Lee Zong Yang 100073484

Table of Contents

Task 1

Description

Develop a game environment with an event loop. Define a class of player's character and instantiate an object to represent the player during the event loop. Design room and monster in the room too. Testing should be done too.

Concept

I choose while loop instead of GOTO because it is a better programming practice. The reason to use a class instead of structure is due to class allow binding of data and function. Class in C++ is the building block, it allowed the software engineer to define a new custom data type. In other word, it is a blueprint for an object.

Implementation

```
int main()
{

    while(true) {
        system("cls");
        cout << "A New Game." << endl;
        /* initialize random seed: */
        srand(time(0));

        // player and dungeon
        Player p("Constantine");
        cout << "Player name: " << p.name << endl;

        Dungeon d;
        d.buildDungeon();

        // get starting room from Dungeon
        TreeNode<Room>* startingRoom = d.startingRoom;

        // visit the first room
        startingRoom->acceptVisitor(&p);

        // ask whether to move to left room or right room
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;
        cout << "(2)Right" << endl;

        // get input
        int selection;
        cin >> selection;
```

```
        // go to left room if input is 1
        if (selection == 1) {
            startingRoom->left->acceptVisitor(&p);
        }
        else {
            startingRoom->right->acceptVisitor(&p);
        }

        // ask whether to move forward or not
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // get input
        cin >> selection;

        // proceeed
        if (selection == 1) {
            startingRoom->left->right->acceptVisitor(&p);
        }

        // ask again
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // proceed
        if (selection == 1) {
            startingRoom->left->right->left->acceptVisitor(&p);
        }

        // success in finding the treasure
        cout << "You found the treasure !!!" << endl;
    }
}
```

Figure: Game Loop

```
class Monster
{
public:
    string monsterDes;

    Monster(string MonsterDes) :
        monsterDes(MonsterDes) {};

    Monster() {};

    friend ostream& operator<<(ostream& aOstream, Monster monster) {
        aOstream << monster.monsterDes << endl;
        return aOstream;
    }

    ~Monster() {};
};
```

Figure: Monster class

```
class Player: public Visitor
{
public:
    string name;

    Player(string Name) : name(Name) {}

    void visitDungeonRoom(TreeNode<Room>* node) {
        cout << endl << "Room Description:" << endl;
        cout << node->data.roomIntro << endl;
        cout << "Monster Description:" << endl;
        cout << node->data.monsters.monsterDes << endl;


        cout << "---Battle Start---" << endl;
        if (rand() % 2) {
            cout << "You win !!!" << endl ;
        }
        else {
            cout << "You lose TT" << endl;
            cout << endl << "Keep Play ?" << endl;
            int keepPlay = 1;
            cout << "(1) Yes" << endl;
            cout << "(2) No" << endl;
            cin >> keepPlay;
            if (keepPlay == 1) {

            }
            else {
                exit(0);
            }
        }
    }
};
```

Figure: Player class

```cpp
class Room
{
public:
    int id;
    string roomIntro;
    Monster monsters;

    Room() {};

    Room(int ID, string RoomIntro) : id(ID), roomIntro(RoomIntro) {
    };

    friend ostream& operator<<(ostream& aOstream, Room room) {
        aOstream << room.roomIntro << endl;
        return aOstream;
    }


    void addMonster(Monster monster) {
        monsters = monster;
    }

    ~Room() {};
};
```

Figure: Room class

Output

```
A New Game.
Player name: Constantine

Room Description:
Warm Room
Monster Description:
Fire Spirit
```

Figure: console output

Troubleshooting

The design of player class is a bit difficult to accommodate visitor pattern. Otherwise, like the task I attempted before so no problem.

Task 2

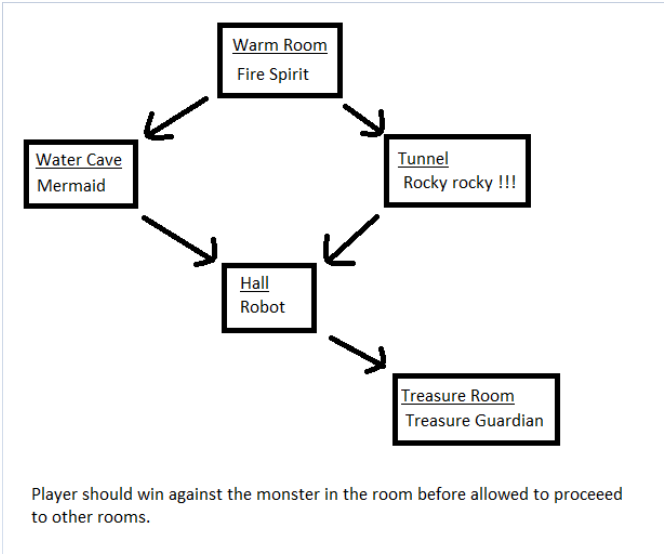
Description

A diagram should be produced to illustrate the dungeon. Indication of what monster each room contains and what actions are possible for the player to carry is required. As the dungeon has a tree structure, connection between rooms should be shown clearly.

Concept

Why uses a diagram? The main reason is the tree is a complicated concept and it is hard to visualize with text description. Humans learn and remember visually, thus the brain is designed to receive and interpret information visually. A pattern is far easier to spot than a list of numbers. Moreover, a diagram could reduce the lengthy time needed for reading and still managed to convey more information is less time.

Implementation



Player should win against the monster in the room before allowed to proceeed to other rooms.

Troubleshooting

To find out what software is the easiest to draw a simple diagram, it took me some time to try out several software on the market. In the end, I decided to use paint as it provide all the necessary feature with simple user interface.

Task 3

Description

This task is to build the in-game dungeon. First, design a node class (node of tree) to describe the room and keep track of its entities. Second, a dungeon class to manage all these nodes. At last, testing of node traversal to show that the tree is functioning correctly.

Concept

A tree data structure is defined as a collection of nodes, where each node composes of its value and references to its child nodes. It is one of the important concepts in computer science, notably in Artificial Intelligence field. The nodes are connected by edges. There are different varieties of tree, binary tree is one of the famous one. It has the benefits of both an ordered array and a linked list. The search is as fast as a sorted array since binary search could be carry out. The insertion or deletion are as fast as linked list.

Root is the node at the top of the tree. Parent is a node that is one edge upward to a node. Child is a node below a given node connected by an edge downward.

Implementation

```cpp
#pragma once
#include <stdio.h>
#include "Visitor.h"

template <class DataType>
class TreeNode
{
public:
    DataType data;
    TreeNode* left;
    TreeNode* right;

    void acceptVisitor(Visitor* visitor) {
        visitor->visitDungeonRoom(this);
    }
};
```

Figure: node class (TreeNode.h)

```cpp
#pragma once
#include "TreeNode.h"
#include "Room.h"

class Dungeon
{
public:
    TreeNode<Room>* startingRoom;

    Dungeon() {};

    void buildDungeon() {
        startingRoom = new TreeNode<Room>();
        Room room1(1, "Warm Room");
        Monster monster1("Fire Spirit");
        room1.addMonster(monster1);
        startingRoom->data = room1;

        TreeNode<Room>* secondRoom = new TreeNode<Room>();
        Room room2(2, "Water Cave");
        Monster monster2("Mermaid");
        room2.addMonster(monster2);
        secondRoom->data = room2;

        TreeNode<Room>* thirdRoom = new TreeNode<Room>();
        Room room3(3, "Tunnel");
        Monster monster3("Rocky rocky !!!");
        room2.addMonster(monster3);
        thirdRoom->data = room3;

        TreeNode<Room>* fourthRoom = new TreeNode<Room>();
        Room room4(4, "Hall");
        Monster monster4("Robot");
        room4.addMonster(monster4);
        fourthRoom->data = room4;


        TreeNode<Room>* fifthRoom = new TreeNode<Room>();
        Room room5(5, "Treasure Room");
        Monster monster5("Treasure Guardian");
        room5.addMonster(monster5);
        fifthRoom->data = room5;

        startingRoom->left = secondRoom;
        startingRoom->right = thirdRoom;
        secondRoom->right = fourthRoom;
        thirdRoom->left = fourthRoom;
        fourthRoom->left = fifthRoom;
    }
};
```

Figure: Dungeon class (Dungeon.h)

```cpp
int main()
{

    while(true) {
        system("cls");
        cout << "A New Game." << endl;
        /* initialize random seed: */
        srand(time(0));

        // player and dungeon
        Player p("Constantine");
        cout << "Player name: " << p.name << endl;

        Dungeon d;
        d.buildDungeon();

        // get starting room from Dungeon
        TreeNode<Room>* startingRoom = d.startingRoom;

        // visit the first room
        startingRoom->acceptVisitor(&p);

        // ask whether to move to left room or right room
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;
        cout << "(2)Right" << endl;

        // get input
        int selection;
        cin >> selection;


        // go to left room if input is 1
        if (selection == 1) {
            startingRoom->left->acceptVisitor(&p);
        }
        else {
            startingRoom->right->acceptVisitor(&p);
        }

        // ask whether to move forward or not
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // get input
        cin >> selection;

        // proceeed
        if (selection == 1) {
            startingRoom->left->right->acceptVisitor(&p);
        }

        // ask again
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // proceed
        if (selection == 1) {
            startingRoom->left->right->left->acceptVisitor(&p);
        }

        // success in finding the treasure
        cout << "You found the treasure !!!" << endl;
    }

}
```

Figure: main (main.cpp)

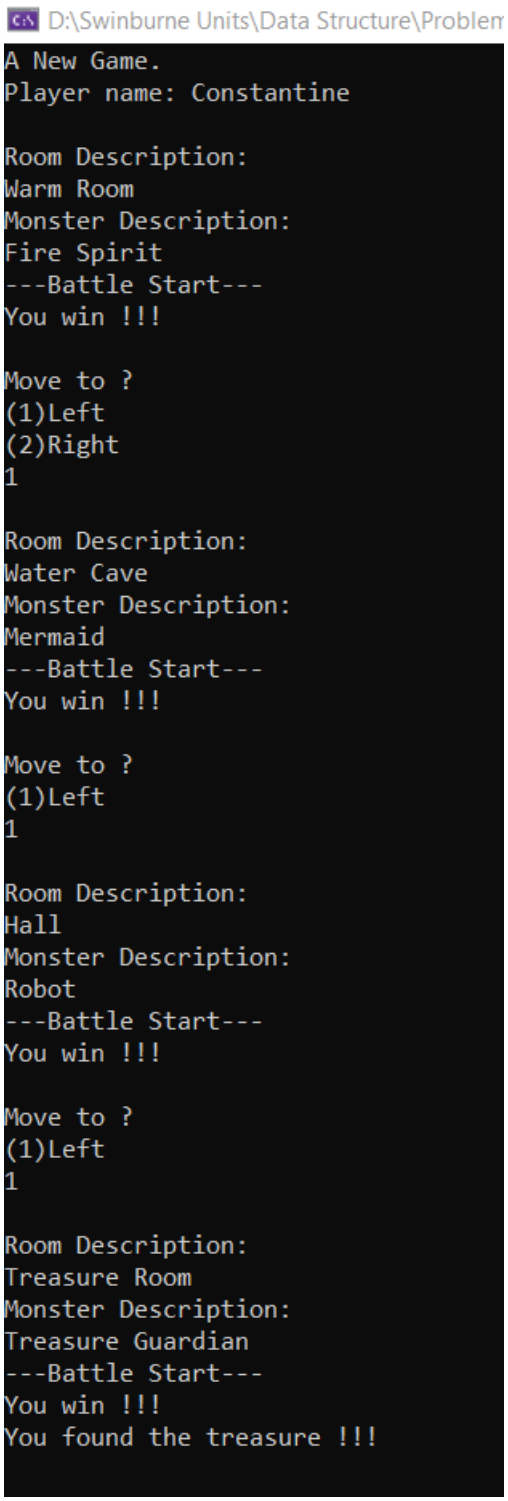


Figure: console output

Troubleshooting

The tree built in this task is different than the classic tree. Thus, I could only use the resources (https://stackoverflow.com/questions/931734/tree-datastructures) on the internet as reference and must think by myself. Luckily, the building of tree is hard coded, thus it is easier to debug through call stack.

Task 4

Description

Research about visitor design pattern and implement the player actions using it.

Concept

A way of separating an algorithm from an object on which it operates is what visitor design pattern trying to accomplish. By applying this design pattern, the software gains the ability to add new algorithm to existing object without modifying the object. This is in align with the open/closed principle. Through the visitor, new functions could be added to the family of existing classes, without modification on the classes. The means that the visitor class will be the one that implements all the new functions.

Code

```
#pragma once
class Room;
template <class DataType>
class TreeNode;

class Visitor
{
public:
    virtual void visitDungeonRoom(TreeNode<Room>* node) = 0;
};
```

Figure: Visitor interface (Visitor.h)

```
class Player: public Visitor
{
public:
    string name;

    Player(string Name) : name(Name) {}

    void visitDungeonRoom(TreeNode<Room>* node) {
        cout << endl << "Room Description:" << endl;
        cout << node->data.roomIntro << endl;
        cout << "Monster Description:" << endl;
        cout << node->data.monsters.monsterDes << endl;


        cout << "---Battle Start---" << endl;
        if (rand() % 2) {
            cout << "You win !!!" << endl ;
        }
        else {
            cout << "You lose TT" << endl;
            cout << endl << "Keep Play ?" << endl;
            int keepPlay = 1;
            cout << "(1) Yes" << endl;
            cout << "(2) No" << endl;
            cin >> keepPlay;
            if (keepPlay == 1) {

            }
            else {
                exit(0);
            }
        }
    }
};
```

Figure: Player class (Player.h)

```
template <class DataType>
class TreeNode
{
public:
    DataType data;
    TreeNode* left;
    TreeNode* right;

    void acceptVisitor(Visitor* visitor) {
        visitor->visitDungeonRoom(this);
    }
};
```

Figure: Tree node class (TreeNode.h)

```cpp
int main()
{

    while(true) {
        system("cls");
        cout << "A New Game." << endl;
        /* initialize random seed: */
        srand(time(0));

        // player and dungeon
        Player p("Constantine");
        cout << "Player name: " << p.name << endl;

        Dungeon d;
        d.buildDungeon();

        // get starting room from Dungeon
        TreeNode<Room>* startingRoom = d.startingRoom;

        // visit the first room
        startingRoom->acceptVisitor(&p);

        // ask whether to move to left room or right room
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;
        cout << "(2)Right" << endl;

        // get input
        int selection;
        cin >> selection;


        // go to left room if input is 1
        if (selection == 1) {
            startingRoom->left->acceptVisitor(&p);
        }
        else {
            startingRoom->right->acceptVisitor(&p);
        }

        // ask whether to move forward or not
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // get input
        cin >> selection;

        // proceeed
        if (selection == 1) {
            startingRoom->left->right->acceptVisitor(&p);
        }

        // ask again
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;

        // proceed
        if (selection == 1) {
            startingRoom->left->right->left->acceptVisitor(&p);
        }

        // success in finding the treasure
        cout << "You found the treasure !!!" << endl;
    }

}
```

Figure: main (main.cpp)

Output

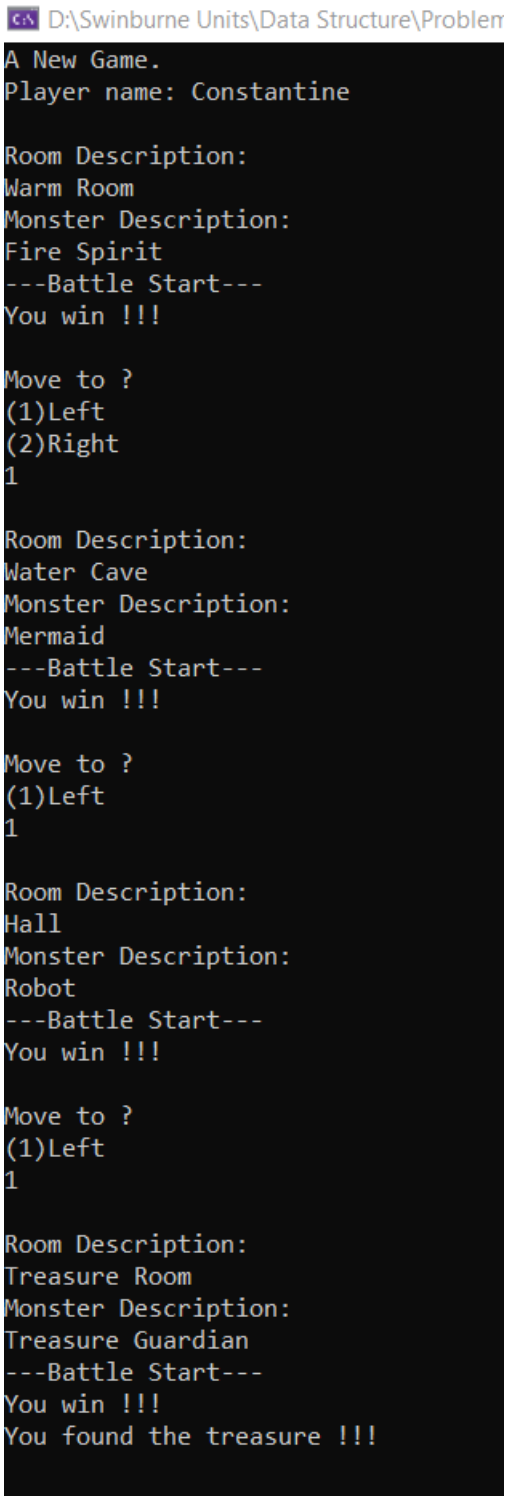


Figure: Console output

Troubleshooting

The understanding of visitor pattern is one of hardest task I had encountered in this unit. To be honest, the lecture slides are not helpful at all. Luckily, Dr.Mark provide with a youtube link

(https://www.youtube.com/watch?v=TeZqKnC2gvA&feature=youtu.be) that explain visitor pattern in layman term. This Stack Overflow question (https://softwareengineering.stackexchange.com/questions/333692/understanding-the-need-of-visitor-pattern) provides great help too.

Appendix

```
class Player: public Visitor
{
public:
    string name;

    Player(string Name) : name(Name) {}

    void visitDungeonRoom(TreeNode<Room>* node) {
        cout << endl << "Room Description:" << endl;
        cout << node->data.roomIntro << endl;
        cout << "Monster Description:" << endl;
        cout << node->data.monsters.monsterDes << endl;


        cout << "---Battle Start---" << endl;
        if (rand() % 2) {
            cout << "You win !!!" << endl ;
        }
        else {
            cout << "You lose TT" << endl;
            cout << endl << "Keep Play ?" << endl;
            int keepPlay = 1;
            cout << "(1) Yes" << endl;
            cout << "(2) No" << endl;
            cin >> keepPlay;
            if (keepPlay == 1) {

            }
            else {
                exit(0);
            }
        }
    }
};
```

Figure: Player class (Player.h)

```
#pragma once
class Room;
template <class DataType>
class TreeNode;

class Visitor
{
public:
    virtual void visitDungeonRoom(TreeNode<Room>* node) = 0;
};
```

Figure: Visitor class (Visitor.h)

```
class Monster
{
public:
    string monsterDes;

    Monster(string MonsterDes) :
        monsterDes(MonsterDes) {};

    Monster() {};

    friend ostream& operator<<(ostream& aOstream, Monster monster) {
        aOstream << monster.monsterDes << endl;
        return aOstream;
    }

    ~Monster() {};
};
```

Figure: Monster class (Monster.h)

```cpp
class Room
{
public:
    int id;
    string roomIntro;
    Monster monsters;

    Room() {};

    Room(int ID, string RoomIntro) : id(ID), roomIntro(RoomIntro) {
    };

    friend ostream& operator<<(ostream& aOstream, Room room) {
        aOstream << room.roomIntro << endl;
        return aOstream;
    }


    void addMonster(Monster monster) {
        monsters = monster;
    }

    ~Room() {};
};
```

Figure: Room class (Room.h)

```cpp
class Dungeon
{
public:
    TreeNode<Room>* startingRoom;

    Dungeon() {};

    void buildDungeon() {
        startingRoom = new TreeNode<Room>();
        Room room1(1, "Warm Room");
        Monster monster1("Fire Spirit");
        room1.addMonster(monster1);
        startingRoom->data = room1;

        TreeNode<Room>* secondRoom = new TreeNode<Room>();
        Room room2(2, "Water Cave");
        Monster monster2("Mermaid");
        room2.addMonster(monster2);
        secondRoom->data = room2;

        TreeNode<Room>* thirdRoom = new TreeNode<Room>();
        Room room3(3, "Tunnel");
        Monster monster3("Rocky rocky !!!");
        room2.addMonster(monster3);
        thirdRoom->data = room3;

        TreeNode<Room>* fourthRoom = new TreeNode<Room>();
        Room room4(4, "Hall");
        Monster monster4("Robot");
        room4.addMonster(monster4);
        fourthRoom->data = room4;


        TreeNode<Room>* fifthRoom = new TreeNode<Room>();
        Room room5(5, "Treasure Room");
        Monster monster5("Treasure Guardian");
        room5.addMonster(monster5);
        fifthRoom->data = room5;

        startingRoom->left = secondRoom;
        startingRoom->right = thirdRoom;
        secondRoom->right = fourthRoom;
        thirdRoom->left = fourthRoom;
        fourthRoom->left = fifthRoom;
    }
};
```

Figure: Dungeon class (Dungeon.h)

```cpp
template <class DataType>
class TreeNode
{
public:
    DataType data;
    TreeNode* left;
    TreeNode* right;

    void acceptVisitor(Visitor* visitor) {
        visitor->visitDungeonRoom(this);
    }
};
```

Figure: Tree node class (TreeNode.h)

```cpp
int main()
{

    while(true) {
        system("cls");
        cout << "A New Game." << endl;
        /* initialize random seed: */
        srand(time(0));

        // player and dungeon
        Player p("Constantine");
        cout << "Player name: " << p.name << endl;

        Dungeon d;
        d.buildDungeon();

        // get starting room from Dungeon
        TreeNode<Room>* startingRoom = d.startingRoom;

        // visit the first room
        startingRoom->acceptVisitor(&p);

        // ask whether to move to left room or right room
        cout << endl << "Move to ?" << endl;
        cout << "(1)Left" << endl;
        cout << "(2)Right" << endl;

        // get input
        int selection;
        cin >> selection;

       // go to left room if input is 1
       if (selection == 1) {
           startingRoom->left->acceptVisitor(&p);
       }
       else {
           startingRoom->right->acceptVisitor(&p);
       }

       // ask whether to move forward or not
       cout << endl << "Move to ?" << endl;
       cout << "(1)Left" << endl;

       // get input
       cin >> selection;

       // proceeed
       if (selection == 1) {
           startingRoom->left->right->acceptVisitor(&p);
       }

       // ask again
       cout << endl << "Move to ?" << endl;
       cout << "(1)Left" << endl;

       // get input
       cin >> selection;

       // proceed
       if (selection == 1) {
           startingRoom->left->right->left->acceptVisitor(&p);
       }

        // success in finding the treasure
        cout << "You found the treasure !!!" << endl;

        int wait;
        cin >> wait;
    }
}
```

Figure: main.cpp