《推箱子》游戏C语言设计实验报告

1. 选题

【游戏题目】

推箱子

【游戏软件功能】

推箱子游戏中玩家通过控制键盘的上下左右键即通过与键盘操作界面的交互控制人物和箱子的移动，通过主菜单的“退出游戏”选项退出游戏，通过“重完本关”重置游戏本关。当人物靠近箱子并朝箱子位置移动的时候，箱子在其移动方向没有被围墙阻碍的情况下可以被移动。

玩家通关即将游戏中的箱子都送到规定的箱子目的地。

玩家没有通关即为箱子推错位置，游戏中的箱子没有到达目的地，却无法再进行移动，箱子的两个方向（左方和上方，左方和下方，右方和上方，右方和下方）都是围墙或者箱子。

1. 方案

【功能模块划分】

1. 初始化模块

（1）.关卡初始化

初始化每一关卡，设置构建每一关卡的围墙，人物位置，目的地位置，箱子个数以及位置。

通过txt格式保存文件分别记录每一关。

（2）.屏幕初始化

初始化屏幕，设置游戏背景。

1. 人物模块

（1）.移动人物

控制人物进行上、下、左、右移动，从而控制箱子移动到目的地。

（2）.画人物

通过8张位图（两张朝左移动，两张朝右移动，两张朝上移动，两张朝下移动）分别展现人物在进行不同移动时的动态变换。

3.箱子模块

（1）.移动箱子

控制箱子移动，可以进行目的地与空地之间，空地与空地之间，空地与目的地之间，目的地与目的地之间的移动。箱子移动的结果有本身不具有移动能力，没有移动，移动到目的地，离开目的地。

（2）.画箱子,

通过两张位图分别表示在空地、目的地的处于不同位置的箱子。

4.总控模块

该模块控制游戏，通过结合包括初始化模块，人物模块，箱子模块并调用每个模块的相关函数实现整个游戏控制，同时判断游戏是否通关，根据玩家需求初始化游戏，重置游戏等。

【数据结构设计】

1. 结构体

（1）.共同属性，基础类

struct  BASE

{

enum BASEType m\_type; // 类型

int m\_x; // 横向坐标

int m\_y; // 纵向坐标

HDC m\_drawer; // 绘图者

HBITMAP m\_Down1;     // 自身图像向下移动的图像1

HDC m\_HDC;        // 自身图像保存者

void\* m\_child; // 子类对象

  /\*对基本对象的基本操作\*/

BASEDraw draw;

BASEMove move;

BASEDelete delete;

};

基础类包含箱子，人物，通道，背景，围墙，目的地等元素的共同属性。

（2）.游戏区域gameboard

struct GameBoard

{

HWND m\_handle; // 窗口

HBITMAP m\_bitmap; // 后台缓冲位图

HDC m\_drawer; // 绘图者

GameBoardDelete delete;

};

struct GameBoard\* New\_GameBoard(HWND parent, int x, int y, int nWidth, int nHeight)

（3）.箱子 Box

struct Box

{

struct BASE\* m\_parent; // 用于继承属性的父类对象

struct BASE\* m\_below;     // 位于当前元素对象下面的元素对象

HBITMAP m\_Down2;     // 向下移动的图像2

HBITMAP m\_CurrentImage; // 当前显示的图像

void\* m\_child; // 子类对象

};

（4）.人物 Man

struct Man

{

    struct Box\* m\_parent; // 用于继承的父类对象

HWND m\_gameBoard; // 游戏区域

HBITMAP m\_Up1;     // 向上移动图像1

HBITMAP m\_Up2;     // 向上移动图像2

HBITMAP m\_Left1;     // 向左移动图像1

HBITMAP m\_Left2;     // 向左移动图像2

HBITMAP m\_Right1;     // 向右移动图像1

HBITMAP m\_Right2;     // 向右移动图像2

HBITMAP m\_image1; // 当前变换图像1

HBITMAP m\_image2; // 当前变换图像2

ManMove move;

ManDancing dancing;

};

（5）.游戏

struct Game

{

 HWND m\_mainWindow; // 主窗口

HWND m\_gameBoard; // 游戏区域

HDC m\_drawer; // 绘图者

enum Status m\_status; // 游戏状态

struct Man\* m\_man; // 小人

int m\_nLevel; // 关卡数

int m\_nBoxCount; // 未到达目的地的箱子数

GameDelete delete;

GameReplay replay;

GameGotoLevel gotoLevel;

GameOnKeyPress onKeyPress;

GameOnTimer onTimer;

};

（6）.MainWindow

struct MainWindow

{

HWND m\_handle;//主窗口

struct GameBoard\* m\_gameBoard;//游戏区域

struct Game\* m\_game;//游戏

MainWindowDoMoal doModal;

};

1. 数组

定义二维数组

struct Base\* game[WGameBoardGridsCount][ HGameBoardGridsCount]，作为整个游戏的的基础存储数据结构，数组中的每个元素为基础类（具有背景，人物，箱子，通道，目的地，围墙的共同属性），根据存储的数据元素，其中Base中的Basetype m\_type等被赋予不同的值。

1. ­­­­­­枚举类型

（1）.数组中每个元素的类型

enum BASEType

{

WALL = 1, // 墙

BOX = 2, // 箱子

SPACE = 3,     //空地

DESTINATION = 4, // 箱子的目的地

MAN = 5, // 人物

BACKGROUND = 0 // 背景

};

（2）.移动结果

enum MoveResult

{

NORMAL\_MOVED, // 正常移动

CANNOT\_MOVE, // 本身不具有移动能力

NO\_MOVE, // 没有移动

MOVE\_TO\_DEST,     // 移动到目的地

LEFT\_FROM\_DEST // 离开目的地

};

（3）.游戏状态

enum Status

{

PLAYING, // 游戏中

END      // 结束

};

【数据】

1. 格子

格子为构成游戏区域的基本元素，需要定义格子的尺寸

#define GRIDS\_SIZE 30 // 尺寸

1. 游戏区域

通过结构体定义游戏区域，并通过函数实现具体游戏区域的相关操作

（1）.结构体

struct GameBoard

{

HWND mHwnd；//游戏窗口

HBITMAP mBitmap；//后台缓冲位图

HDC mDrawer；//绘图者

GameBoardDelete deletegame;//删除游戏窗口区域函数

}\*GameBoard；

（2）.宏常量

<1>.游戏区域名字。

#define GAMEBOARD\_NAME \_T("GameBoard")

<2>.由格子构成的游戏区域的大小（宽度\*高度），包括横向和纵向格子数目。

#define GAMEBOARD\_WIDTH\_GRIDS 12

#define GAMEBOARD\_HEIGHT\_GRIDS 10

#define GAMEBOARD\_WIDTH (GRIDS\_SIZE \* GAMEBOARD\_WIDTH\_GRIDS)

#define GAMEBOARD\_HEIGHT (GRIDS\_SIZE \* GAMEBOARD\_HEIGHT\_GRIDS)

<3>.游戏区域的背景颜色。

#define GAMEBOARD\_COLOR RGB(248, 254, 79)

（3）.函数

<1>.创建游戏区域

struct GameBoard\* New\_GameBoard(HWND, int, int, int, int);

<2>.删除游戏区域

static void GameBoard\_Delete(struct GameBoard\* gameBoard)

{

DeleteObject(gameBoard->m\_drawer);

DeleteObject(gameBoard->m\_bitmap);

DestroyWindow(gameBoard->m\_handle);

free(gameBoard);

}

<3>.绘制游戏区域

static void GameBoard\_OnPaint(struct GameBoard\* gameBoard)

{

PAINTSTRUCT ps = {0};

BeginPaint(gameBoard->m\_handle, &ps);//BeginPaint函数为指定窗口 进行绘图工作的准备，并用将和绘图有关的信息填充到一个PAINTSTRUCT结构中

BitBlt(ps.hdc, 0, 0, GAMEBOARD\_WIDTH, GAMEBOARD\_HEIGHT, gameBoard->m\_drawer, 0, 0, SRCCOPY);//该函数对指定的源设备环境区域中的像素进行位块（bit\_block）转换，以传送到目标设备环境。

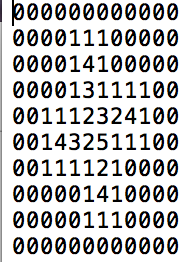
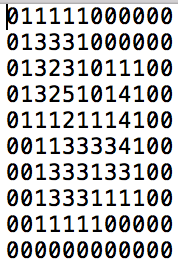
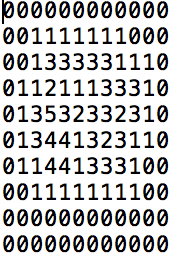
EndPaint(gameBoard->m\_handle, &ps);

}

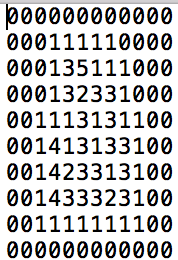
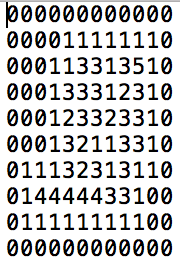
1. 关卡

（1）.关卡初始化。

游戏一共设置5个关卡，通过txt格式保存文件分别记录每一关，由其初始化游戏每一关。

<1> <2> <3>

<4> <5>

注释：每一个不同的数字分别对应屏幕中的­围墙，箱子，空地，目的地，人物，游戏背景（如‘1’表示围墙，‘2’表示箱子，‘3’表示游戏空地‘4’表示目的地，‘5’表示人物，‘0’表示背景）。

1. 关键技术

【.结构体表示的类的数据间关系】

类型

横向坐标

纵向坐标

绘图者

自身图像向下移动的图像1

自身图像保存者

子类对象

BASE

Box

Man

BASE

GameBoard

MainWindoww我W

Game

BASE

窗口

后台缓冲位图

绘图者

用于继承属性的父类对象

位于当前元素对象下面

的元素对象

向下移动的图像2

当前显示的图像

子类对象

用于继承的父类对象

游戏区域

6张位图表示左、右、上

2张当前存储变换位图

主窗口

游戏区域

绘图者

游戏状态

小人

关卡数

未到达目的地的箱子数

主窗口

游戏区域

游戏

【核心算法与函数】

1. 移动箱子

响应键盘上下左右键的操作，根据键盘操作方向，计算箱子（人物）新位置的坐标，可以通过改变箱子（人物）的坐标来实现游戏中箱子（人物）移动。

<1>.当操作键盘的上键，箱子的纵坐标减小；

<2>.当操作键盘的下键，箱子的纵坐标增大；

<3>.当操作键盘的左键，箱子的横坐标减小；

<4>.当操作键盘的右键，箱子的横坐标增大。

enum MoveResult Box\_Move(struct BASE\* base, UINT Direction)

{

struct Box\* box = (struct Box\*)base->m\_child;

enum MoveResult moveResult = NO\_MOVE;

BOOL bLeftFromDest = FALSE;

int nDestX = base->m\_x, nDestY = base->m\_y; // 小人要移动到的位置的坐标

BITMAP bmpInfo = {0};

switch (Direction)

{

case VK\_LEFT:

nDestX--;

break;

case VK\_RIGHT:

nDestX++;

break;

case VK\_UP:

nDestY--;

break;

case VK\_DOWN:

nDestY++;

break;

}

    /\* 如果旁边是墙或箱子，则直接返回\*/

if (g\_bases[nDestY][nDestX]->m\_type <= BOX)

{

return moveResult;

}

moveResult = NORMAL\_MOVED;

    /\* 判断是否要离开目的地\*/

if (base->m\_type == BOX && box->m\_below->m\_type == DESTINATION)

{

bLeftFromDest = TRUE;

moveResult = LEFT\_FROM\_DEST;

box->m\_CurrentImage = base->m\_Down1;

}

/\* 用背后对象擦除自己\*/

box->m\_below->draw(box->m\_below);

g\_bases[base->m\_y][base->m\_x] = box->m\_below;

/\* 计算新坐标位置\*/

base->m\_x = nDestX;

base->m\_y = nDestY;

/\* 设置新的背后对象\*/

box->m\_below = g\_bases[base->m\_y][base->m\_x];

/\* 判断是否到达目的地\*/

if (base->m\_type == BOX && box->m\_below->m\_type == DESTINATION)

{

if (bLeftFromDest)

{

moveResult = NORMAL\_MOVED;

}

else

{

moveResult = MOVE\_TO\_DEST ;

}

box->m\_CurrentImage = box->m\_Down2;

}

/\* 重绘自己\*/

SelectObject(base->m\_HDC, box->m\_CurrentImage);

GetObject( box->m\_CurrentImage, (int)sizeof(BITMAP), &bmpInfo);

StretchBlt(base->m\_drawer, base->m\_x \* GRIDS\_SIZE,

base->m\_y \* GRIDS\_SIZE,GRIDS\_SIZE, GRIDS\_SIZE,

base->m\_HDC, 0, 0, bmpInfo.bmWidth, bmpInfo.bmHeight, SRCCOPY);

/\* 将自己保存到数组中\*/

g\_bases[base->m\_y][base->m\_x] = base;

return moveResult;

}

1. 移动人物

由移动箱子的算法派生，移动人物时先移动箱子，再移动人物

static enum MoveResult Man\_Move(struct Man\* man, UINT Direction)

1. 判断过关与通关

static void Game\_CheckWon(struct Game\* game)

<1>.判断当前关卡是否通过，推到目的地的箱子数等于所有箱子数。

<2>.判断所有关卡是否通过，通过的关卡数目等于总关卡数目。

1. 消息处理

<1>.与时钟响应

static void Game\_OnTimer(struct Game\* game)

{

game->m\_man->dancing(game->m\_man);

}

static void MainWindow\_OnTimer(struct MainWindow\* mainWindow)

{

mainWindow->m\_game->onTimer(mainWindow->m\_game);

}

<2>与上下左右键响应

static void MainWindow\_OnKeyDown(struct MainWindow\* mainWindow, UINT nKey)

{

if (nKey >= VK\_LEFT && nKey <= VK\_DOWN)

{

mainWindow->m\_game->onKeyPress(mainWindow->m\_game, nKey);

}

}

<3>窗口消息处理

static INT\_PTR CALLBACK MainWindow\_Proc(HWND window/\*窗口\*/,

                                    UINT nMessage/\*消息标识\*/,

                                    WPARAM wParam,/\*消息附加参数\*/

                                    LPARAM lParam/\*消息附加参数\*/)

{

struct MainWindow\* mainWindow = (struct MainWindow\*)GetWindowLongPtr(window, GWLP\_USERDATA);

WORD wID = LOWORD(wParam);

UNREFERENCED\_PARAMETER(lParam);

switch (nMessage)

{

case WM\_COMMAND:

switch (wID)

{

case IDC\_MENU\_REPLAY:

MainWindow\_OnMenuItemReplay(mainWindow);

break;

case  ID\_40003:

MainWindow\_OnMenuDelete(mainWindow);

break;

       case IDCANCEL:

MainWindow\_Delete(mainWindow);

break;

}

return TRUE;

case WM\_DESTROY:

PostQuitMessage(0);

return TRUE;

case WM\_TIMER:

MainWindow\_OnTimer(mainWindow);

return TRUE;

}

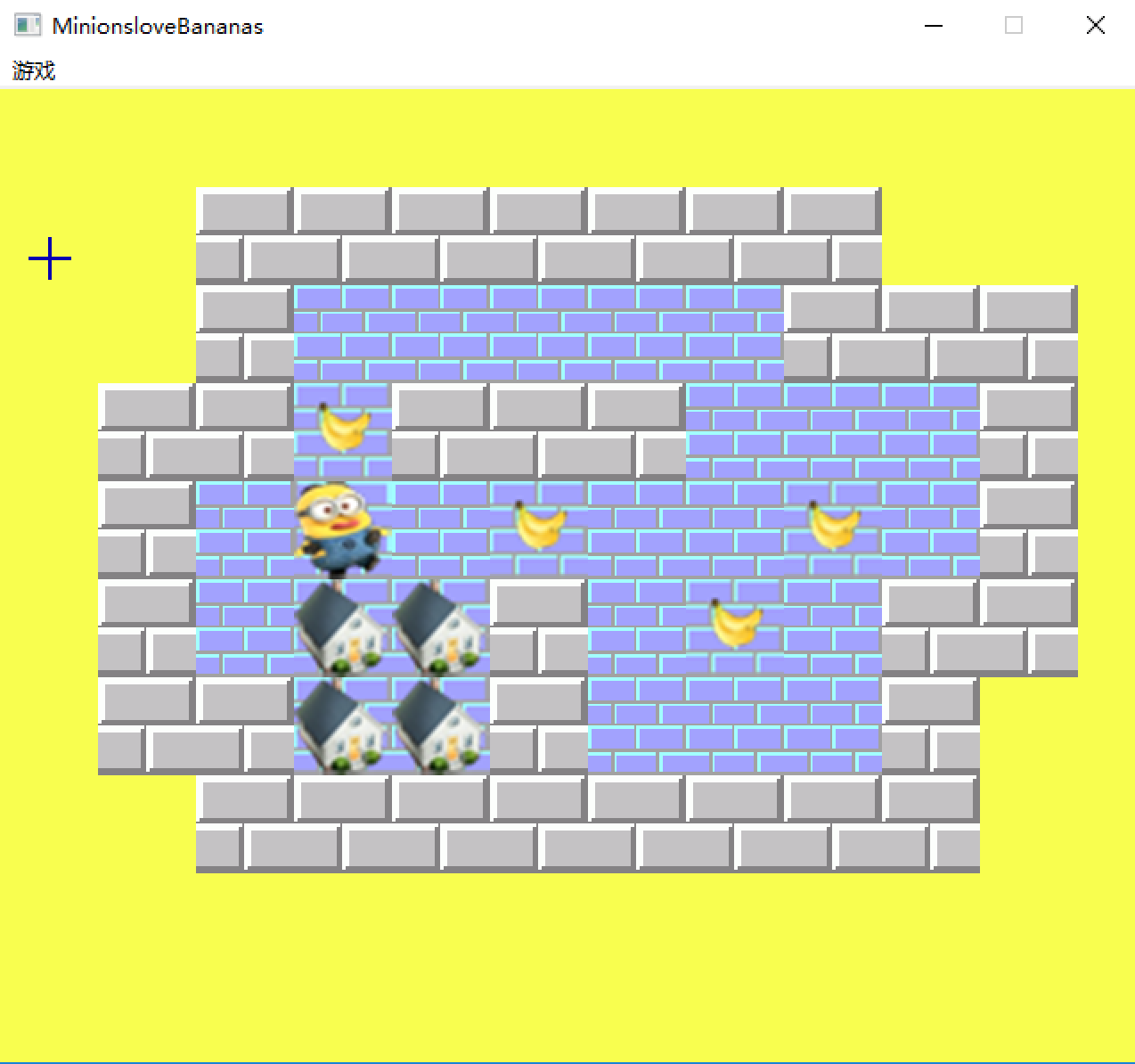
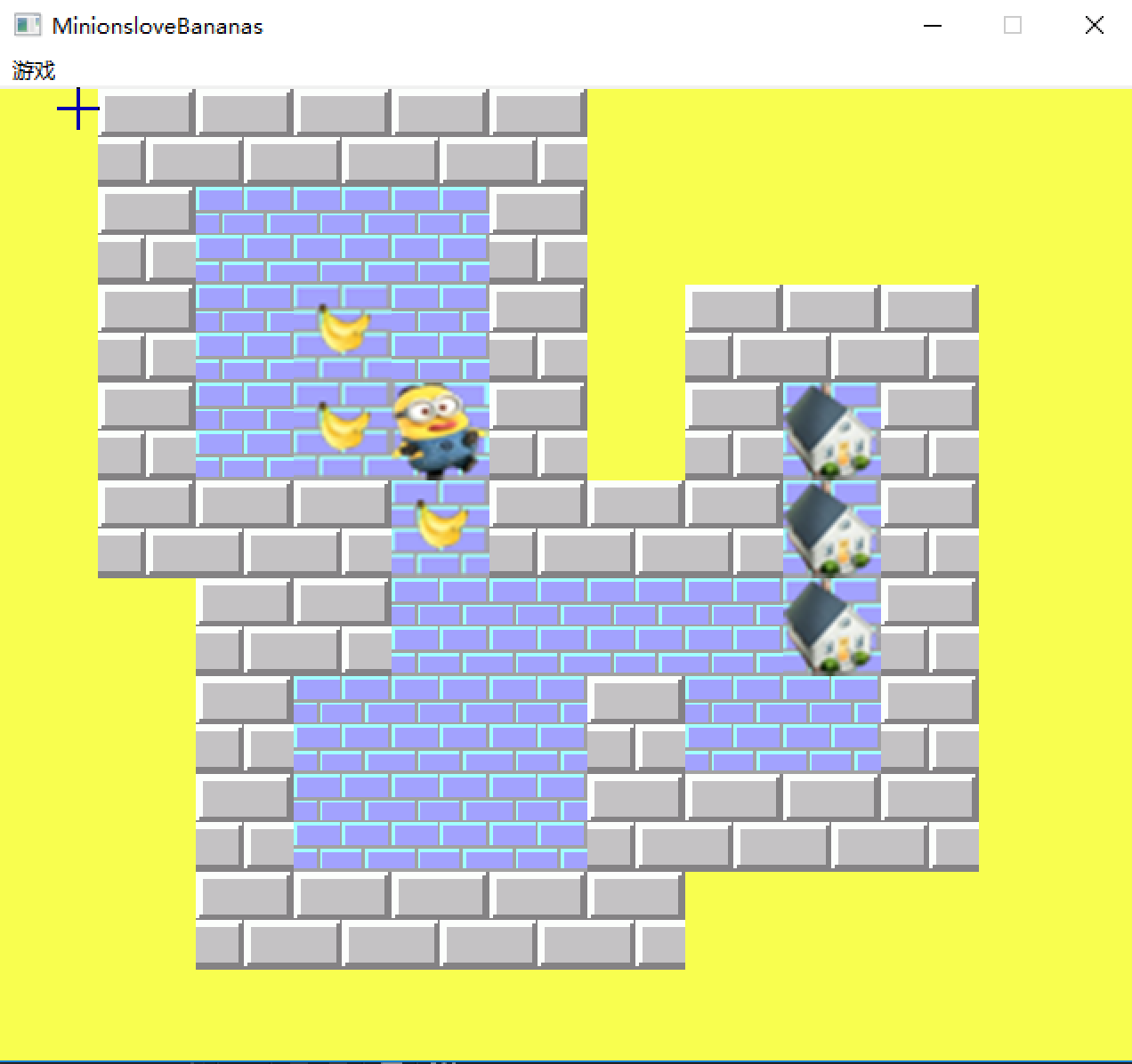
return FALSE;我认为最容易完成

}

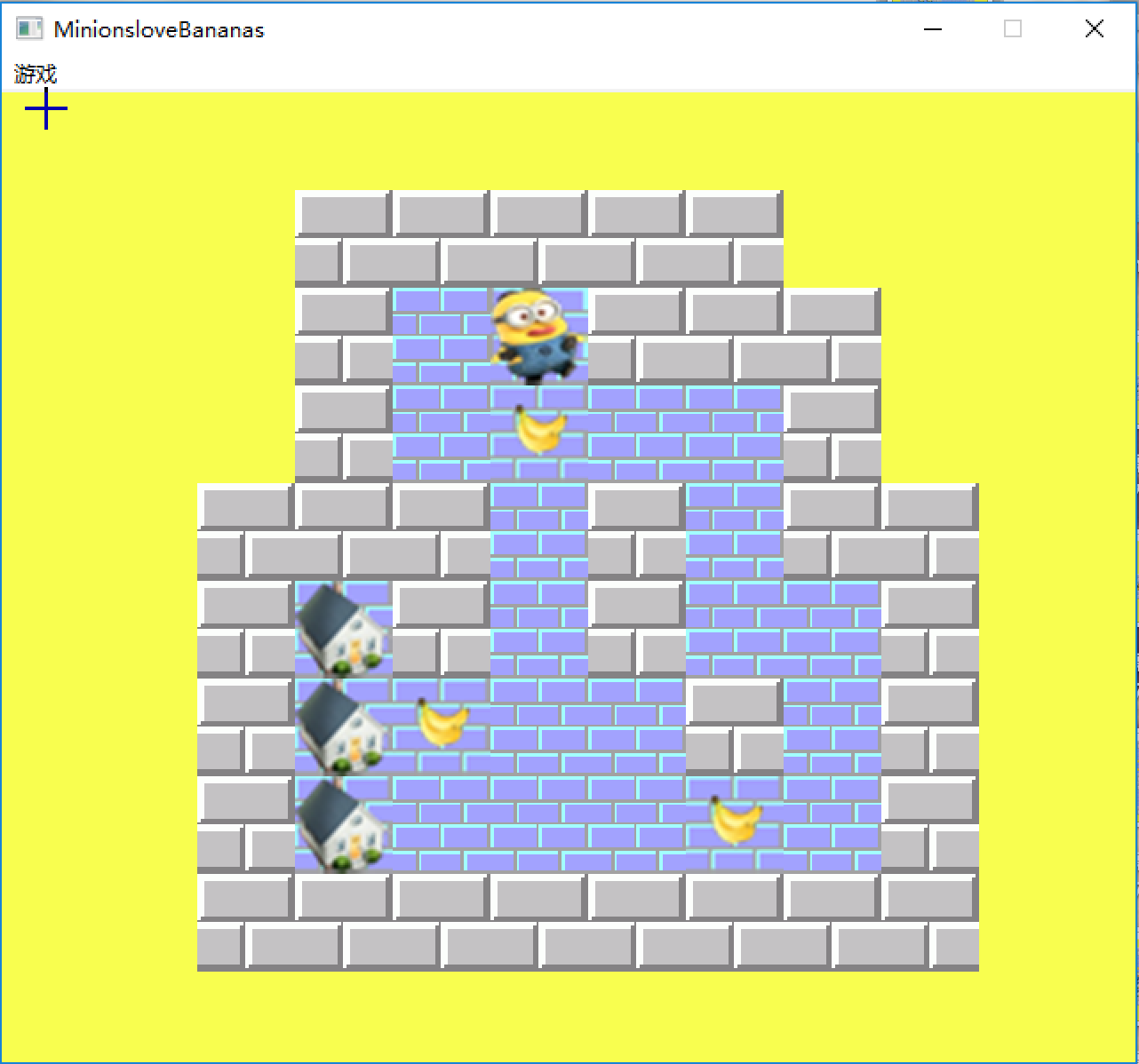
1. 结果和效果

完成了游戏的最初设计构想，通过上下左右键控制人物和箱子移动，并且实现游戏五个关卡的设计，实现了可以与用户交互的窗口菜单项“重玩本关”和“退出游戏”的功能，并且配有背景音乐，在移动过程中人物变换，让游戏更加有趣。

【.五个关卡】



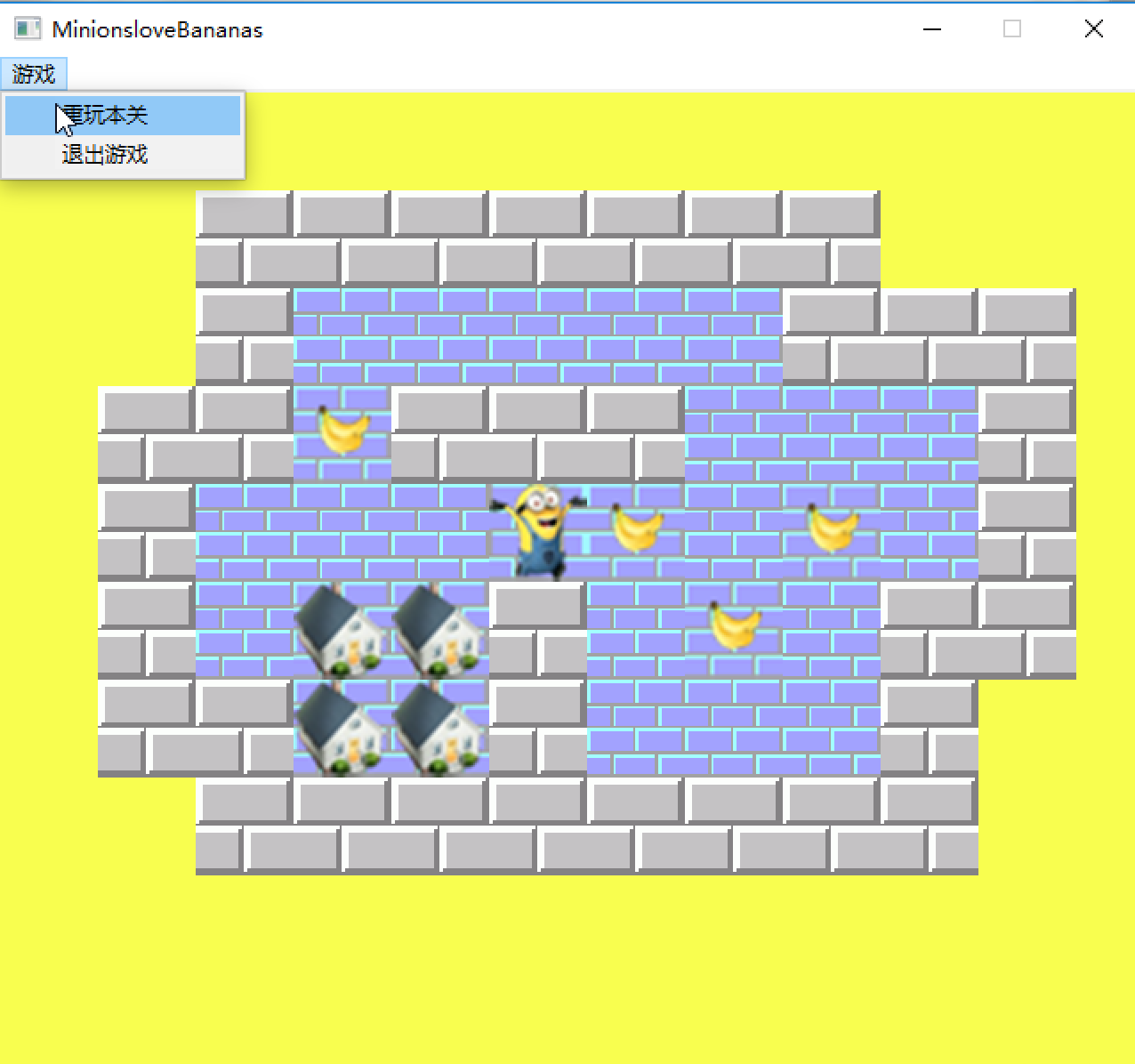
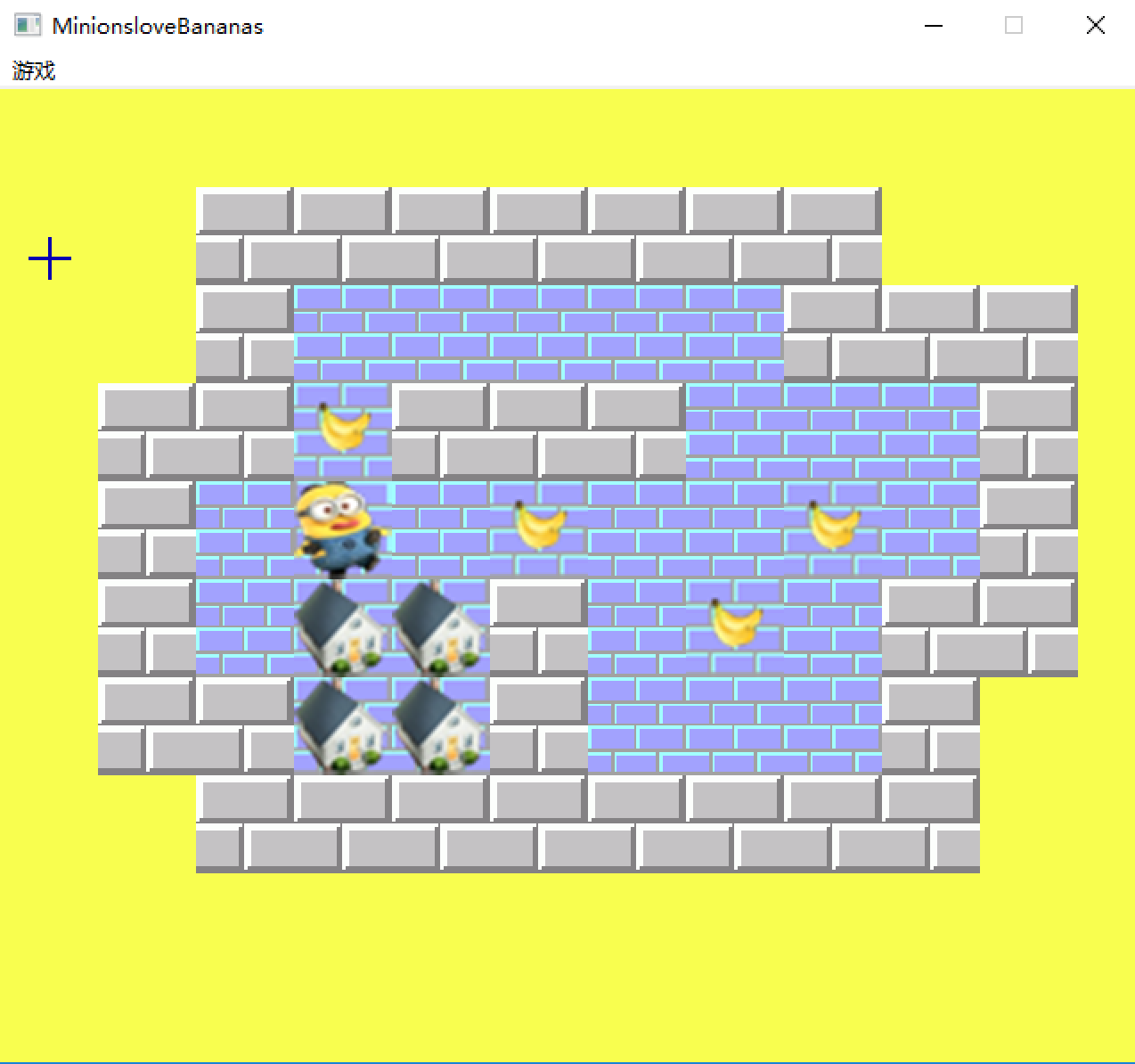
第一关 第二关 第三关



第四关 第五关

【.其他功能】

1. 重玩本关

1. 退出游戏



窗口关闭

过关

1. 总结和不足

很多事情，只有做了，才能知道到底难不难，开始课题时就选择了我认为最容易完成的《推箱子》小游戏的设计，开始觉得自己一个人很难完成，现在完成了的我仍然要说不容易，因为经历的很长的时间，从一开始学习别人的游戏代码，到想自己游戏的入手点，需要通过逻辑思维把游戏中的各种数据转化成计算机语言的数据，我需要确定整个游戏实现的数据结构，分析游戏过程实现的核心算法,学会API把游戏通过可视化窗口展现出来，并且可以与客户交互……很多东西需要自己去学习，很多问题需要去解决，不断的优化实现基础数据结构以及各个模块的函数实现，敲代码，改bug，又敲代码，再改bug……进入往复的循环，当一个个功能实现时，内心是无比的喜悦，难以言喻。做出了最基本的游戏，又希望可以有更多关卡，设置音效，美化显示界面……现在想起自己很多时候改bug改到吐血，头晕眼花，想要放弃，想到自己完全不理解某个问题，可现在都过来了，并且成功做出自己选择的游戏，学到了很多东西，但必须继续学习，因为还有很多需要改进的地方比如画面的处理，分辨率显示，游戏没有通关提示功能，游戏更多功能的设计等，这些不足需要在以后的学习实践中逐步弥补改进，要想获得更多，就要努力让自己一步步进步。