Adventure

2021-06-01

混合1902

余丛杉

3190103165

01. Introduction

> Adventure is a CLI game.

- **Game Rules**
  - The player has to explore in the castle with many levels and a lot of rooms.
  - The task of the player is to find a room where the princess is prinsoned and take her leave the castle.
  - There are many types of rooms, and each type of room has different types of exits.
  - When the game starts, the player is at the lobby of the castle.
  - Note that there's a monster in one of the rooms, which the exact location is not able to be aware. Once the player meets a monster, the game is over.
  - Once the player enters the room of princess, the program shows a message about the princess, and the process is going to leave with the player.
  - The player then has to find their way out the castle.
  - The only way to leave the castle is via the lobby.
- **Output Specification**
  - The program shows information about the lobby's name of the room, how many exits are there, and names of all exits (e.g.: `"east", "south", "up"`), like:

    > Welcome to the lobby. There are 3 exits as: east, west and up.
    >
    > Enter your command:

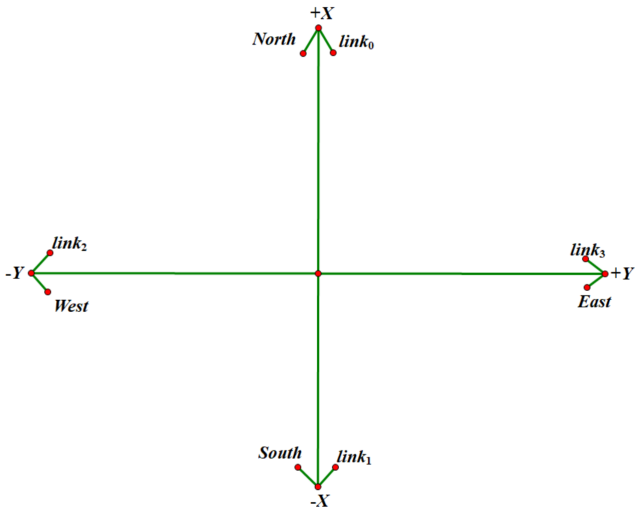
  - The program shows the information about that room, like what happened in the lobby just now. And the player may input command to choose another room.
- **Input Specification**
  - The player then can input `"go"` followed by the name of one exit to enter the room connected with that door, like:

    > go east
- **Requirement**
  - At least three different kinds of room;
  - At least five rooms;
  - The room with monster or princess is randomly set.

02.Algorithm Specification

2.0 Basic framework

- The game can choose three levels of difficulty:
  - `A.Easy` corresponding map 3*3*3
  - `B.Moderate` corresponding map 5*5*5
  - `C.Difficult` corresponding map 7*7*7
- Whether the room is represented by a general array `link[6]`
  - Azimuth map:
    - `up` - `+Z` - `link[4]`
    - `down` - `-Z` - `link[5]`



    - The room has this exit, the `link[i]` is `true`

- `Game.h`

```
#ifndef GAME_H
#define GAME_H

#include <iostream>
#include <cstdlib>
#include <time.h>
#include <string>
using namespace std;

#define NORTH 0
#define SOUTH 1
#define WEST 2
#define EAST 3
#define UP 4
#define DOWN 5

class Map
{
private:
    int mapSize, mapHeight;
    Rooms *rooms;
    Coordinate Lobby;   // Lobby location
    Coordinate Monster; // Monster location
    Coordinate Princess;  // Princess location
public:
    Map(int s = 0, int h = 0);
    void RoomAllocation();  // This function is used to allocate rooms
    int RoomNum(int x, int y, int z) const; // Calculate the room number from
the location information x, y, z
    int RoomNum( const Coordinate& room ) const;    // Overloaded function
    void Link(const Coordinate& terminal);  // Plan two paths from Lobby to
Monster and Princess
    void RandomLink();  // Randomly connect rooms
    friend class Adventure;
    ~Map();
};

class Adventure
```

```
{
private:
    Map *map;
    Coordinate curLoc;
    void Loop();    // Main game loop
    int ExitsNum( );    // Calculate the export quantity
    void CoutExits();   // Export each export direction
    int Direction(const string& s); // The selected direction is obtained by
user input
public:
    Adventure(){}
    void Initialize();  // Game initialization
    void Start();   // Game start
    ~Adventure(){}
};

class Coordinate{
private:
    int x, y, z;
public:
    Coordinate(){}
    Coordinate(int a, int b, int c);
    bool operator==(const Coordinate& that) const;
    bool operator!=(const Coordinate& that) const;
    friend class Map;
    friend class Adventure;
    ~Coordinate(){}
};

class Rooms
{
private:
    Coordinate loc;
    bool link[6];
public:
    Rooms(int x=0, int y=0, int z=0);
    friend class Map;
    friend class Adventure;
    ~Rooms(){}
};

#endif
```

2.1 Coordinate Class

`Coordinate Class` is the coordinate representation:

- Constructor is used for coordinate initialization;
- Overload `==`
- Overload `!=`
- `Coordinate.cpp`

```
#include "Game.h"

// Constructor: coordinate initialization
Coordinate::Coordinate(int a, int b, int c):x(a), y(b), z(c){}

bool Coordinate::operator==(const Coordinate& that) const{
    return ( x==that.x ) && ( y==that.y ) && ( z==that.z );
}

bool Coordinate::operator!=(const Coordinate& that) const{
    return ( x!=that.x ) || ( y!=that.y ) || ( z!=that.z );
}
```

2.2 Rooms Class

`Room Class` is used to record the connectivity of the room in all directions:

- Constructor: Each room is initially disconnected
- `Room.cpp`

```
#include "Game.h"

// Constructor: Each room is initially disconnected
Rooms::Rooms(int x, int y, int z):loc(x,y,z){
    for( int i=0 ; i<6 ; i++ )
        link[i] = false;
}
```

2.4 Map Class

`Map Class` is used to build maps:

- `viod RoomAllocation()`
    - Randomly assign the princess's room
    - Randomly assign the monster's room
    - Confirm Lobby's room
    - Number the room according to the coordinate position of the room

```
// Allocate rooms
void Map::RoomAllocation()
{
    // Random seeding
    srand((unsigned)time(NULL));

    // Create a room based on the size of the map
    rooms = new Rooms[mapSize*mapSize*mapHeight];
    for( int i=0 ; i<mapSize ; i++ )
        for( int j=0 ; j<mapSize ; j++ )
            for( int k=0 ; k<mapHeight ; k++ )
                rooms[RoomNum(i,j,k)].loc = Coordinate(i,j,k);
```

```
    // The lobby is in the center
    Lobby = Coordinate( mapSize/2 , mapSize/2 , mapHeight/2 );

    // Randomly generate Monster position
    do{
        Monster = Coordinate(rand()%(mapSize), rand()%(mapSize), rand()%
(mapHeight));
    }while( Monster == Lobby );
    // cout << "Monster" << Monster.x << " " << Monster.y << " " <<
Monster.z << endl;

    // Randomly generate Princess position
    do{
        Princess = Coordinate(rand()%(mapSize), rand()%(mapSize), rand()%
(mapHeight));
    }while(Princess == Lobby || Princess == Monster );
    // cout << "Princess" << Princess.x << " " << Princess.y << " " <<
Princess.z << endl;

    // Plan two paths from Lobby to Monster and Princess
    Link(Princess);
    Link(Monster);

    // Randomly connect rooms
    RandomLink();
}
```

- `void Link(const Coordinate& terminal)`
  - Find a path from Lobby to terminal, and set the room connectivity on this road

```
// Used to find the path from Lobby to terminal
void Map::Link(const Coordinate& terminal)
{
    // starting point
    Coordinate current = Lobby;
    while( current.x < terminal.x ){
        rooms[RoomNum(current)].link[0] = true;
        current.x++;
        rooms[RoomNum(current)].link[1] = true;
    }

    while( current.x > terminal.x ){
        rooms[RoomNum(current)].link[1] = true;
        current.x--;
        rooms[RoomNum(current)].link[0] = true;
    }

    while( current.y < terminal.y ){
        rooms[RoomNum(current)].link[3] = true;
        current.y++;
        rooms[RoomNum(current)].link[2] = true;
    }

    while( current.y > terminal.y ){
        rooms[RoomNum(current)].link[2] = true;
```

```
        current.y--;
        rooms[RoomNum(current)].link[3] = true;
    }

    while( current.z < terminal.z ){
        rooms[RoomNum(current)].link[4] = true;
        current.z++;
        rooms[RoomNum(current)].link[5] = true;
    }

    while( current.z > terminal.z ){
        rooms[RoomNum(current)].link[5] = true;
        current.z--;
        rooms[RoomNum(current)].link[4] = true;
    }
}
```

- `void RandomLink()`
  - Use the rand() function to randomly determine the connectivity of the room

```
// Randomly connect rooms
void Map::RandomLink()
{
    for( int i=0 ; i<mapHeight*mapSize*mapSize ; i++ ){
        // Randomly pick room coordinates
        int x = rand() % (mapSize-1);
        int y = rand() % (mapSize-1);
        int z = rand() % (mapHeight-1);
        // Randomly select the number of room connections
        int linkNum = rand() % 5;

        for( int j=0 ; j<linkNum ; j++ ){
            // Randomly select the direction of room connection
            int link = rand() % 6;
            switch( link ){
                case EAST:{
                    rooms[RoomNum(x,y,z)].link[3] = true;
                    rooms[RoomNum(x,y+1,z)].link[2] = true;
                    break;
                }
                case SOUTH:{
                    rooms[RoomNum(x,y,z)].link[1] = true;
                    rooms[RoomNum(x-1,y,z)].link[0] = true;
                    break;
                }
                case WEST:{
                    rooms[RoomNum(x,y,z)].link[2] = true;
                    rooms[RoomNum(x,y-1,z)].link[3] = true;
                    break;
                }
                case NORTH:{
                    rooms[RoomNum(x,y,z)].link[0] = true;
                    rooms[RoomNum(x+1,y,z)].link[1] = true;
                    break;
                }
                case UP:{
```

```
                    rooms[RoomNum(x,y,z)].link[4] = true;
                    rooms[RoomNum(x,y,z+1)].link[5] = true;
                    break;
                }
                case DOWN:{
                    rooms[RoomNum(x,y,z)].link[5] = true;
                    rooms[RoomNum(x,y,z-1)].link[4] = true;
                    break;
                }
            }
        }
    }
}
```

- `Other functions`

```
// Calculate the room number from the location information x, y, z
int Map::RoomNum( const Coordinate& room ) const
{
    return room.x + room.y*mapSize + room.z*mapSize*mapSize;
}

// Overloaded function
int Map::RoomNum(int x, int y, int z) const
{
    return x + y*mapSize + z*mapSize*mapSize;
}

Map::Map(int s, int h):mapSize(s), mapHeight(h)
{
    RoomAllocation();
}
```

2.5 Adventure Class

`Adventure Class` is the main class for running the game:

- `void Initialize()` : Game initialization
    - Choose difficulty level
    - Create a map based on the selected difficulty

```
// Game initialization
void Adventure::Initialize()
{
    int side, height,flag;
    char opt;
    cout << "Please select a map mode: " << endl;
    cout << "A.Easy\t" <<  "B.Moderate\t" << "C.Difficult" << endl;
    cin >> opt;

    // Game difficulty selection: Easy Moderate Difficult
    do{
        flag = 0;
        if( opt == 'A' ){
```

```
            side = 3;
            height = 3;
        }else if( opt == 'B' ){
            side = 5;
            height = 5;
        }else if( opt == 'C' ){
            side = 7;
            height = 7;
        }else{
            cout << "Illegal input, please re-enter: " << endl;
            flag = 1;
            cin >> opt;
        }
    }while(flag);
    string str;
    getline(cin, str);

    // Create a map according to the selected difficulty
    map = new Map(side, height);

    // The current location is the lobby
    curLoc = map->Lobby;
}
```

- `void Loop()` : Main game loop
  - Change current location based on user input
  - Determine if the game is over
    - Encounter a monster
    - Successfully rescued the princess

```
// Main game loop
void Adventure::Loop()
{
    bool FindPrincess = false;
    bool Victory = false;
    string str;
    bool bug;
    int dir;
    while(1){
        // cout << "curloc" << curLoc.x << " " << curLoc.y << " " <<
curLoc.z << endl;
        if( curLoc == map->Lobby )
            cout << "Welcome to the lobby. "<<endl;
        cout << "There are " << ExitsNum() << " exits as:";
        CoutExits();
        cout << "Enter your command:" << endl;

        // Ensure that the input is correct
        do{
            bug = false;
            getline(cin, str);
            if( str.substr(0,2) == "go" ){
                dir = Direction(str.substr(3));
            }
            else
```

```
                bug = true;

            if( dir == -1 ){
                bug = true;
            }else{
                // Change the current position according to the input
direction
                switch(dir){
                    case NORTH:
                        curLoc.x += 1;
                        break;
                    case SOUTH:
                        curLoc.x -= 1;
                        break;
                    case WEST:
                        curLoc.y -= 1;
                        break;
                    case EAST:
                        curLoc.y += 1;
                        break;
                    case UP:
                        curLoc.z += 1;
                        break;
                    case DOWN:
                        curLoc.z -= 1;
                        break;
                }
            }
            if( bug )
                cout << "Invalid input, please input again!" << endl;
        }while(bug);

        // Find the princess
        if( curLoc == map->Princess ){
            cout << "Congratulations on finding the princess, please take
her away!" << endl;
            FindPrincess = true;
        }

        // Met a monster
        if( curLoc == map->Monster ){
            cout << "You met a monster!!" << endl;
            break;
        }

        // Take the princess to the lobby and win the game
        if( curLoc == map->Lobby && FindPrincess ){
            Victory = true;
            break;
        }
    }
    if( Victory )
        cout << "Congratulations, lucky to win!" << endl;
    else
        cout << "Haha, you lose!" << endl;
}
```

- `Other Function`

```
#include "Game.h"

string Dire[]={"north","south","west","east","up","down"};

// Calculate the number of room exits
int Adventure::ExitsNum()
{
    Rooms room = map->rooms[map->RoomNum(curLoc)];
    int count = 0;
    for( int i=0 ; i<6 ; i++ )
        if( room.link[i] == true )
            count++;
    return count;
}

// Export each export direction
void Adventure::CoutExits()
{
    bool flag = false;
    Rooms room = map->rooms[map->RoomNum(curLoc)];
    for( int i=0 ; i<6 ; i++ ){
        if( room.link[i] == true ){
            if( flag )
                cout << ", " << Dire[i];
            else
                cout << " " << Dire[i];
            flag = true;
        }
    }
    cout << endl;
}

// The selected direction is obtained by user input
int Adventure::Direction(const string& s)
{
    for( int i=0 ; i<6 ; i++ ){
    Rooms room = map->rooms[map->RoomNum(curLoc)];
        if( s == Dire[i] && room.link[i] == true )
            return i;
    }
    return -1;
}

// Start the game
void Adventure::Start()
{
    if( curLoc != map->Lobby )
        curLoc = map->Lobby;
    Loop();
}
```

03.Testing Results

- **Easy**

```
D:\CPP\project2>test
Please select a map mode:
A.Easy  B.Moderate      C.Difficult
A
Welcome to the lobby.
There are 6 exits as: north, south, west, east, up, down
Enter your command:
go down
There are 4 exits as: south, west, up, down
Enter your command:
go west
Congratulations on finding the princess, please take her away!
There are 5 exits as: north, south, west, east, up
Enter your command:
go west
There are 3 exits as: east, up, down
Enter your command:
go down
There are 6 exits as: north, south, west, east, up, down
Enter your command:
```

- **Moderate**

```
D:\CPP\project2>test
Please select a map mode:
A.Easy  B.Moderate      C.Difficult
B
Welcome to the lobby.
There are 5 exits as: south, west, east, up, down
Enter your command:
go up
There are 5 exits as: north, south, east, up, down
Enter your command:
go north
There are 3 exits as: south, west, east
Enter your command:
go south
There are 5 exits as: north, south, east, up, down
Enter your command:
go south
There are 5 exits as: north, south, west, east, down
Enter your command:
go south
There are 3 exits as: north, east, down
Enter your command:
go down
There are 4 exits as: north, west, up, down
Enter your command:
go down
There are 6 exits as: north, south, west, east, up, down
Enter your command:
go north
There are 3 exits as: south, west, down
Enter your command:
```

- **Difficult**

```
D:\CPP\project2>test
Please select a map mode:
A.Easy  B.Moderate      C.Difficult
C
Welcome to the lobby.
There are 3 exits as: north, south, east
Enter your command:
go north
There are 4 exits as: north, south, east, up
Enter your command:
go south
Welcome to the lobby.
There are 3 exits as: north, south, east
Enter your command:
Invalid input, please input again!^C
D:\CPP\project2>test
Please select a map mode:
A.Easy  B.Moderate      C.Difficult
C
Welcome to the lobby.
There are 4 exits as: north, west, up, down
Enter your command:
go down
There are 5 exits as: north, south, east, up, down
Enter your command:
go down
There are 5 exits as: north, south, west, east, up
Enter your command:
go west
There are 3 exits as: south, east, up
Enter your command:
go west
Invalid input, please input again!
go south
There are 5 exits as: north, south, west, east, down
Enter your command:
```

> Let's play !!

04. Comments

- The code structure of this experiment is clear an readable, but the efficiency is low. Criticism and discussion are welcome.
- The map can only be a regular graph, and the scalability is not strong.

- The code for this experiment is long and can only be divided into modules dubug.
- After dividing files, the overall code is clearer.
- Try to write `.h` file to gain a deeper understanding of the declaration and definition.