

北京邮电大学

Beijing University of Posts and Telecommunications

程序设计实践

“森林冰火人”设计报告



班 级: 2018211305, 2018211306, 2018211305

学 号: 2018211290, 2018211352, 2018211293

姓 名: 崔思颖、曹雅琳、刘小丫

学 院: 计算机学院

目录

程序题目	4
主要功能	5
开发环境	6
程序结构	6
类设计	6
模块设计	6
游戏流程模块	7
按键处理模块	10
人物移动模块	11
地形信息模块	12
地图部件模块	14
移动判断模块	15
程序实现	16
人物的移动与跳跃	16
按键处理——双线程并发	16
跳跃实现	17
地图信息的存储	19
分区信息	19
移动判断	19
区域	21

跳上高层	21
落入低层	21
机关交互.....	22
水池	24
箱子	24
蹦床	26
钻石	27
传送门	28
门	30
游戏开始与结束	30
开始	31
结束	31

程序题目

——实现一个规范的软件原型系统

描述：软件原型系统是验证技术可行性的一类重要的应用软件，在新产品研发和科研活动中被广泛应用。它是介于软件单一技术作业和软件产品之间的一种软件形态，对于综合利用程序设计单项技能提出了较高要求。通过本作业，考察学生规范编写代码、合理设计程序、灵活应用技术、协同组织合作等方面的综合能力。

本作业具体要求和分值分配如下：

Ø 基本功能要求（60 分）

针对一个主题（如：课程表、选课、备忘录、点餐等），在虚拟机或物理设备上，实现一个具有一定实用价值的软件。

保证变量命名、方法命名、类命名、代码块分割、注释等方面满足课堂上讲授的编码风格要求。（40 分）

包含程序设计文档，说明程序的主要功能、结构，如果使用了特定的算法，需要对给出算法的伪代码描述。（10 分）

包含程序测试文档，说明测试环境、测试方法、测试工具、测试例和测试结果。（10 分）

Ø 界面要求（10 分）

所实现的程序满足课堂讲授的界面要求。

具备良好的模块接口，使模块能够在其他项目中较为容易的复用（10 分）。

具备良好的人机交互界面，操作方式符合用户直觉，与主流程序的操作方式一致。图形界面需满足界面简洁、元素反差对照、空白分割、平衡恰当、元素对齐等要求。（10 分）

Ø 并发计算要求（10 分）

所实现的程序使用多进程/多线程等并发计算技术完成某项实际功能（例如：多用户同时在线操作或数据异步更新等）。

Ø 健壮性要求（10 分）

所实现的程序在用户输入错误数据、网络连接中断、鼠标随意点击拖拽等异常情况下不出现异常退出、挂起、输出内部错误信息等情况，能够提供用户易于理解和接受的反应。

Ø 扩展性要求（10 分）

所实现的程序正确的应用了设计模式，合理的使用了并发、持久化、资源池等技术，架构合理，为未来功能和性能的扩展做了相应准备。

主要功能

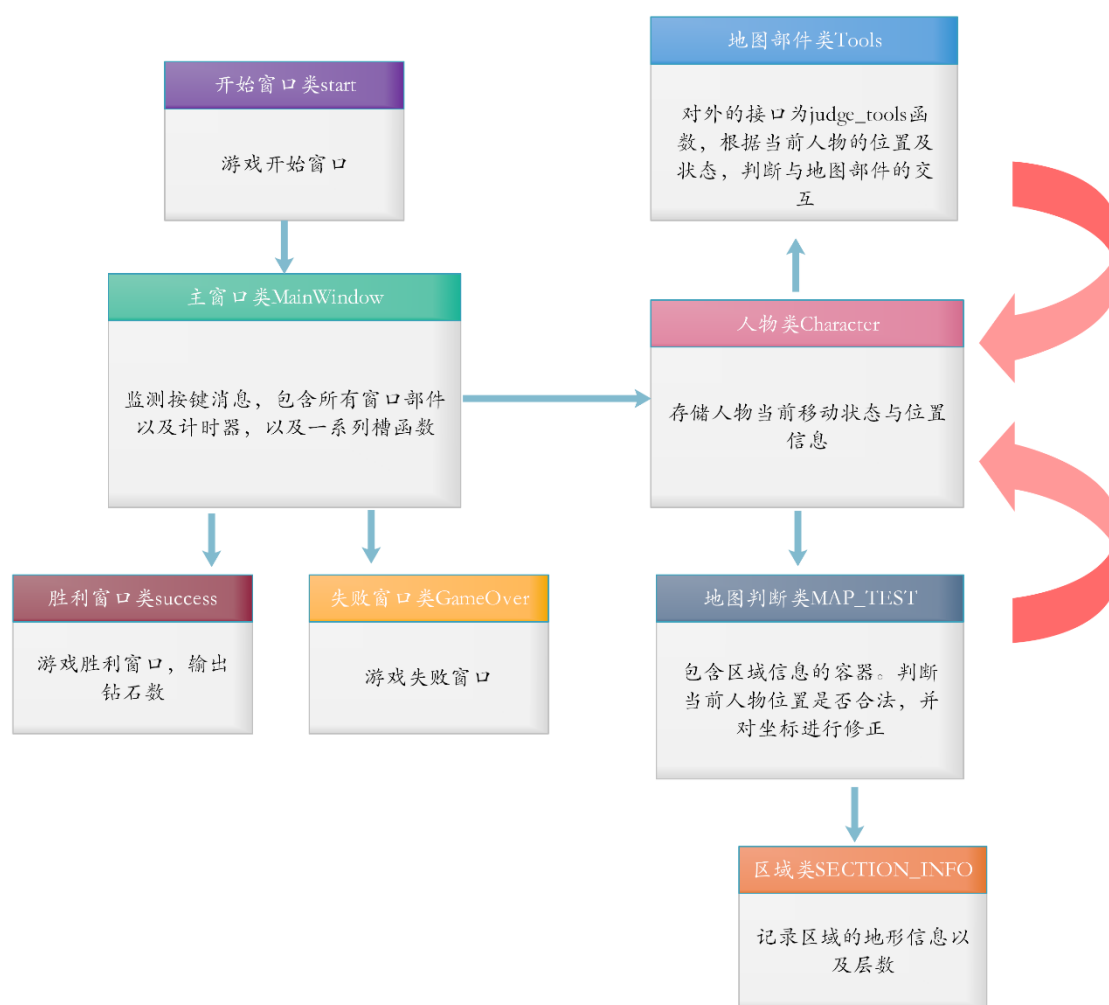
本次设计任务，我们将实现一款深受欢迎的游戏——森林冰火人。森林冰火人是一款经典的益智类闯关小游戏，在游戏中，你要帮助森林中的冰娃和火娃顺利过关，并收集所有的宝石。该游戏具有场景设计、部件贴图、人物移动、地图交互等元素，我们将基于 C++ 语言实现基本游戏人物控制功能、游戏角色与地图交互功能，并使游戏界面布局合理美观。

开发环境

Qt Creator 4.11.0 (Community)+Windows 10

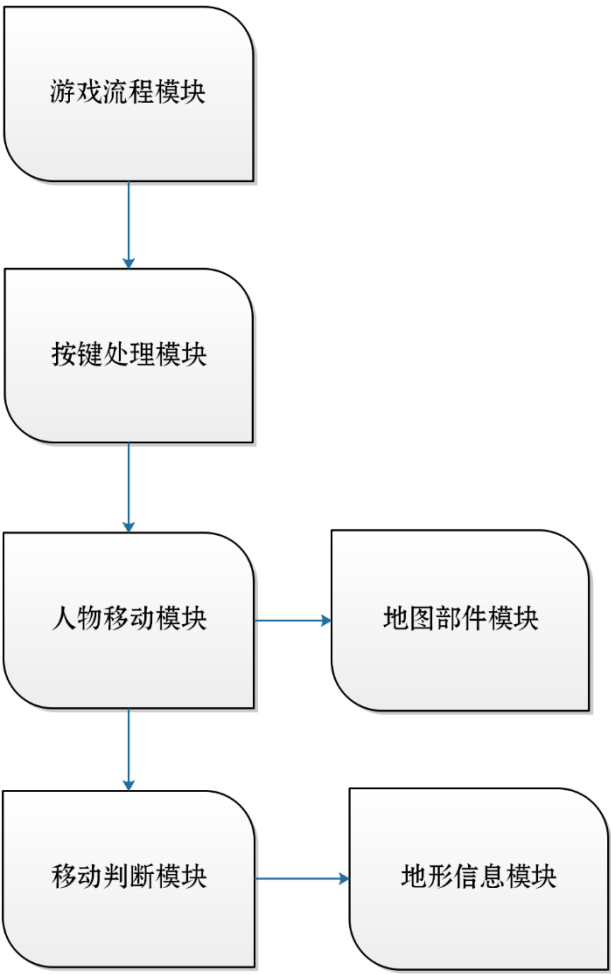
程序结构

类设计



模块设计

我们总共设计了六个模块：游戏流程模块、按键处理模块、人物移动模块、地形信息模块、地图部件模块、移动判断模块。各个模块较为独立、互相调用，具备良好的接口，能够在其他项目中较为容易的复用。



游戏流程模块

模块功能

在触发条件满足时，进行游戏开始界面、游戏主界面、游戏胜利界面、游戏失败界面的切换，从而实现游戏从开始到结束的整个流程。

数据结构

```
1.  /*游戏开始界面类*/
2.  class start : public QWidget
3.  {
4.      Q_OBJECT
5.
6.  public:
7.      explicit start (QWidget *parent = nullptr);
8.      ~start();
9.
10. private slots:
11.     void on_start_game_clicked(); //开始游戏按钮
12.
13. private:
14.     Ui::start *ui;
15. };
16.
17. /*游戏主界面类*/
18. class MainWindow : public QMainWindow
19. {
20.     Q_OBJECT
21.
22. public:
23.     MainWindow(QWidget *parent = nullptr); //移动背景窗口
24.     ~MainWindow();
25.     void keyPressEvent (QKeyEvent *event); //按键按下
26.     void keyReleaseEvent (QKeyEvent *event); //按键释放
27.     void judge_h_jump (Character *cha);
28.     void keyMove(); //处理按键消息处理移动
29.     //鼠标事件
30.     void mousePressEvent (QMouseEvent *event);
31.     void sleep(unsigned int msec);
32.     Ui::MainWindow *ui;
33.     MAP_TEST *map; //地图
34.     Tools tools; //地图部件
35.     Character *icegirl; //冰娃移动对象
36.     Character *fireboy; //火娃移动对象
37.
38. private:
39.     QTimer *keyTimer; //按键定时器
40.     QSet<int> pressedKeys; //保存按下的按键
41. signals: //信号的声明不区分 public 和 private
```



```

42. void gameover_signal(); //游戏失败
43. void success_signal(int, int); //游戏胜利
44.
45. private slots:
46. void gameover_slot() {
47.     gameover->show();
48.     w->hide();
49. };
50.
51. /*根据钻石数评级*/
52. void success_slot(int red_num, int blue_num) {
53.     success_window->show();
54.     success_window->ui->redNum->setText(QString::number(red_num));
55.     success_window->ui->blueNum->setText(QString::number(blue_num));
56.     if (red_num + blue_num >= 7) {
57.         success_window->ui->level->setText("A");
58.     }
59.     else if (red_num + blue_num >= 5) {
60.         success_window->ui->level->setText("B");
61.     }
62.     else if (red_num + blue_num >= 3) {
63.         success_window->ui->level->setText("C");
64.     }
65.     else {
66.         success_window->ui->level->setText("D");
67.     }
68.     w->hide();
69. };
70. };
71.
72. /*游戏失败界面类*/
73. class GameOver : public QWidget
74. {
75.     Q_OBJECT
76.
77. public:
78.     explicit GameOver(QWidget *parent = nullptr);
79.     ~GameOver();
80.
81. private slots:
82. void on_pushButton_clicked(); //再试一次按钮
83. void on_pushButton_2_clicked(); //返回菜单按钮
84. private:
85. Ui::GameOver

```

```

86. };
87.
88. /*游戏胜利界面类*/
89. class success : public QWidget
90. {
91.     Q_OBJECT
92.
93. public:
94.     explicit success(QWidget *parent = nullptr);
95.     ~success();
96.     Ui::success *ui;
97.
98. private slots:
99.     void on_pushButton_clicked(); //继续按钮
100.
101. private:
102.
103.     //ameOver game; //子窗口
104. };

```

按键处理模块

模块功能

收集键盘消息，根据按键内容对人物进行移动处理。

数据结构

```

1. #define KEY_TIME 53 //定时器在每次按键后接收按键消息的时间
2. QTimer* MainWindow::keyTimer; //按键定时器
3. QSet<int> MainWindow::pressedKeys; //保存按下的按键

```

主要函数

```

1. void MainWindow::keyPressEvent(QKeyEvent *event); //按键按下
2. void MainWindow::keyReleaseEvent(QKeyEvent *event); //按键释放

```

```
3. void MainWindow::keyMove() ;//处理按键消息处理移动
```

人物移动模块

模块功能

由按键消息，实现人物控制，包含向左、向右、跳跃三种移动。

数据结构

```
1. #define SPEED 15 //左右移动速度
2. #define JUMP_UPDATE_TIME 69//跳跃动画延迟时间
3. #define JUMP_V0 37 //跳起的初速度
4. #define JUMP_V0_1 50 //在蹦床上的跳起的初速度
5. #define JUMP_G -5 //重力加速度
6. typedef struct POINT {
7.     int x;
8.     int y;
9. } POINT; //坐标结构体
10. class Character
11. {
12. public:
13.     float y_floor; //当前所处的地面高度
14.     float dx_jump=0; //跳起时单位时间内水平位移
15.     int section_id; //section 区号
16.     bool fall_state=0; //是否在乎抛下落状态
17.     bool jump_state = 0; //是否在跳跃状态
18.     int jump_v0 = JUMP_V0; //跳起初速度
19.     bool on_box=0; //是否在箱子上
20.     bool on_button1 = 0;
21.     bool on_button2 = 0;
22.     bool h_jump = 0; //是否平抛
23.     bool enter ing = 0;
24.
25. Character (QLabel* label, bool sex); //传入 label 指针
26. ~Character ();
27. void jump_stop(); //停止跳跃
28. POINT get_pos()
```

```

29. {
30.     old_position.x = img->x();
31.     old_position.y = img->y();
32.     return old_position;
33. }
34. void move(int dx, int dy); //移动人物到新的坐标点, 形参为坐标变化量
35. void jump(); //根据 jump_state 处理人物跳跃
36. //void judge_open(POINT position); //根据位置判断是否该开门
37. //void get_diamonds(POINT position); //根据位置恰钻石
38. bool sex; //人物性别
39.
40. private:
41.     int jump_t = 0; //起跳时间单位数
42.     POINT new_position;
43.     POINT old_position;
44.     QLabel* img; //人物图标
45. };
46. QTimer *jumpTimer_ice; //冰娃跳跃刷新计时器
47. QTimer *jumpTimer_fire; //火娃跳跃刷新计时器

```

主要函数

```

1. void Character::move(float dx, float dy); //移动人物到新的坐标点, 形参为坐标变化量
2. void Character::jump(int speed_V0); //根据 jump_state 处理人物跳跃
3. void Character::jump_stop(); //停止跳跃
4. POINT Character::get_pos(); //得到人物当前位置坐标
5. void judge_h_jump(Character *cha); //判断当前是否进入平抛态

```

地形信息模块

模块功能

设计一个类，用于存储地图的空间信息，限制地图中的人物移动范围和机关位置，从而辅助人物完成游戏流程。

数据结构

```
1. //坐标
2. typedef struct POINT {
3.     int x;
4.     int y;
5. } POINT;
6. //边缘位置的状况信息
7. typedef struct BORDER_STATE {
8.     int state = 0;
9.     QVector<int> up_or_down_num;
10. } BORDER_STATE;
11. //地图分区的区域信息
12. typedef struct SECTION_INFO {
13.     int left_x; //区域左边的范围
14.     int right_x; //区域右边的范围
15.     int ceiling_y; //天花板高度
16.     int ground_y; //层级的地面高度
17.     int level; //记录层数信息
18.     BORDER_STATE left_status; //每区的左边界地形情况: 0 表示平地, 1 表示有跳
        跃向上平台, 2 表示有跳跃向下平台. 3 表示两者都有, 4 表示边界有墙
19.     BORDER_STATE right_status; //每区的右边界地形情况
20.     int is_Steep = 0; //判断是不是斜坡
21.     int is_Air = 0; //判断此处是不是空气区域
22. } SECTION_INFO; //每区的地形信息
23. //地图类
24. class MAP_TEST {
25. public:
26.     MAP_TEST(); //构造函数
27.     ~MAP_TEST(); //析构函数
28.
29.     //功能函数——判断运动范围
30.     POINT judge_move(POINT new_position, POINT old_position, Character *cha);
31.     int judge_section(POINT new_position); //判断在哪个区域
32.     SECTION_INFO get_section_info(int section_num); //获取某个区域的信息
33.     float get_Slope(POINT l, POINT r);
34.     BOX *box;
35.
36. private:
37.     QVector<SECTION_INFO> section_vector; //创建 section 动态数据结构
38. };
```

地图部件模块

模块功能

根据游戏规则在地图的特定位置设置了一些部件，是玩家完成游戏质量的评判标准，也辅助玩家完成游戏，玩家与部件的交互也是本游戏的趣味所在。

数据结构

```
1. public:
2.   int blue_open = 0; //判断冰娃是否到门口
3.   int red_open = 0; //判断火娃是否到门口
4. private:
5.   Character *cha; //人物
6.   POINT new_position; //人物坐标更新
7.   int section_tools[41][3]; //存储每个区中的部件
8.   bool on_button1; //是否踩着按钮 1
9.   bool on_button2; //是否踩着按钮 2
10.  int diamonds[8] = {0}; //存储每颗钻石被吃掉的情况避免重复计数
```

主要函数

```
1. public:
2.   void judge_tools(Character *cha, POINT new_pos); //判断部件的函数
3.   Tools();
4.   ~Tools();
5. private:
6.   void push(int direction); //推动箱子
7.   void judge_box(); //判断与箱子交互
8.   void judge_button1(); //判断是否站在 button1 上
9.   void judge_button2();
10.  void judge_trampoline(); //判断是否在蹦床范围内
11.  void judge_blue_pool(); //水池
12.  void judge_red_pool(); //火池
```

```
13. void judge_green_pool(); //绿池
14. void portal_show(); //传送门瞬移
```

移动判断模块

模块功能

移动模块部分，是在人物移动模块和地形信息模块的基础上，实现对于当前人物所在位置的限制判断。

数据结构

```
1. //边缘状况定义
2. #define SIMPLE 0 //和隔壁区域直接相连
3. #define JUMP_UP 1 //有向上的平台，且没有墙
4. #define JUMP_DOWN 2 //有向下的平台
5. #define JUMP_BOTH 3 //既可以向上，也可向下
6. #define WALL 4 //墙壁
7. #define JUMP_UP_WALL 5 //有墙且可以跳上
8. //人物的图像宽度和高度
9. #define WIDTH 120 //人物图像宽度
10. #define HEIGHT 90 //人物图像高度
```

主要函数

```
1. //存储地图分区信息，构造地图
2. MAP_TEST::MAP_TEST() {}
3. //map_test 类中，判断移动的核心函数
4. POINT judge_move(POINT new_position, POINT old_position, Character *cha);
5. //map_test 类中 判断在哪个区域
6. int judge_section(POINT new_position);
7. //map_test 类中 获取某个区域的信息
8. SECTION_INFO get_section_info(int section_num);
9. //Character 人物类中，调用地图类中的判断移动函数，确定如何移动人物
10. void Character::move(float dx, float dy);
```

程序实现

我们的程序基本架构较为合理，在此基础上细化了各项功能实现，窗口各个组件都能较为容易地添加到程序中，这为未来功能和性能的扩展做了相应准备，增加关卡与提升可玩性都是可行的优化方向。以下是各项功能具体实现的方法。

人物的移动与跳跃

按键处理——多线程并发

冰娃的移动由方向键“←”“↑”“→”控制，火娃的移动由“A”“W”“D”控制。在程序中，我们创建了两个线程，线程1收集按键消息，并把按键的消息放入集合 `pressedKeys` 中，线程2每隔一段时间专门处理 `pressedKeys` 中的按键消息。伪代码如下：

线程1：

if 按键按下

 if 非长按按键

 将按键加入 `pressedKeys`

 if `keyTimer` 定时器未启动

 启动 `keyTimer` 定时器

if 按键释放

 if 非长按按键

移除 pressedKeys 中该按键

if pressedKey 为空

提前结束 keyTimer 定时器

线程 2:

if keyTimer 计时结束

停止 keyTimer

处理 pressedKeys 中的按键消息

switch(key)

case Qt::Key_Up: 冰娃跳跃

case Qt::Key_Right: 冰娃向右

case Qt::Key_Left: 冰娃向左

case Qt::Key_W: 火娃跳跃

case Qt::Key_D: 火娃向右

case Qt::Key_A: 火娃向左

处理人物平抛

跳跃实现

我们发现，在人物跳起之后，只要没有到达地面，跳跃的状态会一直持续下去。因此，为了实现跳跃画面的连续播放，我们又设置了另外两个跳跃定时器 jumpTimer_ice, jumpTimer_fire，分别用于冰娃与火娃的跳跃，我们将这两个定时器的时长视为一个单位时间

jump_t，跳跃的初速度为 jump_v0，重力加速度为 jump_g，则第 jump_t+1 个时间单位时人物改变的高度为：

$$\begin{aligned} dh &= \text{jump_v0} * (\text{jump_t} + 1) + 1/2 * \text{jump_g} * (\text{jump_t} + 1)^2 - \\ &\quad (\text{jump_v0} * \text{jump_t} + 1/2 * \text{jump_g} * \text{jump_t}^2) \\ &= \text{jump_v0} + \text{jump_g} * \text{jump_t} + 1/2 * \text{jump_g} \end{aligned}$$

跳跃总时长为：jump_t=2*jump_v0/jump_g

实现跳跃的算法伪代码如下：

if 跳跃计时器计时结束

 if 非跳起状态

 return

else

 if jump_t < 2*jump_v0/jump_g || 未落到地面

 dh = jump_v0 + jump_g * jump_t + 1/2 * jump_g

 横坐标改变跳跃期间的水平偏移量 dx_jump，纵坐标改变 dh

 jump_t++

 else

 跳跃停止，调用 jump_stop()

 dx_jump=0

暂停计时器

if jump_state==1

 再次启动计时器

地图信息的存储

分区信息

地图采取分区，根据平面和天花板高度进行分区。每个区域包含左右坐标范围、上下坐标范围、边界状况信息、特殊情况（是不是斜坡、可跳跃区等）。每个区域构造完成后存入 **Vector** 数据结构，供人物移动时进行限制

移动判断

此处对人物的移动进行限制。伪代码如下：

if 人物所处的位置已经进入墙壁：

 人物此次移动卡在原来的位置。

else if 人物处在斜坡区：

 更新人物的 **y_floor** 坐标；

 人物水平方向自由移动；

 人物竖直方向上根据 **jump_state** 和人物的移动斜率进行判断：

 if 人物处在斜坡的地面处，或者以下：

 人物坐标处在地面

 else 人物处在斜坡的地面之上

 根据人物的 **jump_State** 进行平抛或者自由移动的判断

 if 没有跳跃：

 平抛；

else :

自由移动

else :人物处于正常区域

更新区域的 y_floor

判断 y 方向上的限制:

if 人物位置顶到了天花板:

人物此次移动卡在原来的位置;

else if 人物到了地面以下的区域

人物回复到站立在地面上

else 人物处在正常区域

人物自由移动

判断 x 方向上的限制:

if 到达左边,

if 左边是墙或者可跳跃高墙:

x 坐标不变;

else x 坐标自由变化

else if 到达右边

if 左边是墙或者可跳跃高墙:

x 坐标不变;

else x 坐标自由变化

else 在区域之后

x 坐标自由变化

区域

根据传入的当前坐标，判断人物所在的区域：

for(遍历区域)：

if 人物中心点在当前区域范围内：

返回当前区域标号

如果没有找到：

返回-1，表示错误。

跳上高层

跳上高层的实现在移动判断函数中已经实现。跳上高层时，人物在 y 坐标上自由移动，在 x 坐标上受制于边界处的信息条件。

落入低层

在人物落入下一层时，系统将执行 jump 函数，起跳初速度很小，设置为 3，进而模拟平抛运动。伪代码如下：

if 更新后的地面坐标大于更新前的地面坐标 && 无斜坡 && 当前没有站在箱子上

```
if !jump_state  
  
    jump_state=1  
  
    fall_state=1  
  
    jump(JUMP_V0)  
  
    启动跳跃计时器
```

之后与跳跃处理过程相同。

机关交互

地图分区如下图所示：




用 `section_tools` 数组来存储 41 个区里部件的出现情况，在本次设计的关卡中，最多一个区有 4 个部件，因此开辟一个 $41*4$ 的二维数

组，根据人物当前所处位置获得分区，遍历数组中该分区范围内的部件来依次判断是否实现部件交互。

部件使用说明如下：

部件名称	部件图例	使用说明
冰娃		女性游戏角色
火娃		男性游戏角色
钻石		冰娃只能获得蓝色钻石，同样，火娃也只能获得红色钻石 钻石数目会累加，数量多少直接影响完成游戏的等级
箱子		可以左右推动箱子，当跳跃高度不足以到达下一层时，可以借助箱子继续游戏
蹦床		跳跃高度不够时，站在蹦床范围内区域可以获得较大的弹跳高度
传送门		按住按钮，传送门就会出现 传送门可以将人物从一端直接传送到另一端 必须在按住按钮的时候传送门才有效，需要两个角色相互配合才能使用此道具
终点门		冰娃站在蓝色门前 火娃站在红色门前时， 门会呈现打开状态 当两个人物都到达对应门前， 玩家完成本关游戏，游戏结束
水池		红色池子会杀死冰娃，蓝色池子杀死火娃，绿色池子杀死两个角色

亡魂		当人物进入不兼容水池被杀死时显示的图标
----	---	---------------------

水池

根据森林冰火人的游戏规则，冰娃遇到红色水池、红娃遇到蓝色水池、冰娃火娃遇到绿色水池都会使游戏失败，触发信号量，弹出游戏失败界面。

在每次移动时，都需要判断人物是否处于与自己不兼容的水池范围，判断逻辑如下：

以红池为例

```
if(当前角色是冰娃 && new_position 位于红色水池范围) {
    隐藏人物角色图标;
    显示亡魂图标;
    触发游戏结束信号量显示 gameover 界面;
}
```

如果是与自己兼容的水池，则能够淌过水池抵达对岸，此处在水池底部添加分区，使得界面显示与行走在平地相区别。

箱子

箱子用于帮助人物跳上比较高的平台，我们定义了箱子类 BOX 如下：

```
class BOX
{
public:
    BOX(QLabel *label)
    {
        img = label;
        pos.x = label->x();
        pos.y = label->y();
    };

    void push(int direction); // 推箱子，传入方向，direction
    // 向左为-1，向右为1

    bool judge_push(PPOINT new_pos, PPOINT old_pos, bool
on_box); // 判断人物是否推动箱子

private:
    PPOINT pos; // 位置
    QLabel *img; // 贴图
};
```

其中，主要的函数接口为 judge_push，当人物处于箱子所在的 section 中时，会调用此接口来判断人物与箱子的交互。

judge_push 伪代码如下：

if 人物未高于箱子 && 人物未站在箱子上

if 人物紧贴箱子左边 && 人物向右移动

箱子被向右推动

if 人物紧贴箱子右边 && 人物向左移动

箱子被向左推动

else 人物在箱子宽度范围内

跳上箱子

跳上箱子后将改变 `y_floor` 的值为箱子高度，标志人物是否站在箱子上的状态变量 `on_box` 被值 1，若未跳上箱子则置 0。

蹦床

地图上有部分区域超出人物跳跃高度范围，需要借助工具完成游戏。蹦床可以使人物获得跳跃技能，比平地起跳跳得更高从而跳上更高的台阶。

实现逻辑：判断人物移动的坐标是否在蹦床的范围内，如果处于蹦床范围内，则应该加大起跳的初速度，获得更高的跳跃高度。

在人物类中添加 `is_trampoline` 属性，标记当前人物是否位于蹦床范围内，若是，则处理键盘 up（冰娃）或 W（火娃）时，`jump` 函数的参数为 `JUMP_V0_1`，常量值为 50，如果不在蹦床范围内起跳则 `jump` 函数传进的参数为 `JUMP_V0`，常量值为 37。

伪代码：

```
if(new_position 在蹦床范围内 && 人物不在跳跃状态){  
    将人物的起跳初速度改为 JUMP_V0_1; //定义的常量  
}
```

需要在其他区里将起跳的初速度设置回 JUMP_V0。

钻石

在本次程序的关卡中，共 8 颗钻石，蓝色红色各四颗，蓝色钻石需要由冰娃获得，红色钻石需要由火娃获得。在通关过程中获得的钻石数会决定游戏完成后的等级值。7~8 颗将显示 A 级，5~6 颗为 B 级，3~4 颗为 C 级，3 颗以下为 D 级。

需要先判断当前人物的身份，如果是冰娃只能获得蓝色钻石，在每次移动时都需要判断是否在能够获得钻石的区域，如果在区域内，则全局变量加一，钻石的 label 设置为不可见。用一个长度为 8 的数组记录每颗钻石被获得的情况，避免用坐标判断吃钻石重复计数的问题。

伪代码：

```
if(当前角色是冰娃 && new_position 在某颗蓝色钻石范围内)  
{  
    该钻石 label 设置为不可见;
```

```
if(diomands[当前钻石编号]==0) {  
    已获得的蓝钻石数++;  
}  
  
diomands[当前钻石编号]=1;  
}
```

传送门

第三种使人物到达超出跳跃高度的方式是传送门，此方式由两个按钮和两个传送门组成：当至少一个人物踩住按钮时，传送门出现，另一个人物可以通过一个传送门直接到达另一个传送门的位置，如果踩住按钮的人物离开按钮区域，传送门就会消失。此方式需要两个冰火人相互配合共同协作，增大了游戏的有趣性。

由于地图的位置特点，进入两个传送门都是向左移动时进入，如果长按左键就会出现人物在两个门之间来回闪现的情况，为避免此情况出现，在每次传送之后都会 `sleep 1` 秒钟，进而提高游戏体验感。

此功能的实现需要在人物类中判断人物位置是否在按钮范围内，并且对人物属性 `on_button` 进行赋值，再在 `mainwindow` 类中根据两个人物的位置和按钮情况进行图标显示：

如果至少有一个人物按住按钮，则显示传送门的图标，如果没有人物按住按钮则传送门不可见；根据按住按钮的人物身份以及该人物所处的坐标信息，判断哪一个按钮该被显示。

button 判断伪代码:

```
if(当前人物位于按钮范围内) {  
    当前人物的 on_button 属性设置为 true;  
}  
  
else {  
    on_button 属性设置为 false;  
}  
  
if(至少一个人物在 button 上) {  
    button 的 label 设置为不可见;  
}  
  
else {  
    button 的 label 可见;  
}
```

传送门伪代码:

```
if(当前至少有一个按钮被按下) {  
    if(当前人物在传送门门口) {  
        将人物移动到另一个传送门门口;  
    }  
}
```

门

当两个人物都站在最终结束的门口时，意味着玩家顺利通过关卡，可以进入下一关游戏。只需要根据人物的身份及人物坐标，如果人物在自己对应的门范围内，则显示打开的门的图标，否则将图标隐藏，当两个人物都处于自己的门口，则触发游戏成功信号，显示 `success` 界面，提示获得的钻石数以及游戏得分等级。

伪代码：

```
if (当前人物位于与自己颜色匹配的门口) {  
    将该人物对应的全局变量 door_open 置 1;  
    开门图标显示 label;  
}  
  
else {  
    开门图标 label 不可见;  
    door_open 置 0;  
}  
  
if (两个人物的 door_open 都为 1) {  
    触发信号量显示游戏成功界面，统计钻石数，给出评分等级;  
}
```

游戏开始与结束

游戏的开始和结束时，均需要进行窗口切换和信号量传递。

开始

游戏开始时，将会显示游戏规则说明和“开始游戏”按钮，点击

“开始游戏”即可进行游戏。伪代码如下：

显示开始窗口，

设置开始窗口属性

if 开始游戏按钮被点击：

关闭开始界面，

显示游戏窗口

结束

当角色死亡或者通过关卡时，将会调用游戏的两个结束界面。

角色死亡时，调用“游戏结束”界面，伪代码如下：

显示游戏窗口，

角色死亡触发信号量

游戏窗口阻塞且关闭

“游戏结束”窗口显示

if 点击“再玩一次”或者“返回菜单”按钮：

本窗口关闭

游戏窗口显示

角色通关时，调用“通关成功”界面，伪代码如下：

显示游戏窗口

游戏中记录钻石数

角色通关触发信号量，传递钻石数目

游戏窗口阻塞且关闭

“通关成功”界面设置

“通关成功”界面显示

if 点击“再玩一次”按钮：

本窗口关闭

游戏窗口显示