# Project 2-Adventure

## 一、项目介绍

“Adventure”是一个在随机生成的城堡里营救公主的小游戏。

该游戏运行在命令行界面（CLI）下，使用简单命令来操作主人公在若干个已知或未知的房间里探索。

（游戏不会帮你记住探索的路线，也不会绘制城堡结构，只会描述每个房间的出口信息）

主人公从城堡外部向北通过大门进入城堡大厅（Lobby），开始营救公主的旅程。

在城堡深处找到公主后，玩家需要操纵主人公按照原路线返回城堡大厅，完成营救行动。

在找寻公主或与公主一同逃离城堡的过程中，如果不幸遇到怪物，则营救失败，游戏终止。

## 二、编译及操作说明

- 本项目的目录结构：
  - `./src/`:程序源代码文件(`adventure.cpp`)
  - `./bin/`:预编译好的可执行文件(Linux:`adventure`,Windows:`adventure.exe`)
- 该游戏使用C++编写，语法与GUN C++11标准兼容
- 推荐使用`Microsoft VSCode`编辑器浏览和修改源代码
- 修改源代码后运行测试，需要在项目根目录下执行：
  - Linux/WSL：

    `g++ -std=c++11 ./src/adventure.cpp -o ./bin/adventure`

    `./bin/adventure`
  - Windows(Dev-C++)：

    Windows资源管理器中打开`./src/adventure.cpp`

    `工具`-`编译选项`-`代码生成/优化`-`代码生成`-`语言标准(-std)`-`GNU C++11`-`确定`

    `运行`-`编译`
- 请在命令行环境下执行生成的可执行文件（bash/zsh/cmd/PowerShell）,桌面环境下双击执行可能导致程序结束前的输出不被观察到
- 测试过程中可能需要修改的几处代码（源代码节选）：

```
#define PROB_MONSTER 0.15 //可生成怪物房间生成静态怪物的概率
#define PROB_UNSAFE 0.3 //生成危险的房间的概率
#define PROB_UNSAFE_MONSTER 0.2 //危险的房间在游戏中动态刷出怪物的概率
//所有概率的取值范围均为[0,1]
#define CLS_COMMAND "clear" //清屏命令：Windows命令行为"cls"，Linux为"clear"

int main(void){
    Game adventure(10,2); //参数：(生成房间数,可生成连续楼层的层数)
    adventure.start();
    return 0;
}
```

- 游戏操作方法：

- 游戏启动后提示在城堡外，按下回车键(Enter)向北进入城堡大厅
- 在任何一个房间内，都会提示当前房间的类型和楼层号
- 同时，游戏会显示当前房间存在的所有的出口方向：`east`，`west`，`north`，`south`，`upstairs`，`downstairs`中的一个或多个(楼层内：东、西、南、北，楼层间：上楼、下楼)
- 根据提示的房间出口方向，键入`go 出口方向`进入该出口连接的另一房间
- 找到公主后，游戏提示`You meet the princess!`此时需要原路返回城堡大厅
- 与公主返回城堡大厅后，游戏提示`Congratulations!You rescued the princess!`并结束
- 若遇到怪物，游戏提示`You are eatten by the monster!Game Over!`并结束
- 在任意状态下，输入`exit`可以退出游戏

# 三、主要算法及数据结构

- 随机数与概率：
  - 使用系统时间产生伪随机数种子，通过求模来控制产生随机数的范围

    `::srand((int)time(NULL));`

    `#define random(x) (::rand()%(x))`
  - 概率模型使用一维几何概型（试验区间[0,100]的有效区间长度正比于设定概率），使用蒙特卡罗方法做生成随机数的试验来产生随机事件

```
bool prob_generator(double prob){
    if(prob>1) return true;
    else if(prob<=0) return false;
    else{
        if(random(100)+1>100*prob) return false;
        else return true;
    }
}
```

- 房间的类型及生成机制：
  - 游戏有三种类型的房间：
    - 大厅(Lobby)：进入城堡后第一个房间；找到公主后逃离城堡的目的地
    - 普通房间(Normal Room)：静态怪物存在的房间
    - 危险的房间(Dangerous Room)：动态怪物存在的房间
  - 选择五叉树结构作为房间关系的描述：节点代表房间；根节点代表大厅
  - 单一节点连接的至多五个子树代表除该的房间生成入口外每个出口连接的房间群
  - 通过递归地生成五叉树来生成房间，不存在环状结构（任意两个房间之间的路径唯一确定），生成过程受到允许楼层范围参数的制约
  - 单一节点连接的子树的个数是随机确定的，但这一随机生成子树的过程必须受到鸽洞（抽屉）原理的限制（子树的数量不能过多而导致某个子树中没有节点）
  - 通过各子树节点数分划来生成随机的房间拓扑关系（即树的形状），这一随机分划的过程也受到鸽洞原理的限制（不能导致某个子树分不到节点）
  - 唯一的公主在城堡房间的最深（某一个叶子节点）处
- 怪物的生成与刷新机制：
  - 怪物有两种类型：静态怪物在游戏开始前生成地图时随机产生；动态怪物在游戏运行时产生
  - 大厅使用静态怪物产生机制，且永远没有怪物
  - 普通房间产生静态怪物：一旦生成就会一直存在；没有怪物的普通房间永远不会刷出怪物
  - 危险的房间产生动态怪物：玩家每次造访该房间后，其怪物状态都会刷新一次

- 房间和怪物产生机制通过以下手段保证存在获胜的可能性：

  在根节点到公主所在叶子节点的路径上，

  - 不存在有怪物的静态房间
  - 所有动态房间的初始状态都没有怪物

- 关键类与游戏控制：
  - 关键类：

```
//所有类使用默认析构函数，析构在游戏结束、程序退出时按照FILO顺序进行
class Room{ //房间类，充当五叉树的节点
private:
    room_types room_type; //房间类型
    int room_level; //房间楼层号
    map<direction,Room*> room_linker; //使用STL Map实现五叉树的连接
    bool has_monster; //怪物存在标记
    bool has_princess; //公主存在标记
public:
    Room(int max_rooms,int room_level,bool gen_princess,directions
entrance_dir,Room* entrance_room); //构造函数（递归产生子树）
    inline string get_info(void); //获取房间信息字符串
    map<direction,Room*> get_exit(void){return room_linker;} //获取子树连
接信息
    void room_update(void); //更新动态怪物状态（若有）
    bool check_lobby(void) {return (room_type==lobby);} //检查是否回到大厅
    bool check_princess(void) {return has_princess;} //检查是否遇到公主
    bool check_monster(void) {return has_monster;} //检查是否遇到怪物
};

class Game{ //游戏状态类
private:
    Room* entrance; //指向大厅节点的指针（游戏入口）
    static int min_level; //最低楼层号（以-1为基数，随机产生，可以为负）
    static int max_level; //最高楼层号（以1为基数）
    //|min_level|<|max_level|，整体上看产生地上城堡而非地下城堡
    //静态成员变量在类外初始化，属于类而不属于具体对象，方便通过类来访问到
    bool found_princess; //判断行进方向（是否已经遇见公主）
public:
    Game(int max_rooms,int level_range); //初始化游戏状态
    void start(void); //开始游戏，CLI交互逻辑和树的访问逻辑所在处
     //房间类作为游戏状态类的友元类，便于生成城堡时读取楼层限制信息
    friend class Room;
};
```

  - 其他全局变量及函数：

```
//枚举类型typedef，增强代码可读性
typedef enum room_types{lobby,stable,unsafe} room_type;
typedef enum directions{north,west,upstairs,south,east,downstairs}
direction;
//枚举类型到字符串类型的映射表，便于输入输出
string room_types_dict[3]={"Lobby","Normal","Dangerous"};
string directions_dict[6]=
{"north","west","upstairs","south","east","downstairs"};

template <class T> //函数模板，用于获取泛型反查索引
int get_index(T* array,T elem,int size){
```

```
    int index;
    for(index=0;index<size;index++)
        if(array[index]==elem) break;
    if(index>=size) return -1;
    else return index;
}
```

## 声明

我声明该程序是由我独立完成的。