausibleTargetTiles(movementAbility);
    std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
    if (printToText && outFile.is_open())
        outFile << character->name << "$";
    // Check if the file opened successfully
```

```
for (std::vector<Tile *>::size_type i = 0; i < possibleTiles.size(); ++i)



{



    if (printToText && outFile.is_open())
    {
        // Write data to the file
        outFile << possibleTiles[i]->tileType / 1000000;
        if (i != possibleTiles.size() - 1)
        {
            outFile << "$";
        }
    }
    // std::cout << possibleTiles[i]->tileType  << " ";
    std::cout << possibleTiles[i]->tileType / 1000000 << " ";
}
if (printToText && outFile.is_open())
    outFile.close();
```

```
    std::cout << "\n";
}


void handleCmd(std::queue<std::string> *cmd)
{


    std::string color;
    int number = 0;
    MovementAbility *movementAbility = new MovementAbility();
    if (cmd->front() == "quit")
    {
        exit(0);
    }
    // std::cout << "hey there " << cmd->front() << "\n";
    if (cmd->front() == "start")
    {
        cmd->pop();
        std::vector<int> myLinkedList;
        //
        // std::cout<<"k";
```

```
    while (!cmd->empty())
    {



        // for (int i = 0; i < stoi(cmd->front()); i++)



        myLinkedList.push_back(std::stoi(cmd->front()));
        // std::cout<<"\n here here "<<cmd->front();
        cmd->pop();
        // cmdStr.append("");
    }
    // std::cout<<"k";
    Field::getInstance()->futureFieldPieces = Utils::createQueueFromVector(myLinkedList);



    started = true;
}
if (started && cmd->front() == "open")
{
    cmd->pop();
    color = cmd->front();
    cmd->pop();
```

```
Character *character;
if (color == "green")
{
    character = Field::getInstance()->getGreenCharacter();
}
if (color == "purple")
{
    character = Field::getInstance()->getPurpleCharacter();
}
if (color == "orange")
{
    character = Field::getInstance()->getOrangeCharacter();
}
if (color == "yellow")
{
    character = Field::getInstance()->getYellowCharacter();
}
// std::cout << "request to open";
if (canOpenFieldPiece(character))
{
    // std::cout << "able to open";
    openFieldPiece(character);
}
```

```
    }
    if (started && cmd->front() == "move")
    {
        cmd->pop();



        color = cmd->front();
        cmd->pop();
        number = std::stoi(cmd->front());
        cmd->pop();
        // std::cout<<color<<" \n";
        bool didMove = false;
        movementAbility = queueToMovementAbility(cmd, movementAbility);
        if (color == "green")
        {
            std::vector<Tile *> possibleTiles =
Field::getInstance()->getGreenCharacter()->getPlausibleTargetTiles(movementAbility);
            for (std::vector<Tile *>::size_type i = 0; i < possibleTiles.size(); ++i)
            {



                if (possibleTiles[i]->tileType / 1000000 == number)
                {
```

```
                // std::cout << " value is is: " << possibleTiles[i]->tileType / 1000000;
                didMove = true;



                Field::getInstance()->getGreenCharacter()->move(possibleTiles[i],
movementAbility);
                std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
                if (outFile.is_open())
                    outFile << "move$" << color << "$" << number;
            }
        }
    }
    else if (color == "purple")
    {
        std::vector<Tile *> possibleTiles =
Field::getInstance()->getPurpleCharacter()->getPlausibleTargetTiles(movementAbility);
        for (std::vector<Tile *>::size_type i = 0; i < possibleTiles.size(); ++i)
        {
            if (possibleTiles[i]->tileType / 1000000 == number)
            {
                didMove = true;
```

```
                Field::getInstance()->getPurpleCharacter()->move(possibleTiles[i],
movementAbility);
                std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
                if (outFile.is_open())
                    outFile << "move$" << color << "$" << number;
            }
        }
    }
    else if (color == "orange")
    {
        std::vector<Tile *> possibleTiles =
Field::getInstance()->getOrangeCharacter()->getPlausibleTargetTiles(movementAbility);
        for (std::vector<Tile *>::size_type i = 0; i < possibleTiles.size(); ++i)
        {
            if (possibleTiles[i]->tileType / 1000000 == number)
            {
                didMove = true;



                Field::getInstance()->getOrangeCharacter()->move(possibleTiles[i],
movementAbility);
```

```
                std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
                if (outFile.is_open())
                    outFile << "move$" << color << "$" << number;
            }
        }
    }
    else if (color == "yellow")
    {
        std::vector<Tile *> possibleTiles =
Field::getInstance()->getYellowCharacter()->getPlausibleTargetTiles(movementAbility);
        for (std::vector<Tile *>::size_type i = 0; i < possibleTiles.size(); ++i)
        {
            if (possibleTiles[i]->tileType / 1000000 == number)
            {
                didMove = true;



                std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
                if (outFile.is_open())
                    outFile << "move$" << color << "$" << number;
```

```
                Field::getInstance()->getYellowCharacter()->move(possibleTiles[i],
movementAbility);
            }
        }
    }



    if (!didMove)
    {
        std::ofstream
outFile("C://Users//tomer//Documents//school//cpProject//src//extras//toPython.txt");
        if (outFile.is_open())
            outFile << "move$nowhere";
        std::cout << "moving nowhere";
    }
  }
}



MovementAbility *queueToMovementAbility(std::queue<std::string> *cmd, MovementAbility
*movementAbility)
{
  movementAbility->canMoveUp = stringToBool(cmd->front());
```

```
    cmd->pop();
    movementAbility->canMoveDown = stringToBool(cmd->front());
    cmd->pop();
    movementAbility->canMoveLeft = stringToBool(cmd->front());
    cmd->pop();
    movementAbility->canMoveRight = stringToBool(cmd->front());
    cmd->pop();
    movementAbility->canUseEscalator = stringToBool(cmd->front());
    cmd->pop();
    movementAbility->canUsePortals = stringToBool(cmd->front());
    cmd->pop();
    movementAbility->canOpenFieldPiece = stringToBool(cmd->front());
    cmd->pop();
    return movementAbility;
}


bool stringToBool(const std::string &str)
{
    // Convert the string to lowercase for case-insensitive comparison
    std::string lowerStr;
    for (char c : str)
    {
```

```
        lowerStr += std::tolower(c);
    }



    // Check if the string represents true
    if (lowerStr == "true" || lowerStr == "yes" || lowerStr == "1")
    {
        return true;
    }
    // Check if the string represents false
    else if (lowerStr == "false" || lowerStr == "no" || lowerStr == "0")
    {
        return false;
    }
    // Invalid string representation
    else
    {
        // Handle error, throw exception, or return a default value
        throw std::invalid_argument("Invalid boolean string representation: " + str);
    }
}
bool canOpenFieldPiece(Character *character)
{
```

```
    // std::cout<<"please";
    // std::cout<<"\n";
    // std::cout<<"is opening "<< (Utils::getTileFeature(character->tileOn->tileType) )<<"  tile
color " << Utils::getTileColor(character->tileOn->tileType) << " character color "<<
character->name;
    bool canOpen = Utils::getTileFeature(character->tileOn->tileType) == "opening" &&
Utils::getTileColor(character->tileOn->tileType) == character->name;
    if (Utils::getDirection(character->tileOn->tileType) == "up")
    {
        canOpen = canOpen && character->tileOn->tileAbove == nullptr;
    }
    if (Utils::getDirection(character->tileOn->tileType) == "down")
    {
        canOpen = canOpen && character->tileOn->tileBellow == nullptr;
    }
    if (Utils::getDirection(character->tileOn->tileType) == "right")
    {
        canOpen = canOpen && character->tileOn->tileToRight == nullptr;
    }
    if (Utils::getDirection(character->tileOn->tileType) == "left")
    {
        // std::cout<< canOpen<<" ";
        canOpen = canOpen && character->tileOn->tileToLeft == nullptr;
```

```
    }
    // std::cout << "can open" << canOpen << "\n";
    return canOpen;
}



void openFieldPiece(Character *character)
{
    character->openFieldPiece();
    // FieldPiece newFieldPiece =
FieldPiece(Field::getInstance().allFieldPieces[Field::getInstance().futureFieldPieces]);
}
```

Server.cpp

```
#include <iostream>
#include <WinSock2.h>
#include <vector>
#include <thread>
#include <algorithm>
#include <cryptopp/rsa.h>
```

```
#include <cryptopp/osrng.h>
#include <cryptopp/aes.h>
#include <cryptopp/modes.h>
#include <cryptopp/filters.h>
#include <cryptopp/files.h>
#include <cryptopp/hex.h>


// Add other necessary headers here


using namespace CryptoPP;


const int DEFAULT_PORT = 27015;
const int MAX_CLIENTS = 10;
const int BUFFER_SIZE = 1024;


std::vector<SOCKET> clients;
SOCKET serverSocket;
bool gameStarted = false;
```

```
AutoSeededRandomPool rng;
RSA::PrivateKey privateKey;
RSA::PublicKey publicKey;
SecByteBlock aesKey(AES::DEFAULT_KEYLENGTH);
SecByteBlock aesIV(AES::BLOCKSIZE);


void GenerateRSAKeys() {
    privateKey.GenerateRandomWithKeySize(rng, 2048);
    RSA::PublicKey pubKey(privateKey);
    publicKey = pubKey;
}


std::string RSAEncrypt(const std::string &plain, const RSA::PublicKey &key) {
    std::string cipher;
    RSAES_OAEP_SHA_Encryptor e(key);
    StringSource(plain, true, new PK_EncryptorFilter(rng, e, new StringSink(cipher)));
    return cipher;
}
```

```
std::string RSADecrypt(const std::string &cipher, const RSA::PrivateKey &key) {
    std::string recovered;
    RSAES_OAEP_SHA_Decryptor d(key);
    StringSource(cipher, true, new PK_DecryptorFilter(rng, d, new StringSink(recovered)));
    return recovered;
}



std::string AESEncrypt(const std::string &plain) {
    std::string cipher;
    try {
        CFB_Mode<AES>::Encryption e;
        e.SetKeyWithIV(aesKey, aesKey.size(), aesIV);
        StringSource(plain, true, new StreamTransformationFilter(e, new StringSink(cipher)));
    }
    catch (const Exception &e) {
        std::cerr << e.what() << std::endl;
        exit(1);
    }
    return cipher;
}
```

```
std::string AESDecrypt(const std::string &cipher) {
    std::string recovered;
    try {
        CFB_Mode<AES>::Decryption d;
        d.SetKeyWithIV(aesKey, aesKey.size(), aesIV);
        StringSource(cipher, true, new StreamTransformationFilter(d, new StringSink(recovered)));
    }
    catch (const Exception &e) {
        std::cerr << e.what() << std::endl;
        exit(1);
    }
    return recovered;
}


void BroadcastMessage(const char* message, int size) {
    for (SOCKET client : clients) {
        send(client, message, size, 0);
    }
}


void ClientHandler(SOCKET clientSocket) {
```

```
    char buffer[BUFFER_SIZE];
    int bytesReceived;



    // Send public key to the client
    std::string pubKeyStr;
    StringSink pubKeySink(pubKeyStr);
    publicKey.Save(pubKeySink);
    send(clientSocket, pubKeyStr.c_str(), pubKeyStr.length(), 0);



    // Receive encrypted AES key and IV from client
    bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0);
    if (bytesReceived == SOCKET_ERROR || bytesReceived == 0) {
        std::cerr << "Error receiving AES key from client. Closing connection." << std::endl;
        closesocket(clientSocket);
        return;
    }
    std::string encryptedAesKey(buffer, bytesReceived);
    aesKey = SecByteBlock((const byte*)RSADecrypt(encryptedAesKey, privateKey).data(),
AES::DEFAULT_KEYLENGTH);
```

```
    bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0);
    if (bytesReceived == SOCKET_ERROR || bytesReceived == 0) {
        std::cerr << "Error receiving AES IV from client. Closing connection." << std::endl;
        closesocket(clientSocket);
        return;
    }
    std::string encryptedAesIV(buffer, bytesReceived);
    aesIV = SecByteBlock((const byte*)RSADecrypt(encryptedAesIV, privateKey).data(),
AES::BLOCKSIZE);



    while (true) {
        bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (bytesReceived == SOCKET_ERROR || bytesReceived == 0) {
            std::cerr << "Error receiving from client. Closing connection." << std::endl;
            closesocket(clientSocket);
            return;
        }



        buffer[bytesReceived] = '\0';
        std::string decryptedMessage = AESDecrypt(buffer);
        std::cout << "Received from client: " << decryptedMessage << std::endl;
```

```
        if (!gameStarted && decryptedMessage == "start") {
            std::cout << "Start command received. Broadcasting start to all clients." << std::endl;
            gameStarted = true;
            BroadcastMessage(AESEncrypt("start$2").c_str(), AESEncrypt("start$2").length());
        } else if (gameStarted) {
            // Broadcast received message to all clients
            BroadcastMessage(buffer, bytesReceived);
        }
    }
}


int main() {
    WSADATA wsaData;
    int result = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (result != 0) {
        std::cerr << "WSAStartup failed: " << result << std::endl;
        return 1;
    }
```

```
serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if (serverSocket == INVALID_SOCKET) {
    std::cerr << "Error creating server socket: " << WSAGetLastError() << std::endl;
    WSACleanup();
    return 1;
}


sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = INADDR_ANY;
serverAddr.sin_port = htons(DEFAULT_PORT);


result = bind(serverSocket, reinterpret_cast<sockaddr*>(&serverAddr), sizeof(serverAddr));
if (result == SOCKET_ERROR) {
    std::cerr << "Bind failed: " << WSAGetLastError() << std::endl;
    closesocket(serverSocket);
    WSACleanup();
    return 1;
}
```

```
result = listen(serverSocket, SOMAXCONN);
if (result == SOCKET_ERROR) {
    std::cerr << "Listen failed: " << WSAGetLastError() << std::endl;
    closesocket(serverSocket);
    WSACleanup();
    return 1;
}


std::cout << "Server started. Waiting for clients..." << std::endl;


// Generate RSA keys
GenerateRSAKeys();


while (true) {
    SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "Accept failed: " << WSAGetLastError() << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
```

```
        }



        if (clients.size() >= MAX_CLIENTS) {
            std::cerr << "Maximum clients reached. Closing new connection." << std::endl;
            closesocket(clientSocket);
        } else {
            clients.push_back(clientSocket);
            std::cout << "Client connected. Total clients: " << clients.size() << std::endl;
            std::thread clientThread(ClientHandler, clientSocket);
            clientThread.detach(); // Detach thread so it can run independently
        }
    }



    closesocket(serverSocket);
    WSACleanup();
    return 0;
}
```

FrontEnd.py

```
import pygame
from multiprocessing import shared_memory
import threading


# Shared memory size (2 buffers of 256 bytes each + 2 flags)
SHM_SIZE = 514
BUFFER_SIZE = 256
started = False
# Create shared memory
shm = shared_memory.SharedMemory(create=True, size=SHM_SIZE,
name='my_shared_memory')
buffer = shm.buf


# Pygame setup
pygame.init()
window_size = (800, 600)
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("Python-C++ IPC")


font = pygame.font.Font(None, 36)
```

```
clock = pygame.time.Clock()



# Load images
background_image = pygame.image.load('../../DB/pictures/background.jpg')
# IMAGE = pygame.image.load('../../DB/pictures/image.jpg')
field_settings = pygame.image.load('../../DB/pictures/fieldSettings.png')
image_x, image_y = 100, 100  # Initial position
image_width, image_height = 100, 100  # Initial size
perimeter_width = 125  # Width of the cyan perimeter



# Button class
class Button:
    def __init__(self, x, y, width, height, text, onclick):
        self.rect = pygame.Rect(x, y, width, height)
        self.color = (0, 128, 255)
        self.text = text
        self.onclick = onclick
        self.font = pygame.font.Font(None, 36)



    def draw(self, screen):
```

```
        pygame.draw.rect(screen, self.color, self.rect)
        text_surface = self.font.render(self.text, True, (255, 255, 255))
        text_rect = text_surface.get_rect(center=self.rect.center)
        screen.blit(text_surface, text_rect)


    def is_clicked(self, pos):
        return self.rect.collidepoint(pos)


# Create a start button
start_button = Button(350, 275, 100, 50, 'Start', lambda: send_to_cpp('start'))


# Function to send a message to the C++ program
def send_to_cpp(message):
    if len(message) > BUFFER_SIZE:
        raise ValueError("Message too long")
    buffer[0] = 1  # Indicate message is ready
    buffer[2:2 + len(message)] = bytearray(message.encode('utf-8'))
    buffer[2 + len(message):258] = bytearray(BUFFER_SIZE - len(message))  # Clear remaining
bytes
```

```
# Function to receive a message from the C++ program
def recv_from_cpp():
    if buffer[1] == 1:
        message = bytes(buffer[258:514]).decode('utf-8').rstrip('\x00')
        buffer[1] = 0  # Clear the flag
        return message
    return None



def display_basic_game_UI():
    screen.blit(background_image, (0, 0))  # Draw background
    pygame.draw.rect(screen, (0, 255, 255), (0, 0, window_size[0], window_size[1]),
perimeter_width)
    display_image(field_settings, 250, 0, 300, 125)
    pygame.display.flip()



def listen_for_cpp_messages():
    do_once = True
    while True:
        message = recv_from_cpp()
        if message:
```

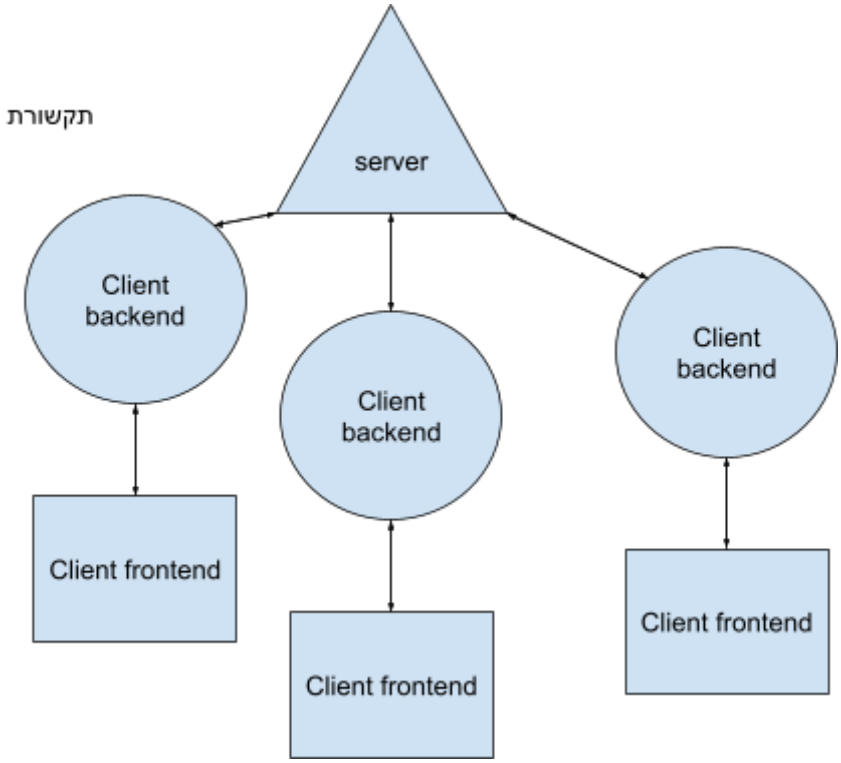
```
            print(f"Received from C++: {message}")
            if message == "start":
                display_basic_game_UI()
            else:
                display_message(message)
        else:
            if do_once:
                do_once = False
                # Display the background image if no message is received
                screen.blit(background_image, (0, 0))  # Draw background
                pygame.display.flip()



# Display message on the Pygame window
def display_message(message):
    screen.blit(background_image, (0, 0))  # Draw background
    text = font.render(message, True, (255, 255, 255))
    screen.blit(text, (20, 20))
    pygame.display.flip()



# Display image with cyan perimeter on the Pygame window
def display_image(image, x, y, width, height):
```

```
    # Do not clear the screen here to preserve the perimeter
    scaled_image = pygame.transform.scale(image, (width, height))
    screen.blit(scaled_image, (x, y))
    pygame.display.flip()


# Start the listener thread
listener_thread = threading.Thread(target=listen_for_cpp_messages)
listener_thread.daemon = True
listener_thread.start()


try:
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                shm.close()
                shm.unlink()
                exit()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if start_button.is_clicked(event.pos):
                    start_button.onclick()
```

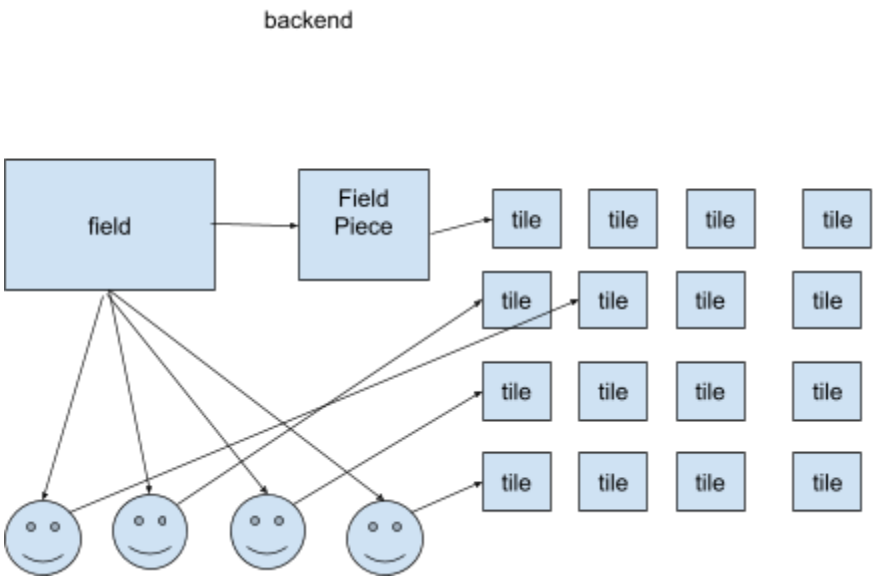
```
                started = True



        # Redraw the screen with the button
        screen.blit(background_image, (0, 0))
        if not started:
            start_button.draw(screen)
        pygame.display.flip()



        clock.tick(30)  # Limit to 30 frames per second



finally:
    shm.close()
    shm.unlink()
```

נספח

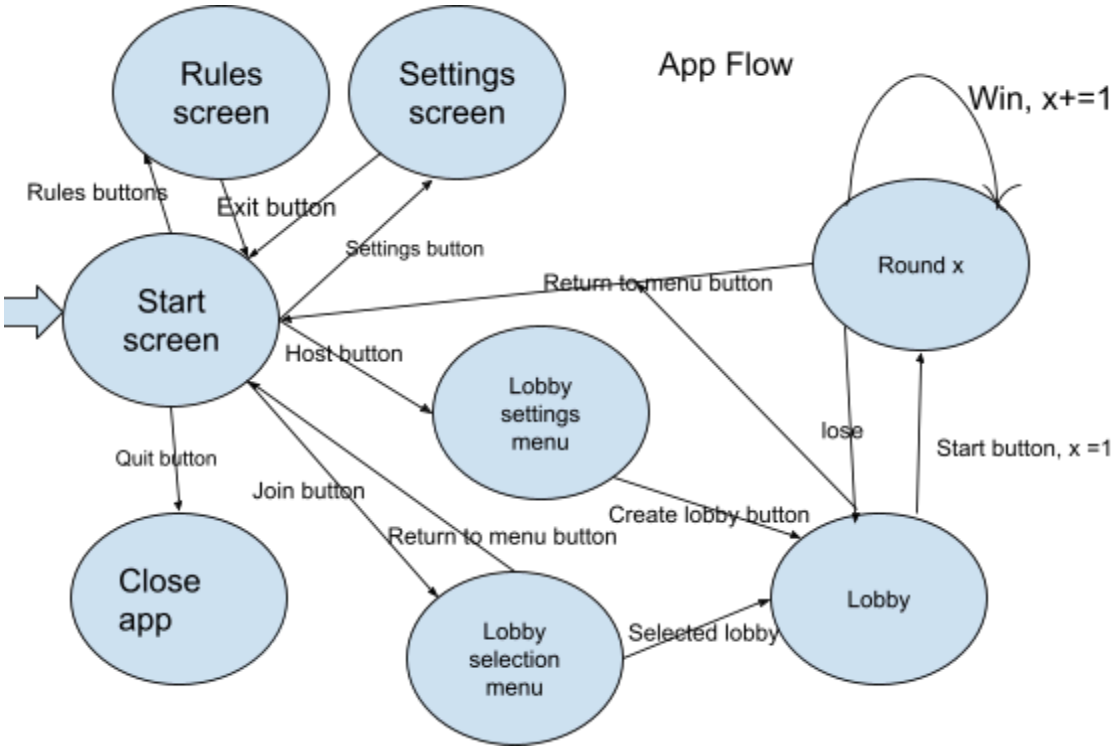
מודל תקשורת:



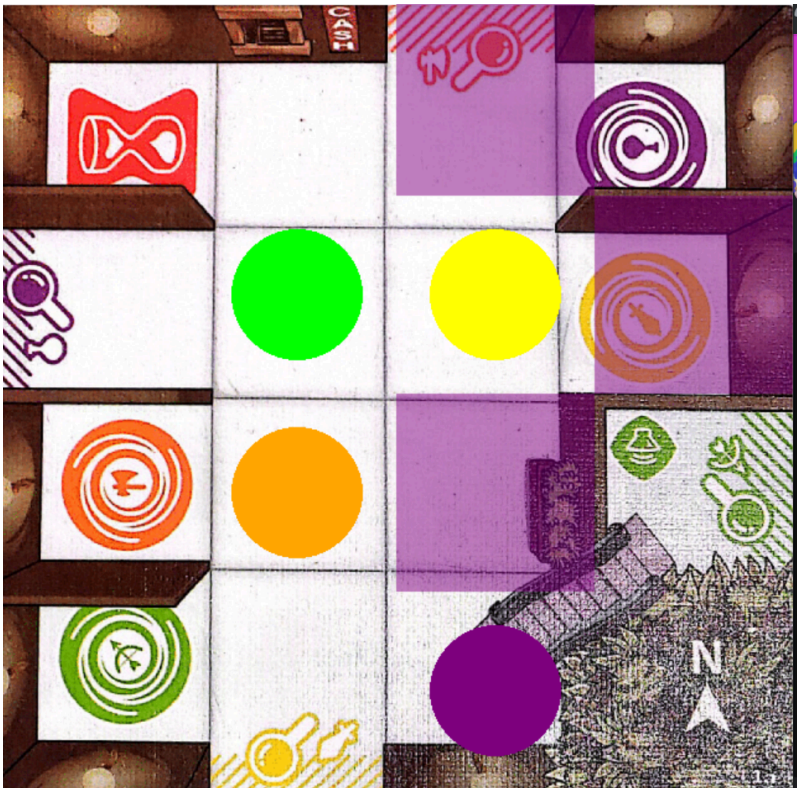
מבנה נתונים



מודל המשחק



רציף משחק


CASH
N