推箱子游戏应用

——FIRMWARE 期末项目

目录

1. 项目概述

A. 项目目的

在 UEFI 环境下成功运行推箱子游戏应用

B. 开发环境

EDK2，windows 10

本次开发选用的 IDE 是 Sublime Text2，好处在于相当简洁而且对 UEFI 的语法同样有高亮处理。(C 语法部分)坏处在于无法调试(或者是因为自己不会用)，以至于自己开发的时候很多需要调试的地方都是用注释的方式。。。所以还是推荐大家用 VS 编译器进行相关的开发，具体如何用 VS 进行调试需要查找一下资料。

C. 基本需求

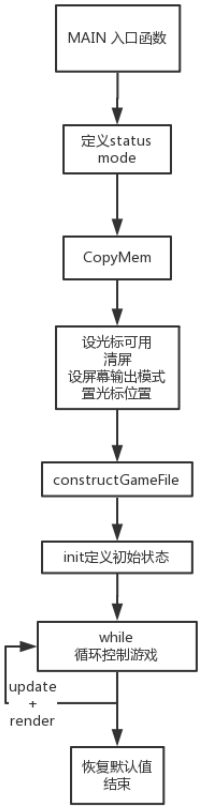
i. 推箱子游戏可以方向键控制人物、物品移动

ii. 一共有 1 ~ 35 个关卡选择

iii. 实现 GUI 界面像素图

2. 流程概述

A. 主流程图



B. 函数定义

i. void constructGameFile()

为我们的游戏资源文件添加到模拟器环境的文件系统中

ii. void showWelcome();

显示欢迎界面

iii. void showMenu()

显示菜单界面

iv. void showGame();

将游戏界面显示出来, 进入一个游戏界面时,只会调用一次

v. void showTips();

在游戏右侧显示一些说明

vi. void showSteps();

显示步数

vii. void showMap();

viii. void showWin();

显示通关的界面

ix. void showSelectLevel();

显示选关的界面

x. void loadMap();

读取游戏关卡地图文件信息到二维数组中

xi. void exitGame();

xii. void updateKey();

相应键盘按键事件, 设置游戏状态位

xiii. void render();

根据游戏的按键及相应的状态 进行下一步操作

xiv. void init();

初始化游戏状态位

xv. void clearScreen();

清屏

xvi. void selectLevel();

选关

xvii. void playGame();

操纵人物

xviii. void initMap();

得到人物的初始位置 和 箱子 目的地的 位置信息

xix. void basicShow(char *str) { gST->ConOut->OutputString(gST->ConOut, str); }

xx. void getFileName(int level, char* buf);

xxi. void drawMan();

绘制人物

xxii. void drawDestination();

绘制箱子目的地

xxiii. void drawBox();

绘制箱子

xxiv. void drawFloor();

xxv. void drawLine(int start_x, int start_y, int end_x, int end_y, int width);

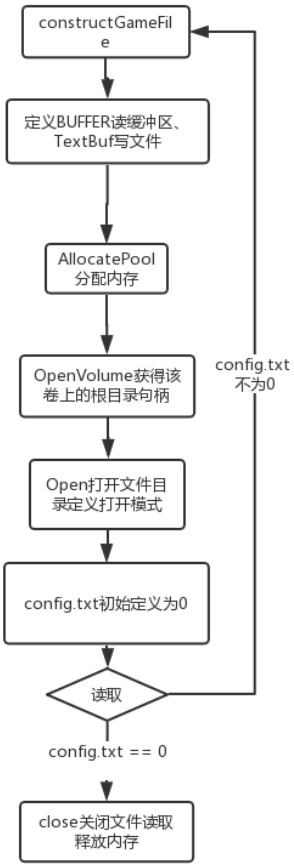
xxvi. void drawRectangle(int start_x, int start_y, int width, int length,EFI_GRAPHICS_OUTPUT_BLT_PIXEL* BltBuffer)

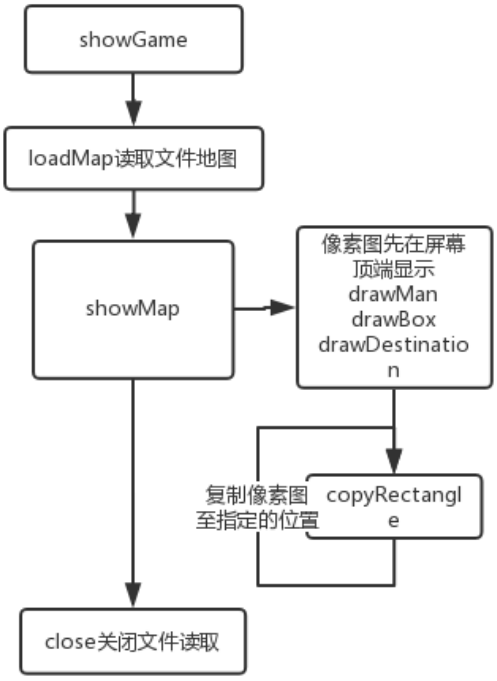
绘制单色矩形

xxvii. void copyRectangle(int s_x, int s_y,int e_x, int e_y,int width, int length)

拷贝屏幕区域(矩形)到指定的位置

其中两个重要子程序 constructGameFile & showMap 流程图如下





3. 代码详情

A. 变量基本定义

i. *Game_status*：分 8 种

ii. Directions：上下左右

iii. Node : 存放玩家位置信息

```
typedef enum _game_status{
    WELCOME, MENU, SELECT_LEVEL, GAMING, PAUSE, EXIT, BACK, WIN
}Game_Status;                       //游戏状态

typedef enum _directory {
    UP, DOWN, LEFT, RIGHT
}Directions;                         //人物的移动方向


typedef struct {
    int x;
    int y;
    char text[3];
}Node;
```

B. 控制台入口函数 main

参考书上第三章：使用 main 函数的应用程序工程模块

但实质上仍是 ShellCEntryLib 库

1. *定义 status/mode：EFI_STATUS、EFI_SIMPLE_TEXT_OUTPUT_MODE*
2. *其他服务 CopyMem：复制内存（参考书上第五章）*

   函数将 ConsoleMode 存到 SavedConsoleMode（SavedConsoleMode 是一个变量，保存了光标的位置和字符输出的颜色属性，用引用的方式传进去的）
3. *全局变量 gTS(指向 SystemTable)用 来完成设置光标可用、屏幕输出样式、设定光标位置*

   之后给每一步操作带一个 exception 的判定：

   ASSERT_EFI_ERROR(Status)
4. *初始化界面等完成后用 while 循环监测键盘事件的读取和游戏状态*
5. *游戏结束后恢复默认值*

```
int main
    (IN int Argc
    , IN char **Argv)
{
    EFI_STATUS Status;
    EFI_SIMPLE_TEXT_OUTPUT_MODE SavedConsoleMode;
     // Save the current console cursor position and attributes
    CopyMem(&SavedConsoleMode, gST->ConOut->Mode, sizeof(EFI_SIMPLE_TEXT_OUTPUT_MODE));
    Status = gST->ConOut->EnableCursor(gST->ConOut, FALSE); //设置光标是否可用
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->ClearScreen(gST->ConOut);    //清屏
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY,
    EFI_BLACK));//设置屏幕输出的样式
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->SetCursorPosition(gST->ConOut, 0, 0);   //设置光标位置
    ASSERT_EFI_ERROR(Status);


    constructGameFile();
    init();
    while(1){        //使用一个循环还控制游戏的进行
        updateKey(); //updateKey() 和 render()函数 将键盘事件的读取和游戏状态的设置 与 相应的逻辑处理
        进行了分离，使整个项目逻辑更清晰
        if (isRefresh == 1)
            render();
        if (game_status == EXIT)
            break;
    };

    gST->ConOut->EnableCursor(gST->ConOut, SavedConsoleMode.CursorVisible);
    gST->ConOut->SetCursorPosition(gST->ConOut, SavedConsoleMode.CursorColumn,
    SavedConsoleMode.CursorRow);
    gST->ConOut->SetAttribute(gST->ConOut, SavedConsoleMode.Attribute);
    gST->ConOut->ClearScreen(gST->ConOut);
    return 0;
}
```

C. 文件管理函数 constructGameFile（ ）

i. UEFI 内置操作 FAT 文件系统服 务：EFI_SIMPLE_FILE_SYSTEM_PROTOCOL（参考书上第七章）

1. *定义 BUFFER 读缓冲区、TextBuf 写文件*

```
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL *SimpleFileSystem; //文件系统句柄
EFI_FILE_PROTOCOL *Root; //文件系统根目录句柄
EFI_FILE_PROTOCOL *Dir;  //文件系统，游戏资源文件夹句柄
EFI_FILE_PROTOCOL *File; //游戏文件句柄
CHAR16 * TextBuf = (CHAR16*)L"CREATE SUCCESSFULLY!!!";
CHAR16 *BUFFER;
UINTN Buffer_Size = sizeof(CHAR16) * 65;
status = gBS->AllocatePool(EfiBootServicesCode, Buffer_Size, (VOID**)&BUFFER);
if (!BUFFER || EFI_ERROR(status)) {
    Print(L"ALLOCATE FAILURE");
}

UINTN W_BufSize = 0;
UINTN R_BufSize = 64;
status = gBS->LocateProtocol(&gEfiSimpleFileSystemProtocolGuid,
    NULL, (VOID**)&SimpleFileSystem);
```

2. *使用 OpenVolume 获得该卷上的根目录句柄*
3. *Open()打开文件目录，并定义打开模式 EFI_FILE_MODE_CREATE | EFI_FILE_MODE_READ | EFI_FILE_MODE_WRITE (文件不存在就 create 一个, 存在就以 读和写 的 方式打开*
4. *每次游戏的时候只执行一次这个函数, 如果是第一次使用这个程序的时候, 会 create 出 config.txt 这个文件, 这时候 其中的内容是空的, 会进入新创建游戏地图文件 1.txt ~ 35.txt 的循环体, 并在文件里面写入内容, 之后的游戏因为 config.txt 里不为空就不需要再次创建地图文件了*

```
status = SimpleFileSystem->OpenVolume(SimpleFileSystem, &Root);
status = Root->Open(Root,
    &Dir,
    (CHAR16*)L"Sokoban",
    EFI_FILE_MODE_CREATE | EFI_FILE_MODE_READ | EFI_FILE_MODE_WRITE,
    EFI_FILE_DIRECTORY);
status = Dir->Open(Dir,
    &File,
    (CHAR16*)L"config.txt",
    EFI_FILE_MODE_CREATE | EFI_FILE_MODE_READ | EFI_FILE_MODE_WRITE,
    0);

if (File && !EFI_ERROR(status))
{
    status = File->Read(File, &R_BufSize, BUFFER); //从config.txt文件中读信息，如果读取到的字节数为0(第一次游戏)，则创建游戏资源文件
    //Print(L"R_BufSize: %d\n\r", R_BufSize);      //第二次游戏及以后不需要重新创建资源文件
    if (R_BufSize == 0) {
```

5. *最后 close()关闭文件读取，并适当添加 exception 判断*

```
        else {
            //BUFFER[R_BufSize] = 0;
            //Print(L"config Read: %s\n\r", BUFFER);
            status = File->Close(File);
        }
    }
    else {
        File->Close(File);
        Print(L"config.txt not exist");
    }
    status = Dir->Close(Dir);
    status = Root->Close(Root);
    if (EFI_ERROR(status))
        Print(L"Close root Failure");
```

ii. 内存管理服务（AllocatePool\FreePool）

主要提供没存的分配与释放服务、管理系统内存映射

1. *分配内存*

读数据之前先按 BUFFER 大小分配

```
CHAR16 *BUFFER;
UINTN Buffer_Size = sizeof(CHAR16) * 65;
status = gBS->AllocatePool(EfiBootServicesCode, Buffer_Size, (VOID**)&BUFFER);
if (!BUFFER || EFI_ERROR(status)) {
    Print(L"ALLOCATE FAILURE");
}
```

2. *释放内存*

文件读取结束之后释放

```
  status = gBS->FreePool(BUFFER);
  if (EFI_ERROR(status))
      Print(L"Free error");
```

D. 输入输出处理 updateKey

i. 实现对键盘输入检测（gST->ConIn->ReadKeyStroke）

如果是在选关状态，则这个函数不处理键盘事件，改由选关函数专门处理

ii. *不同 game_*status 针对键盘输入调用不同的函数

并对三个状态值：game_status/ isRefresh（是否需要刷新界面）

```
void updateKey() {
    EFI_STATUS _status;
    EFI_INPUT_KEY key;
    if (!isSelectLevel) { //如果是在选关状态，则这个函数不处理键盘事件，改由选关函数专门处理
        _status = gST->ConIn->ReadKeyStroke(gST->ConIn, &key);
        ASSERT_EFI_ERROR(_status);
        if (EFI_ERROR(_status))
            return;
    }

    switch (game_status) {
    case WELCOME:
        if (key.ScanCode == SCAN_ESC) //退出游戏
            exitGame();
        else if (key.ScanCode == SCAN_F1) { //进入菜单界面
            showMenu();
            game_status = MENU;
        }
        break;
    case GAMING:
```

E. 图形界面展示

参考书上第 11 章

添加#include <Protocol/GraphicsOutput.h>

i. Protocol 使用类服务：LocateProtocol（找出系统中指定的 Protocol 的实例）

通过 Boot Service 的 LocateProtocol 获得 GraphicsOut 实例

```
EFI_GRAPHICS_OUTPUT_PROTOCOL *gGraphicsOutput;
EFI_STATUS _status;
_status = gBS->LocateProtocol(&gEfiGraphicsOutputProtocolGuid,
    NULL, (VOID**)&gGraphicsOutput);
if (EFI_ERROR(_status)) {
    Print(L"draw Fail!!!");
}
```

1. *—在 Teletype 模式下显示字符串*

   入口参数：AH =13H

2. *BH =页码*

   BL =属性(将颜色存放入其中)

   CX =显示字符串长度（所以字符串无需$结尾）

ii. Blt 传输图像

1. *定义颜色像素信息*

```
// 下面定义了一些颜色的像素信息
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL BLACK[1] = { 0,0,0,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL SKIN[1] = { 216,229,247,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL WHITE[1] = { 255,255,255,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL LIGHT_BROWN[1] = { 47,81,175,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL DARK_BROWN[1] = { 35,40,147,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL LIGHT_BLUE[1] = { 229,183,159,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL DARK_BLUE[1] = { 161,128,44,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL LIGHT_RED[1] = { 108,127,247,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL PINK[1] = { 201,181,255,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL RED[1] = { 0,0,255,0 };
static EFI_GRAPHICS_OUTPUT_BLT_PIXEL YELLOW[1] = {0,255,255,0 };
```

2. *用颜色填充对应的屏幕*

```
_status = gGraphicsOutput->Blt(gGraphicsOutput,
    BltBuffer,
    EfiBltVideoFill,
    0, 0,
    start_x, start_y,
    width, length,
    0);
```

3. *同理复制屏幕区域*

```
void copyRectangle(int s_x, int s_y,  //拷贝屏幕区域(矩形)到指定的位置
    int e_x, int e_y,int width, int length) {
    EFI_GRAPHICS_OUTPUT_PROTOCOL *gGraphicsOutput;
    EFI_STATUS _status;
    _status = gBS->LocateProtocol(&gEfiGraphicsOutputProtocolGuid,
        NULL, (VOID**)&gGraphicsOutput);
    _status = gGraphicsOutput->Blt(gGraphicsOutput,
        0,
        EfiBltVideoToVideo,
        s_x, s_y,
        e_x, e_y,
        width, length,
        0);
}
```

4. *绘制物品的矢量图*

   a. *将物体（HU、MAN_PIC、BOX_PIC、DESTINATION_PIC）以矢量图形式记录下来*

```
static UINT8 BOX_PIC[15][15] = {  //箱子的图像信息
    { 0,0,0,1,1,1,1,1,1,1,1,1,1,1,1 },
    { 0,0,1,2,2,2,2,2,2,2,2,2,2,2,1 },
    { 0,1,2,2,2,2,2,2,2,2,2,2,2,2,1 },
    { 1,2,2,2,2,2,2,2,2,2,2,2,2,2,1 },
    { 1,2,2,2,2,2,2,2,2,2,2,2,2,2,1 },
    { 1,2,2,2,2,2,2,2,2,2,2,2,2,2,1 },
    { 1,2,2,2,2,2,1,1,1,2,2,2,2,2,1 },
    { 1,1,1,1,1,1,1,3,1,1,1,1,1,1,1 },
    { 1,3,3,3,3,3,1,1,1,3,3,3,3,3,1 },
    { 1,3,3,3,3,3,3,3,3,3,3,3,3,3,1 },
    { 1,3,3,3,3,3,3,3,3,3,3,3,3,3,1 },
    { 1,3,3,3,3,3,3,3,3,3,3,3,3,3,1 },
    { 0,1,3,3,3,3,3,3,3,3,3,3,3,3,1 },
    { 0,0,1,3,3,3,3,3,3,3,3,3,3,1 },
    { 0,0,0,1,1,1,1,1,1,1,1,1,1,1,1 },
};
```

   b. *将信息存入代替 BltBuffer（显卡帧缓冲为 4 字节）的 COMMON 中再传入 Blt 函数作为参数*

   同理完成一下四个函数

   void drawMan(); //绘制人物

   void drawDestination(); //绘制箱子目的地

   void drawBox(); //绘制箱子

   void drawFloor();//绘制地板

```
//遍历表示图像的数组，一个像素一个像素的绘制图像
void drawBox() {
    EFI_GRAPHICS_OUTPUT_PROTOCOL *gGraphicsOutput;
    EFI_STATUS _status;
    _status = gBS->LocateProtocol(&gEfiGraphicsOutputProtocolGuid,
        NULL, (VOID**)&gGraphicsOutput);
    if (EFI_ERROR(_status)) {
        Print(L"draw Fail!!!");
    }
    EFI_GRAPHICS_OUTPUT_BLT_PIXEL COMMON[1] = { 0,0,0,0 };
    for (int i = 0; i < 15; i++) {
        for (int j = 0; j < 15; j++) {
            switch (BOX_PIC[i][j])
            {
            case 1:
                COMMON[0] = BLACK[0];
                break;
            case 2:
                COMMON[0] = RED[0];
                break;
            case 3:
                COMMON[0] = WHITE[0];
                break;
            }

            _status = gGraphicsOutput->Blt(gGraphicsOutput,
                COMMON,
                EfiBltVideoFill,
                0, 0,
                j + 97, i + 7,
                1, 1,
                0);

        }
    }
}
```

F. 游戏逻辑

i. 游戏之前先 showGame()

1. *loadMap()需要文件管理，同理通过 EFI_FILE_PROTOCOL、EFI_SIMPLE_FILE_SYSTEM_PROTOCOL 对文件进行读取*

   通过定义 CHAR8 R_BUF[500]存放读取的文件中的内容

```
while (R_SIZE == 64) { //循环读入游戏资源文件到 R_BUF ，一次最多读入64个字节
    status = File->Read(File, &R_SIZE, R_BUF + SUM);
    SUM += R_SIZE;
}

char c;
int _row = 0;
int _col = 0;
for (UINTN i = 0; i < SUM; i++) { //将 R_BUF中的信息映射到MAP中
    c = R_BUF[i];
    if (c != NULL) {
        if (c == '\n'){
            _row++;
            _col = 0;
        }
        else MAP[_row][_col++] = c;
    }
}
```

   初始的 map 通过'X 表示目的地'，'Q'表示箱子，'@'表示人物初始地

2. *showMap()*

   其中现将像素图在屏幕最顶端显示

之后通过与 loadMap 中读取的地图相对比，判断什么位置可以需要是用什么像素图，从而进行 copyRectangle()使得初始化更快捷

```
void showMap() { //在进入每一关游戏时只会调用一次，绘制游戏地图界面
    drawRectangle(1, 1, 28, 13,DARK_BROWN);   //画墙
    drawRectangle(0, 16, 14, 13,DARK_BROWN);  //画墙
    drawRectangle(17, 16, 14, 13,DARK_BROWN); //画墙
    drawDestination();//现将其初始化在界面顶部展示
    drawMan();
    drawBox();
    int i, j;
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            char tmp = MAP[i][j];
            if (tmp == '#') {
                copyRectangle(0, 0, 30 * j, 30 * i, 30, 30);
            }
            else if (tmp == 'X') {
                copyRectangle(61, 0, 30 * j, 30 * i, 30, 30);
            }
            else if (tmp == 'O' || tmp == 'Q' ) {
                copyRectangle(91, 0, 30 * j, 30 * i, 30, 30);
            }
            else if (tmp == '@') {
                copyRectangle(31, 0, 30 * j, 30 * i, 30, 30);
            }
        }
    }

}
```

3. *如果进入新的一关需要覆盖掉上一关的 map 使用 updateMap 函数否则会出现 map 重叠的 bug*

```
void updateMap(int x, int y, char c) { //更新游戏界面的地图

    switch (c)
    {
    case 'X':
        copyRectangle(61, 0, 30 * x, 30 * y, 29, 30);
        break;
    case 'Q':
    case 'O':
        copyRectangle(90, 0, 30 * x, 30 * y, 30, 30);
        break;
    case '@':
        copyRectangle(31, 0, 30 * x, 30 * y, 29, 30);
        break;
```

ii. 游戏进行 playGame()

1. *对人的移动做判断*

   一次只需最多更新 3 个位置（人、箱子会管理移动）

```
_x = MAN.x + direction.x;
_y = MAN.y + direction.y;
thing_front_man = MAP[_y][_x];
if (thing_front_man != '#') {  //如果人物的前面不是墙
    if (thing_front_man == ' ' || thing_front_man == 'X') { //如果人物的前面是空位置或者箱子的
    目的地
        MAN.y += direction.y;
        MAN.x += direction.x;
        MAP[MAN.y][MAN.x] = '@';
        MAP[old_pos.y][old_pos.x] = ' ';
        _move = 1;
    }
    else {  //如果人物的前面是箱子
        __x = _x + direction.x;
        __y = _y + direction.y;
        thing_front_box = MAP[__y][__x];
        if (thing_front_box == 'X' || thing_front_box == ' ') {//箱子的前面是空地或目的地
            MAP[__y][__x] = 'O';
            MAP[_y][_x] = '@';
            MAP[MAN.y][MAN.x] = ' ';
            MAN.x += direction.x;
            MAN.y += direction.y;
            _move = 1;

        }
    }
```

2. *进入判断是否游戏成功 filrate()*

   判断箱子是否都进入了目的地，是则 pass

```
int filrate() {   // "掩膜法" ，对箱子目的地的情况刷新一次
    int i;
    int j = 0;
    char c;
    for (i = 0;  i < boxNum; i++) {
        c = MAP[Y_Pos[i]][X_Pos[i]];
        if (c == ' ')    // 本来是箱子目的地但在playGame()函数的操作中被弄成了空位置，则将其重新设置成
        箱子目的地
            MAP[Y_Pos[i]][X_Pos[i]] = 'X';

        if (c == 'O'|| c== 'Q') { //本来是箱子目的地，而此时放的是箱子，则将该位置表示为箱子推到了目的
        地
            MAP[Y_Pos[i]][X_Pos[i]] = 'Q';
            j++;
        }

    }
    if (j == boxNum) // 如果，所有的箱子都到了目的地，则pass
        return 1;
    else return 0;
```

4. 编译运行

A. 在 AppPkg\Applications 文件夹下新建一个文件夹, 将 final.c 和 final.inf 添加进去

B. 在 AppPkg.dsc 的 [Components] 标签下添加 final.inf

C. 在 vs 命令行提示符里, 进入工程目录，输入以下指令:

   edksetup.bat

   build -p AppPkg\AppPkg.dsc

   即可完成对 final.c 的编译, 生成的 Final.efi 在 工程 目录的 build 文件夹 下找

D. 把上一步得到的 Final.efi (或使用给出的 Final.efi) copy 到 SecMain.exe 的目录下

E. 把 Socoban 这个文件夹 copy 到 SecMain.exe 的目录下

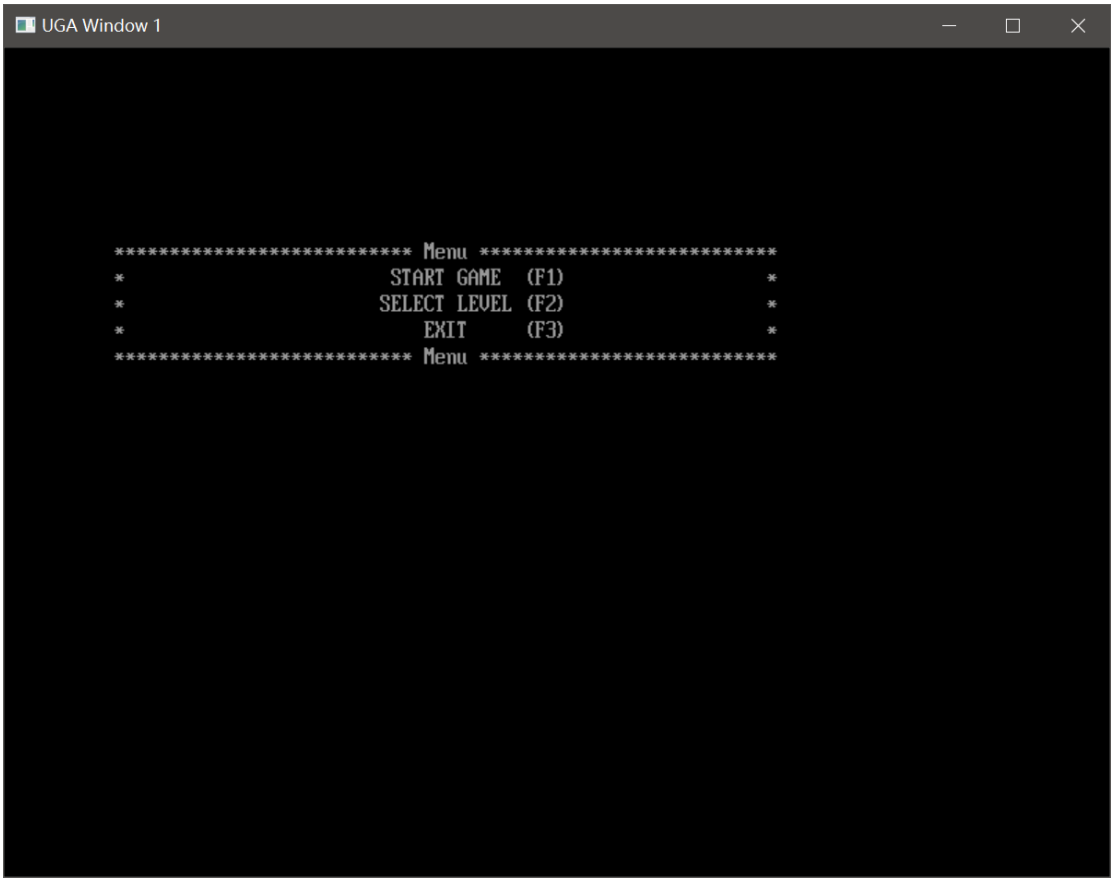
   这是用来存放关卡信息的

F. 运行 SecMain.exe, 输入 Final

G. 便可以运行成功

5. 使用流程

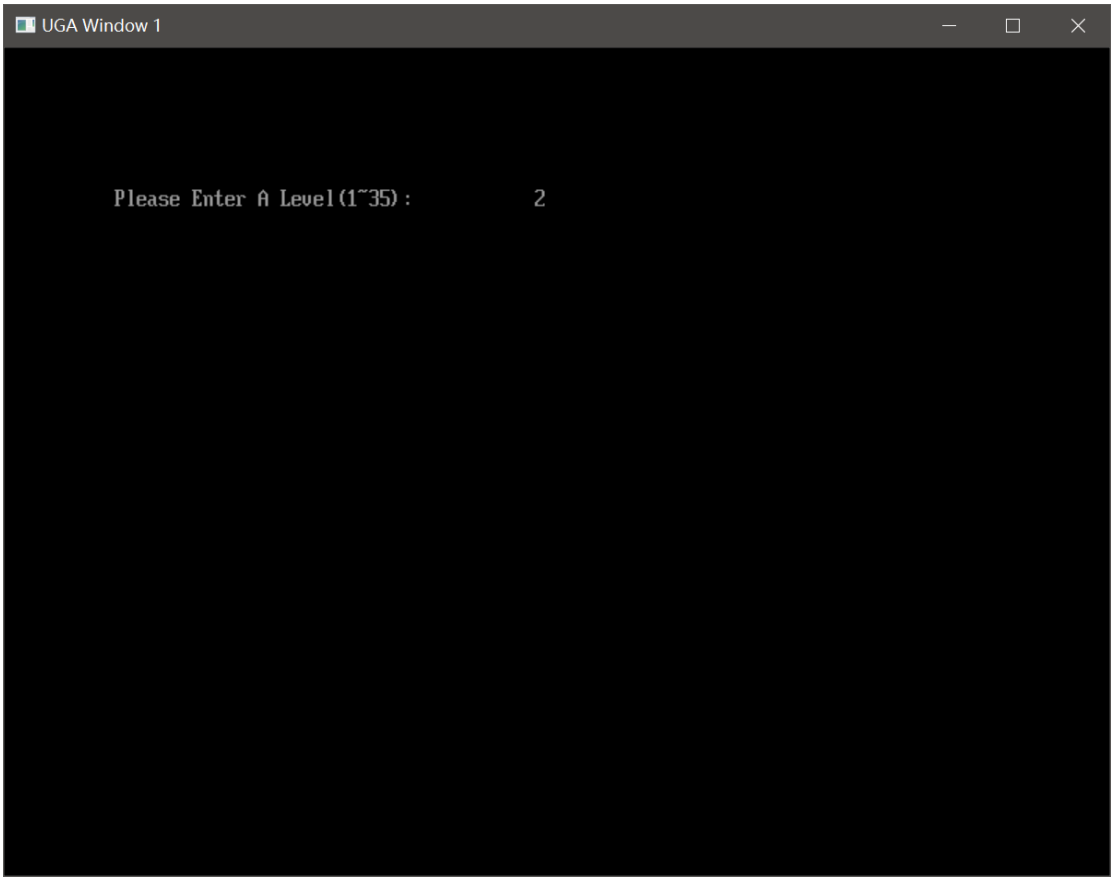
A. 进入开始界面



B. 输入 F1 进入游戏界面

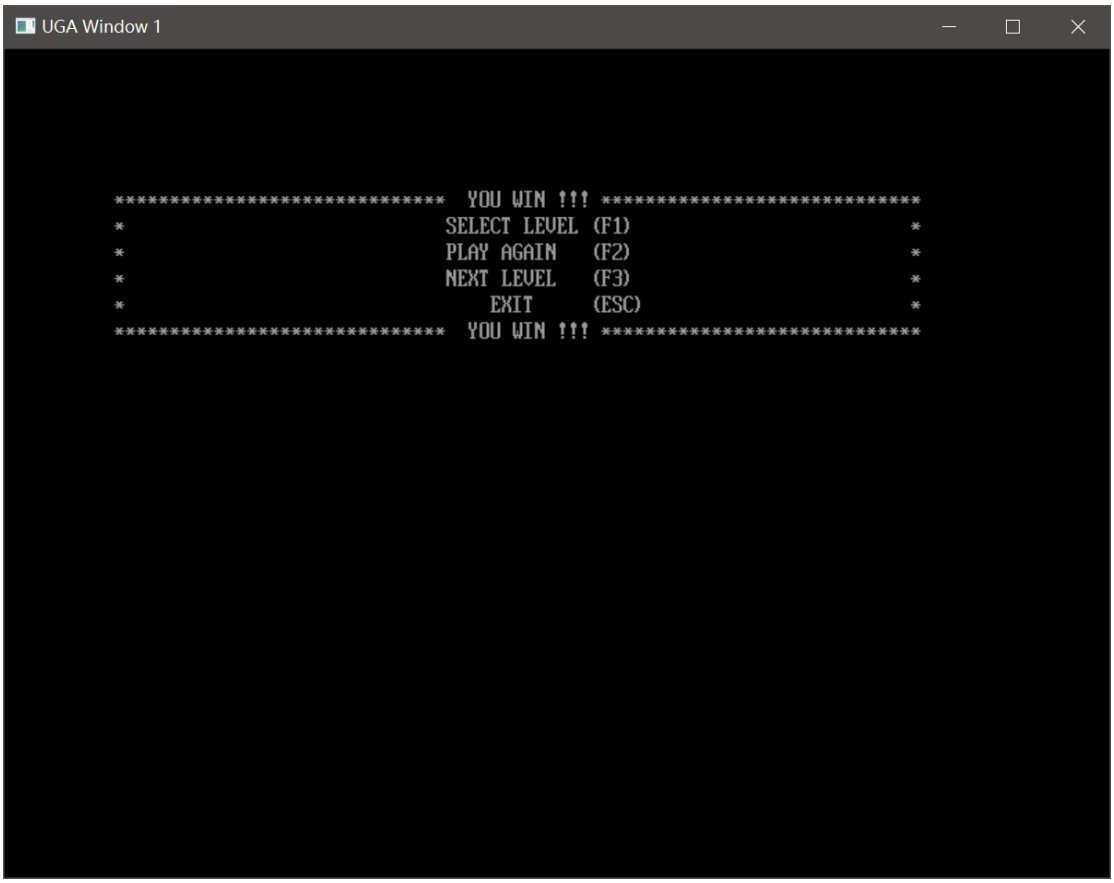
   可以玩箱子游戏（及将精灵球推进皮卡丘的洞里）

C. 在 menu 中输入 F2 则会进入选择关卡界面



输入你想去的关卡（1-35）页面将跳转

D. 游戏结束



6. 开发中遇到的一些问题

1. 图形界面的显示问题: 我们将图形定义在一个二维数组里面, 一个元素代表一个像素, 然后逐像素的绘制到界面上
2. 游戏地图的存储问题: 我们使用 txt 文件保存了游戏的地图, 使用该平台提供的 API, 将文件中的内容读取到一个一维数组中, 然后再经过筛选将其变换到一个二维数组中, 这个数组存放了我们游戏时的地图信息, 在开发中, 我们遇到了一个惊天的坑: 这个平台只能读取在自己文件系统里创建的文件, 并且不能读取在其他操作系统内创建的同名的文件. 比如我在该平台下创建了 1.txt 文件, 如果我用电脑上其他其他的 1.txt 替换它, 写没有问题, 但读不了...
3. 其他的一些游戏逻辑控制问题: 这个和平台无关就不展开了

7. 感想

1. 通过这次开发, 我们彼此结识了新的队友, 收获了很宝贵的友谊
2. “纸上得来终觉浅”, 在这次坎坎坷坷的开发之旅中, 虽然代码不太多, 但遇到的问题着实不少, 我们通过看书, 猜想与验证, 上网找资料终于解决了所有问题, 在这之后 原本很陌生的代码, API, 在现在看来是无比的亲切, 我们对 UEFI 产生了浓厚的兴趣
3. 感谢王老师的悉心培养，我们在课程中收获了很多，我们明白了计算机操作系统未被 load 的时候的样子，也学会了在这个状态下的编程知识。
4. 学弟 学妹们 一定要选这门课哟 ☺

8. 文件目录

A. .doc（项目文档）

B. final.c，final.dsc（源代码）

C. final.efi（执行程序）

项目已上传到 github 上：https://github.com/78541234h/UEFI_Final_Project

9. 参考资料

A. 《UEFI 原理与编程》

B. http://www.lab-z.com/

C. https://github.com/liute62/Firmware-UEFI-Maze-Game