目录

[1. 内容提要 1](#_Toc28277970)

[2. 设计内容 2](#_Toc28277971)

[3. 本设计所使用的数据结构 3](#_Toc28277972)

[4. 功能模块详细设计 4](#_Toc28277973)

[4.1 头文件 4](#_Toc28277974)

[4.1.1 head.h 4](#_Toc28277975)

[4.1.2 define.h 4](#_Toc28277976)

[4.1.3 func.h 5](#_Toc28277977)

[4.1.4 data.h 5](#_Toc28277978)

[4.2 全局变量 6](#_Toc28277979)

[4.2.1 data.cpp 6](#_Toc28277980)

[4.3 绘制内容 7](#_Toc28277981)

[4.3.1 loadimage.cpp 7](#_Toc28277982)

[4.3.2 drawMap.cpp 8](#_Toc28277983)

[4.4 初始化 9](#_Toc28277984)

[4.4.1 initData.cpp 9](#_Toc28277985)

[4.4.2 initMap.cpp 10](#_Toc28277986)

[4.5 操作 14](#_Toc28277987)

[4.5.1 level.cpp 14](#_Toc28277988)

[4.5.2 move.cpp 15](#_Toc28277989)

[4.6 开始游戏 21](#_Toc28277990)

[4.6.1 startGame.cpp 21](#_Toc28277991)

[4.7 主函数 23](#_Toc28277992)

[4.7.1 main.cpp 23](#_Toc28277993)

[5. 收获与感想 24](#_Toc28277994)

# 1. 内容提要

本说明书介绍了本小组完成“推箱子”这一基于C语言和Easy\_X图形库制作的小游戏的主要思路以及相关的源代码。完成C语言游戏制作的目的是增强我们对C语言程序相关内容的熟练掌握，也为了加强同学们的动手实践能力与小组的合作能力，增加对使用程序解决实际问题的分析能力，以及加深C语言的基本知识内容的进一步理解。

# 2. 设计内容

本次设计计划设计一个游戏，能够通过基础C语言以及Easy\_X图形库实现包含图形界面的经典的推箱子游戏，游戏要实现经典的玩法，包含简洁的界面，以及简易的操作方法，并且包含多个关卡。

# 3. 本设计所使用的数据结构

使用如下例的二维数组保存地图数据：

const int map\_1[8][8] = {  
  {0, 0, 1, 1, 1, 0, 0, 0},  
  {0, 0, 1, 4, 1, 0, 0, 0},  
  {0, 0, 1, 0, 1, 1, 1, 1},  
  {1, 1, 1, 3, 0, 3, 4, 1},  
  {1, 4, 0, 3, 2, 1, 1, 1},  
  {1, 1, 1, 1, 3, 1, 0, 0},  
  {0, 0, 0, 1, 4, 1, 0, 0},  
  {0, 0, 0, 1, 1, 1, 0, 0}  
 };

其中0代表空白，1代表墙，2代表人，3代表箱子，4代表目的地，5代表已经完成的箱子。

# 4. 功能模块详细设计

## 4.1 头文件

### 4.1.1 head.h

#### 4.1.1.1 详细设计思想

引入整个程序所用到的标准头文件，以解决在其他文件中需要频繁引入这些头文件的问题。

#### 4.1.1.2 核心代码

#ifndef \_HEAD\_H\_  
 #define \_HEAD\_H\_  
 ​  
 #include <stdio.h>  
 #include <stdlib.h>  
 // ...  
 // ...  
 ​  
 #endif

### 4.1.2 define.h

#### 4.1.2.1 详细设计思想

单独保存宏定义，便于在多个文件内使用，解决宏定义需在其他文件中需要频繁引入的问题，同时需要更改时更加便于查找。

使用宏定义，增强代码的可读性。

#### 4.1.2.2 核心代码

#ifndef \_DEFINE\_H\_  
 #define \_DEFINE\_H\_  
 ​  
 #define BLANK     0 // 空  
 #define WALL       1 // 墙  
 // ...  
 // ...  
 ​  
 #endif

### 4.1.3 func.h

#### 4.1.3.1 详细设计思想

单独保存函数声明，便于在多个文件内使用，解决函数声明需在其他文件中需要频繁引入的问题，同时需要更改时更加便于查找。

将程序分解为多个函数构成的模块，减少代码的重复，便于模块化编程。

使用多文件编程技术，避免由于单一文件过于庞大而引起维护上的困难。

#### 4.1.3.2 核心代码

#ifndef \_FUNC\_H\_  
 #define \_FUNC\_H\_  
 ​  
 void startGame(void);        // 进入游戏  
 void initData(void);         // 初始化游戏数据  
 // ...  
 // ...  
 ​  
 #endif

### 4.1.4 data.h

#### 4.1.4.1 详细设计思想

单独保存全局变量声明，便于在多个文件内使用，解决全局变量声明需在其他文件中需要频繁引入的问题，同时需要更改时更加便于查找。

#### 4.1.4.2 核心代码

#ifndef \_DATA\_H\_  
 #define \_DATA\_H\_  
 ​  
 extern int\*\* recoveryMAP;  
 extern int\*\* restartMAP;  
 // ...  
 // ...  
 ​  
 #endif

## 4.2 全局变量

### 4.2.1 data.cpp

#### 4.2.1.1 详细设计思想

将程序中用到的全局变量保存在单独的文件中，便于查找和修改。

#### 4.2.1.2 核心代码

#include "head.h"  
 // ...  
 // ...  
 ​  
 int\*\* recoveryMAP = NULL; // 用于复原  
 int\*\* restartMAP = NULL;  // 用于重启  
 // ...  
 // ...  
 ​  
 IMAGE BkImg;  // 背景图片  
 IMAGE Blank;  // 空白  
 // ...  
 // ...  
 ​  
 const int map\_1[8][8] = {  
  {0, 0, 1, 1, 1, 0, 0, 0},  
  {0, 0, 1, 4, 1, 0, 0, 0},  
  {0, 0, 1, 0, 1, 1, 1, 1},  
  {1, 1, 1, 3, 0, 3, 4, 1},  
  {1, 4, 0, 3, 2, 1, 1, 1},  
  {1, 1, 1, 1, 3, 1, 0, 0},  
  {0, 0, 0, 1, 4, 1, 0, 0},  
  {0, 0, 0, 1, 1, 1, 0, 0}  
 };  
 // ...  
 // ...

## 4.3 绘制内容

### 4.3.1 loadimage.cpp

#### 4.3.1.1 详细设计思想

将加载图片资源的代码单独写出，便于查找和修改，简化主函数逻辑，便于设计整体结构。

#### 4.3.1.2 核心代码

void loadImage(void)  
 {  
  loadimage(&BkImg, L"res/BackGround.png", Width, Height);  // 背景  
  loadimage(&Blank, L"res/BLANK.png", BLOCK, BLOCK);   // 空  
  // ...  
  // ...  
 }

### 4.3.2 drawMap.cpp

#### 4.3.2.1 详细设计思想

通过二维数组中保存的内容输出游戏画面。

其中输出通过双重for循环控制。

由于putimage函数的特殊性，构造mapPicLoad函数来返回对应的图片引用，简化代码逻辑，提高可读性和可维护性。

为了防止图片闪烁，使用批量绘图技术来实现画面的连贯，解决了闪烁的问题。

为了防止上一个画面的残留影响新的画面，输出时优先再次输出背景，彻底覆盖之前的内容。

#### 4.3.2.2 核心代码

void drawMap(void)  
 {  
  BeginBatchDraw();  
  putimage(0, 0, &BkImg);  
  for (int i = 0; i < height; i++) {  
  for (int j = 0; j < width; j++) {  
  putimage(BLOCK \* (j + 2), BLOCK \* (i + 2), mapPicLoad(map[i][j]));  
  }  
  }  
  putimage(1179, 0, &Explain);  
  putimage(1179, 260, &Miku);  
  FlushBatchDraw();  
  EndBatchDraw();  
 }

IMAGE\* mapPicLoad(int id)

{

switch (id) {

case BLANK:

return &Blank;

break;

case WALL:

return &Wall;

break;

case HUMAN:

return &Human;

break;

case BOX:

return &Box;

break;

case DEST:

return &Dest;

break;

case DONE:

return &Box\_D;

break;

default:

return NULL;

}

}

## 4.4 初始化

### 4.4.1 initData.cpp

#### 4.4.1.1 详细设计思想

为了游戏的进行，正式开始游戏前需要对部分数据进行初始化，为了实现模块化，将其封装为一个函数。

由于数据的初始化基于完整的地图数据，首先调用initMap函数初始化地图数据。

通过遍历数组的方法找到人的位置和盒子的数目。

#### 4.4.1.2 核心代码

void initData(void)

{

initMap(level);

boxes = 0;

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

if (map[i][j] == 2) {

x = i;

y = j;

}

if (map[i][j] == 3) {

boxes++;

}

}

}

}

### 4.4.2 initMap.cpp

#### 4.4.2.1 详细设计思想

由于地图的初始化需要读入整个地图的数据，为了便于其他函数操作，这里采用了通过初始化的全局二维数组存储地图数据，通过拷贝进分配好内存的空指针来就进行操作。

由于内容很多，没有和initData.cpp写在一起，而是将其独立为一个文件。

由于部分函数仅在本操作中用到，故单独对其进行声明，不声明在func.h文件中。

由于释放内存，开辟空间这些操作需要反复使用，故封装为函数map\_malloc map\_free init。

#### 4.4.2.2 核心代码

#include "head.h"

// ...

void initMap\_1(void);

void initMap\_2(void);

// ...

// ...

void initMap(int level)

{

switch (level) {

case 1:

initMap\_1();

break;

case 2:

initMap\_2();

break;

// ...

// ...

default:

initMap\_NULL();

break;

}

}

int\*\* map\_malloc(int w, int h)

{

if (w <= 0 || h <= 0) {

return NULL;

}

int\*\* ret = NULL;

ret = (int\*\*)malloc(sizeof(int\*) \* h);

for (int i = 0; i < h; i++) {

ret[i] = (int\*)malloc(sizeof(int) \* w);

}

return ret;

}

void map\_free(int\*\* map)

{

for (int i = 0; i < height; i++) {

free(map[i]);

}

free(map);

}

void init(int h, int w)

{

map\_free(recoveryMAP);

map\_free(restartMAP);

map\_free(map);

recoveryMAP = map\_malloc(w, h);

restartMAP = map\_malloc(w, h);

map = map\_malloc(w, h);

height = h;

width = w;

}

void initMap\_1(void)

{

init(8, 8);

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

map[i][j] = map\_1[i][j];

restartMAP[i][j] = map\_1[i][j];

recoveryMAP[i][j] = recMap\_1[i][j];

}

}

}

void initMap\_2(void)

{

init(8, 8);

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

map[i][j] = map\_2[i][j];

restartMAP[i][j] = map\_2[i][j];

recoveryMAP[i][j] = recMap\_2[i][j];

}

}

}

// ...

// ...

## 4.5 操作

### 4.5.1 level.cpp

#### 4.5.1.1 详细设计思想

通过更改全局变量中当前关卡的值来更换关卡，更改变量之后重新加载游戏即可实现关卡的切换。

#### 4.5.1.2 核心代码

void nextLevel(void)

{

if (level < maxLevel) {

level++;

}

else {

level = 1;

}

startGame();

}

void prevLevel(void)

{

if (level > 1) {

level--;

}

else {

level = maxLevel;

}

startGame();

}

void restart(void)

{

startGame();

}

### 4.5.2 move.cpp

#### 4.5.2.1 详细设计思想

分析推箱子中存在的各种情况（前方为空白、箱子、墙、目的地等不同情况），分别进行处理。

对应上下左右移动这些情况，分别进行不同的处理。

首先计算次坐标，然后根据次坐标判断当前状态，根据返回状态判断进行何种操作。

考虑到移动后地图需要恢复，开辟一个新的数组，保存不同的数据，人移动走之后通过查看这个地图的数据来判断这里应该是空白还是目的地。

#### 4.5.2.2 核心代码

void moveUp(void)

{

tx = x - 1;

ty = y;

ttx = tx - 1;

tty = ty;

move(initStatus());

}

void moveDown(void)

{

tx = x + 1;

ty = y;

ttx = tx + 1;

tty = ty;

move(initStatus());

}

void moveLeft(void)

{

tx = x;

ty = y - 1;

ttx = tx;

tty = ty - 1;

move(initStatus());

}

void moveRight(void)

{

tx = x;

ty = y + 1;

ttx = tx;

tty = ty + 1;

move(initStatus());

}

void move(int stu)

{

switch (stu) {

case COULD\_NOT\_MOVE: // 不能移动

return;

break;

case COULD\_MOVE\_B: // 移动向空地

map[tx][ty] = 2;

mapRec(x, y);

x = tx;

y = ty;

break;

case COULD\_PULL\_B: // 推向空地

map[ttx][tty] = 3;

map[tx][ty] = 2;

mapRec(x, y);

x = tx;

y = ty;

break;

case COULD\_PULL\_D: // 推向目的地

map[ttx][tty] = 5;

map[tx][ty] = 2;

mapRec(x, y);

x = tx;

y = ty;

boxes--;

break;

case COULD\_PULL\_D\_B: // 推完成的箱子向空地

map[ttx][tty] = 3;

map[tx][ty] = 2;

mapRec(x, y);

x = tx;

y = ty;

boxes++;

break;

case COULD\_PULL\_D\_D: // 推完成的箱子向目的地

map[ttx][tty] = 5;

map[tx][ty] = 2;

mapRec(x, y);

x = tx;

y = ty;

break;

default: // 默认什么都不做（一般不会出现这种可能）

return;

}

}

int initStatus(void)

{

if (tx == EDGE || tx == EDGE\_R || ty == EDGE || ty == EDGE\_D) {

return COULD\_NOT\_MOVE;

}

else if (map[tx][ty] == BLANK) {

return COULD\_MOVE\_B;

}

else if (map[tx][ty] == WALL) {

return COULD\_NOT\_MOVE;

}

else if (map[tx][ty] == BOX) {

if (ttx == EDGE || ttx == EDGE\_R || tty == EDGE || tty == EDGE\_D) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == BLANK) {

return COULD\_PULL\_B;

}

else if (map[ttx][tty] == WALL) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == BOX) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == DEST) {

return COULD\_PULL\_D;

}

else if (map[ttx][tty] == DONE) {

return COULD\_NOT\_MOVE;

}

}

else if (map[tx][ty] == DEST) {

return COULD\_MOVE\_B;

}

else if (map[tx][ty] == DONE) {

if (ttx == EDGE || ttx == EDGE\_R || tty == EDGE || tty == EDGE\_D) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == BLANK) {

return COULD\_PULL\_D\_B;

}

else if (map[ttx][tty] == WALL) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == BOX) {

return COULD\_NOT\_MOVE;

}

else if (map[ttx][tty] == DEST) {

return COULD\_PULL\_D\_D;

}

else if (map[ttx][tty] == DONE) {

return COULD\_NOT\_MOVE;

}

}

}

void mapRec(int x, int y)

{

map[x][y] = recoveryMAP[x][y];

}

## 4.6 开始游戏

### 4.6.1 startGame.cpp

#### 4.6.1.1 详细设计思想

开始游戏前，首先初始化对应数据。

数据初始化结束后绘制画面。

之后进入循环，读取用户按键，并进行对应操作。

当未完成的箱子数为0，即胜利后跳出循环，进入下一关。

#### 4.6.1.2 核心代码

void startGame(void)

{

// 初始化数据

initData();

// 绘制地图

drawMap();

// 循环控制

do {

direction = \_getch();// 读取键盘输入

switch (direction) {

case 'w':

case 'W':

moveUp();

break;

case 'a':

case 'A':

moveLeft();

break;

case 's':

case 'S':

moveDown();

break;

case 'd':

case 'D':

moveRight();

break;

case 'r':

case 'R':

restart();

break;

case 'n':

case 'N':

nextLevel();

break;

case 'b':

case 'B':

prevLevel();

break;

default:

; // 输入错误的按键执行空语句

}

drawMap(); // 绘制地图

} while (boxes); // 当盒子数不为0时循环

// 跳出循环说明游戏结束，进行下一关

nextLevel();

}

## 4.7 主函数

### 4.7.1 main.cpp

#### 4.7.1.1 详细设计思想

主函数在完成初始化窗口，加载图片文件等准备工作后启动游戏。

#### 4.7.1.2 核心代码

int main()

{

// 初始化窗口

initgraph(Width, Height);

// 读取图片资源文件

loadImage();

// 进入游戏

startGame();

// 关闭绘图界面

closegraph();

// 结束游戏

return 0;

}

# 5. 收获与感想

在本次游戏设计中，除了学会了Easy\_X这一图形库，了解了用C语言不仅仅可以写出单调的终端界面，还可以写出美观的图形界面之外，更熟练的掌握了C语言的基础操作，加深了对C语言模块化编程的理解。

在项目设计中，项目基本实现了设想功能，实现了推箱子的基本功能，暂时没有在程序运行过程中发现bug，但是游戏界面略显单调，没有精美的图片和特效，内容略显老旧，缺乏创意。