Firmware 期末项目

迷宫游戏 Maze

liute62@sina.cn

目录

1 项目概述

本次项目的开发是基于UDK2014 （与课上讲的EDK ｜｜ 还是有所不同，至于UDK2010没用过不知道，下载地址 http://www.tianocore.org/），实际上实现的是一个UEFI的应用。一个迷宫类的游戏。用户通过控制人物走出地图获得胜利。地图的生成利用随机数以及图的深度优先算法。该游戏还有些不足的地方，保存地图和编辑地图并没有完成。

2 环境介绍

2.1 开发环境

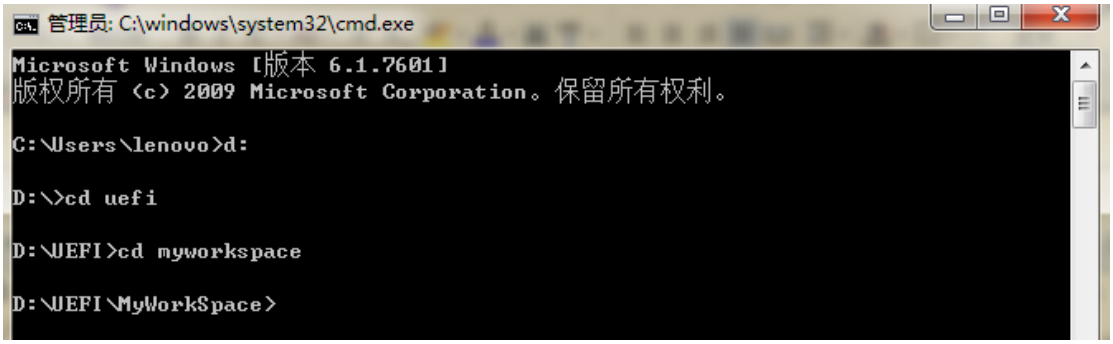
本次开发选用的IDE 是 Sublime Text2， 好处在于相当简洁而且对UEFI的语法同样有高亮处理。(C语法部分）坏处在于无法调试(或者是因为自己不会用)，以至于自己开发的时候很多需要调试的地方都是用注释的方式。。。所以还是推荐大家用VS编译器进行相关的开发，具体如何用VS进行调试需要查找一下资料。

2.2 编译过程

具体的编译工程如果之前写过UEFI 的Helloworld的话就很容易知道（尽管本次项目的入口函数与Helloworld的那次不同，所以导入的包不同，但无伤大雅）下述提供一个编译过程的参考:

Step1:

Win+R 打开cmd 的命令行，进入你的UDK2014的安装目录下(可以看到有几十个xxxxpkg的文件夹和一个edksetup.bat文件)



Step2:

输入如下命令进入NT32虚拟机的环境下

```
D:\UEFI\MyWorkSpace>edksetup.bat --nt32
```

Step3:

继续输入相关命令(build 命令 、-p 编译 、mypkg文件下的final.dsc 文件)

```
!!! WARNING !!! No CYGWIN_HOME set, gcc build may not be used !!!

D:\UEFI\MyWorkSpace>build -p mypkg/final.dsc
```

Step4:

实际上此时就开始编译了，如果没有语法错误的话，就会编译成功，生成.efi文件。如果有语法错误，可以通过命令行查看错误信息和错误行数是在哪。

```
[IA32]
Building ... d:\uefi\myworkspace\MyPkg\Application\final\final.inf [IA32]

- Done -
Build end time: 15:01:57, Jan.03 2015
Build total time: 00:00:13


D:\UEFI\MyWorkSpace>
```

几点说明:

设置final.dec (也可参考MdePkg的helloworld 的.dec，拷贝过来改个名字即可,我也不清楚有没有用..)

设置final.dsc （可以参考MdePkg的helloworld 的.dsc）

[BuildOptions] （编译选项，不把warning 当做 error）

*_*_*_CC_FLAGS = /WX-

关键是在Sample Applicaitons 里面加入链接的文件的地址(final.inf)

```
#### Sample Applications.
  MyPkg/Application/final/final.inf

################################################################
```

设置 final.inf （如下图，里面定义了编译后输出的名字Final，MODULE_TYPE 类型是UEFI 的应用，入口点是 ShellCEntryLib)

```
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = Final
  FILE_GUID                      = 2767252B-F4FA-40F6-A545-D82099859735
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 1.0
  ENTRY_POINT                    = ShellCEntryLib
```

```
[Sources]
  final3.c

[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.de
  ShellPkg/ShellPkg.dec

[LibraryClasses]
  ShellCEntryLib
  UefiApplicationEntryPoint
  UefiBootServicesTableLib
  UefiRuntimeServicesTableLib
  MemoryAllocationLib
  BaseLib
  BaseMemoryLib
  DebugLib
  PrintLib
  UefiLib
  HiiLib

[BuildOptions]
  *_*_*_CC_FLAGS = /WX-
```

编译的源文件是:
final3.c
用到的库的包是:
...
一些类库有:
....
BuildOptions
（很重要）
编译的选项，不记得是应该放在.inf文件还是.dsc文件里面了，我两个都放了。这个选项是让编译器不把warning 当作 error，程序员首选。

2.3 运行环境

UDK里面提供了一个模拟器nt32，利用这个我们就可以运行我们的UEFI的应用了。参考步骤如下:

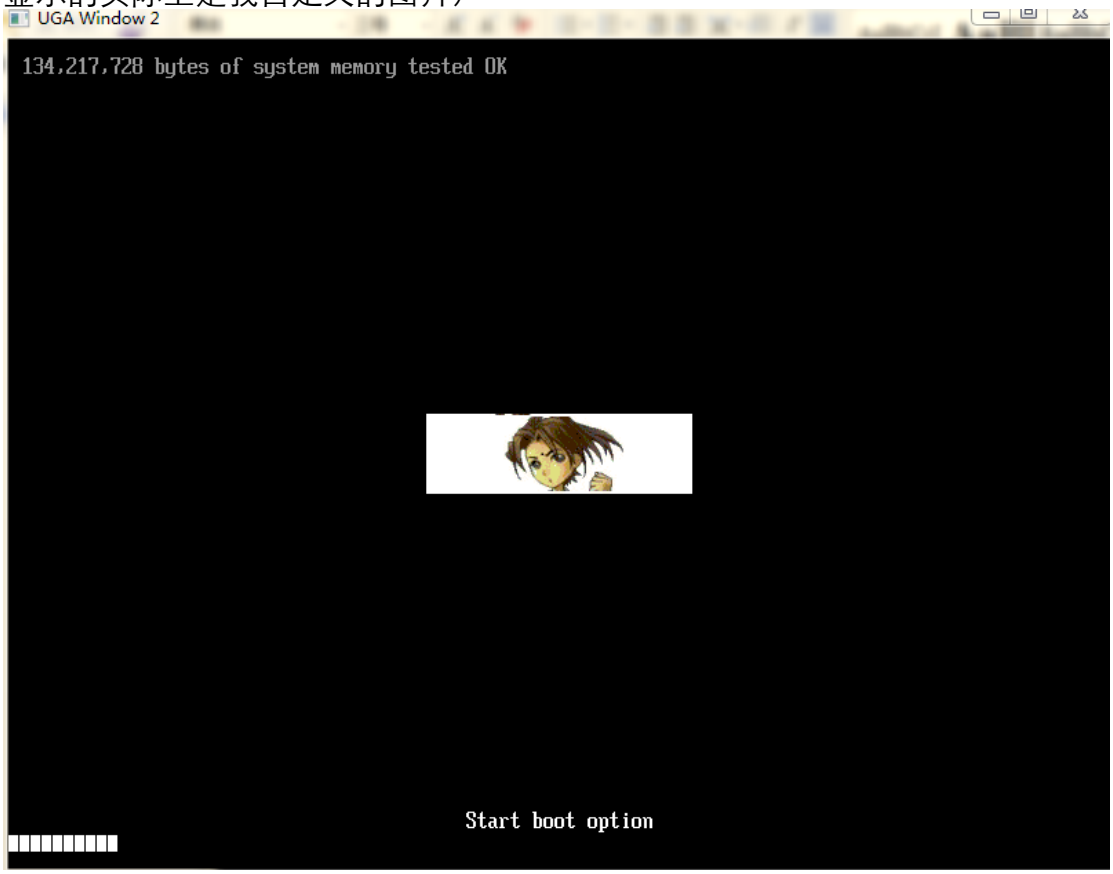
Step1:

记住要在edksetup.bat --nt32 之后再输build run

```
- Done -
Build end time: 15:01:57, Jan.03 2015
Build total time: 00:00:13


D:\UEFI\MyWorkSpace>build run
```

Step2:

进入efi shell 的环境下（因为我更改过开机的引导图片，所以boot加载的时候显示的实际上是我自定义的图片）



Step3:

进入fsnt0: 进入该盘（该盘的位置应该是依据之前配置tool_chain(VS)的位置定，实际上便是你生成出.efi文件的地方。可以用dir命令查看该目录下有什么文件。

```
EFI Shell version 2.40 [1.0]
Current running mode 1.1.2
Device mapping table
  fsnt0 :BlockDevice - Alias f8
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-BC
FA-0080C73C8881,00000000)
  fsnt1 :BlockDevice - Alias f9
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A935-A006-11D4-BC
FA-0080C73C8881,01000000)
  blk0  :BlockDevice - Alias (null)
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A928-A006-11D4-BC
FA-0080C73C8881,00000000)
  blk1  :BlockDevice - Alias (null)
        VenHw(58C518B1-76F3-11D4-BCEA-0080C73C8881)/VenHw(0C95A92F-A006-11D4-BC
FA-0080C73C8881,01000000)

Press ESC in 1 seconds to skip startup.nsh, any other key to continue.
Shell> fsnt0:_
```

dir命令：

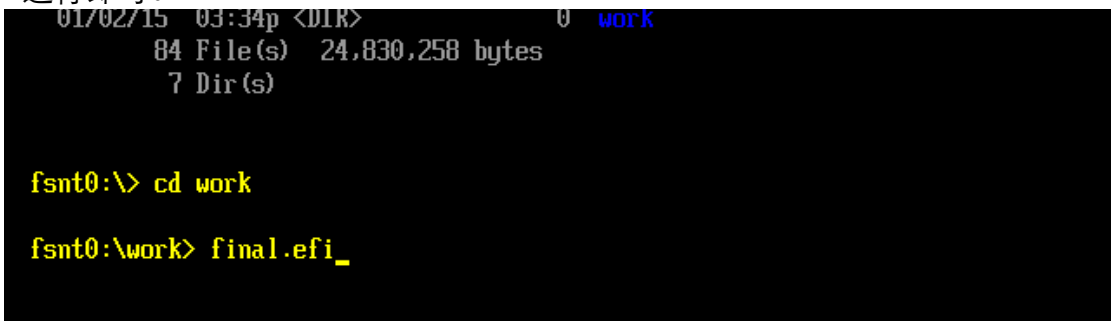
可以看到有该目录下有大量文件，通过这些文件就可以知道fsnt0对应你磁盘上的哪个位置（注：我在该目录下创建了一个work的文件夹，我运行的文件夹就放在work文件夹下）

```
Shell> fsnt0:

fsnt0:\> dir_
```

```
10/11/14  03:02p               3,093  TOOLS_DEF.IA32
10/11/14  03:06p              53,248  Udp4Dxe.efi
10/11/14  03:06p              53,248  UefiPxe4BcDxe.efi
10/11/14  03:06p              24,576  VariableInfo.efi
10/11/14  03:05p              45,056  VariableRuntimeDxe.efi
10/11/14  03:06p              45,056  VlanConfigDxe.efi
10/11/14  03:05p              24,576  WatchdogTimer.efi
10/11/14  03:04p              28,672  WinNtAutoScan.efi
10/11/14  03:05p              32,768  WinNtBlockIo.efi
10/11/14  03:05p              28,672  WinNtBusDriverDxe.efi
10/11/14  03:04p              28,672  WinNtFirmwareVolumePei.efi
10/11/14  03:04p              28,672  WinNtFlashMapPei.efi
10/11/14  03:05p              36,864  WinNtGopDxe.efi
10/11/14  03:05p              28,672  WinNtOemHookStatusCodeHandlerDxe.efi
10/11/14  03:04p              28,672  WinNtOemHookStatusCodeHandlerPei.efi
10/11/14  03:05p              32,768  WinNtSerialIoDxe.efi
10/11/14  03:05p              36,864  WinNtSimpleFileSystemDxe.efi
10/11/14  03:04p              24,576  WinNtThunkDxe.efi
10/11/14  03:04p              28,672  WinNtThunkPPIToProtocolPei.efi
01/02/15  03:34p <DIR>             0  work
      84 File(s)  24,830,258 bytes
       7 Dir(s)


fsnt0:\> _
```

Step4:

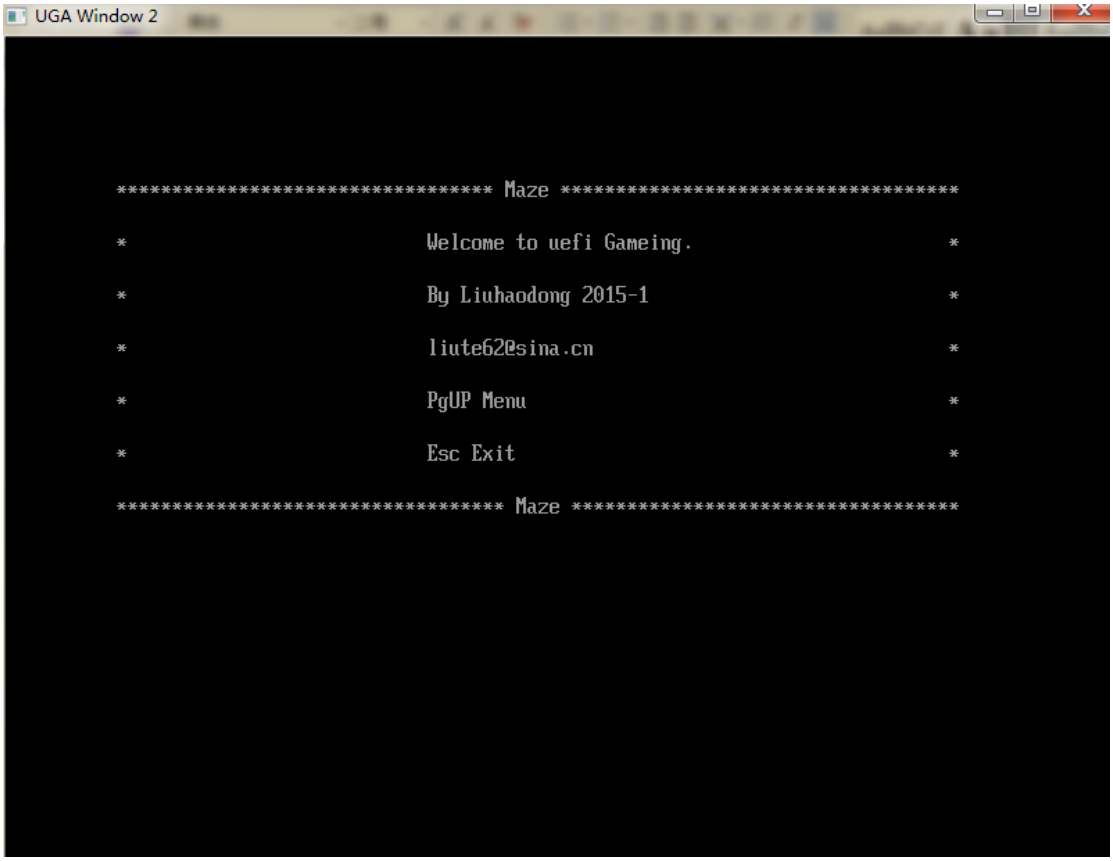
运行即可。



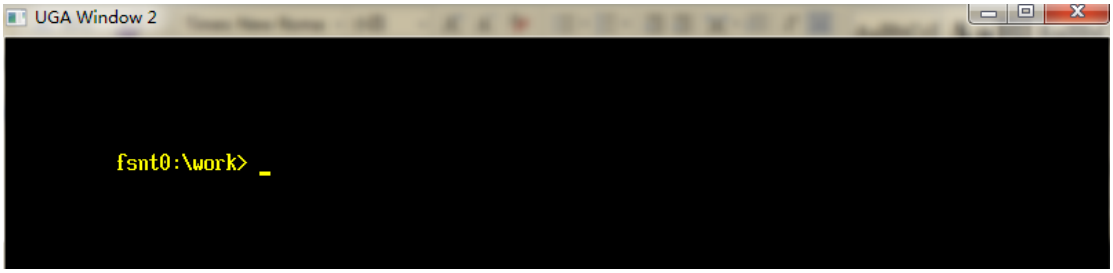
3 项目详情

3.1 运行实例

游戏界面如图所示，按**PgUp**键进入菜单，按**ESC**键退出菜单

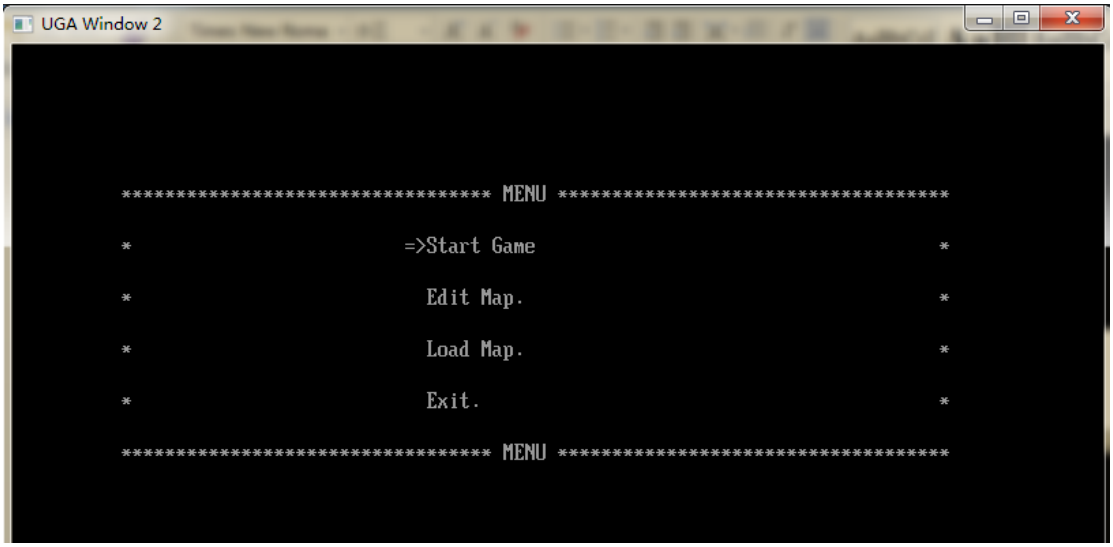


Esc:



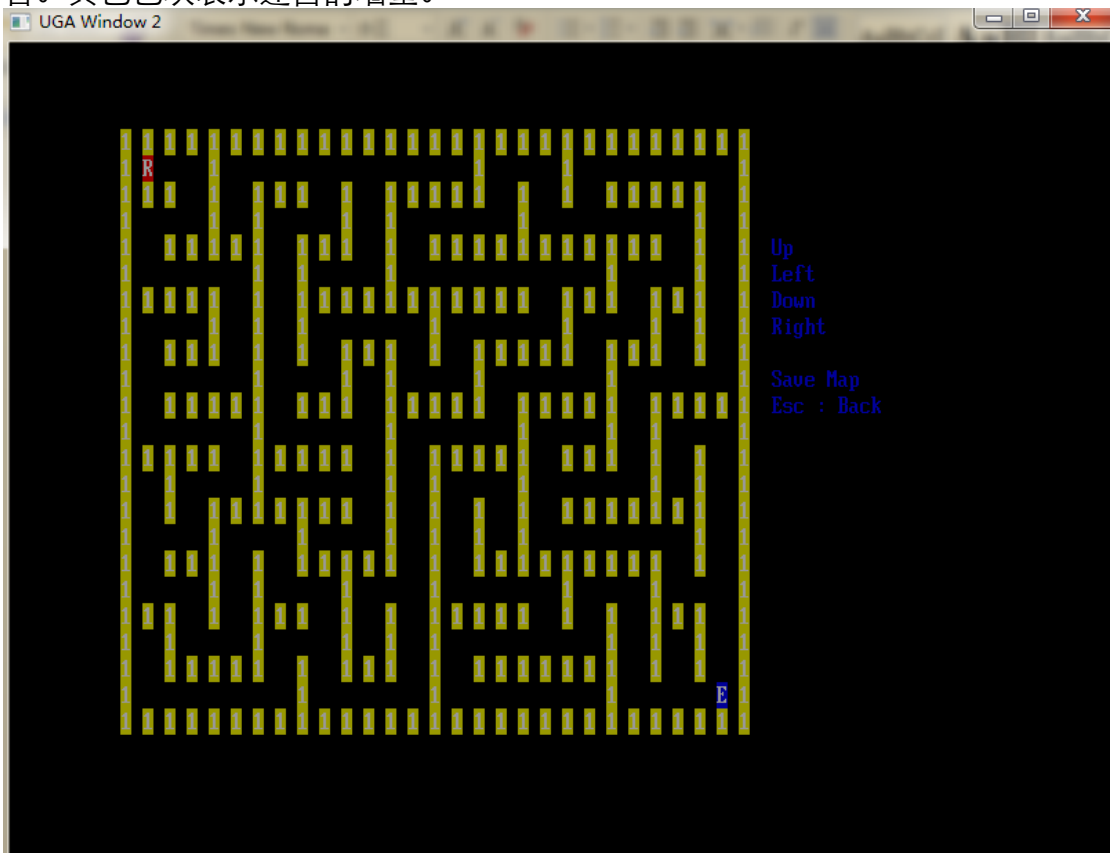
PgUp:

选择游戏内容，键盘上下键选择开始游戏还是退出，**PgUp**键确认

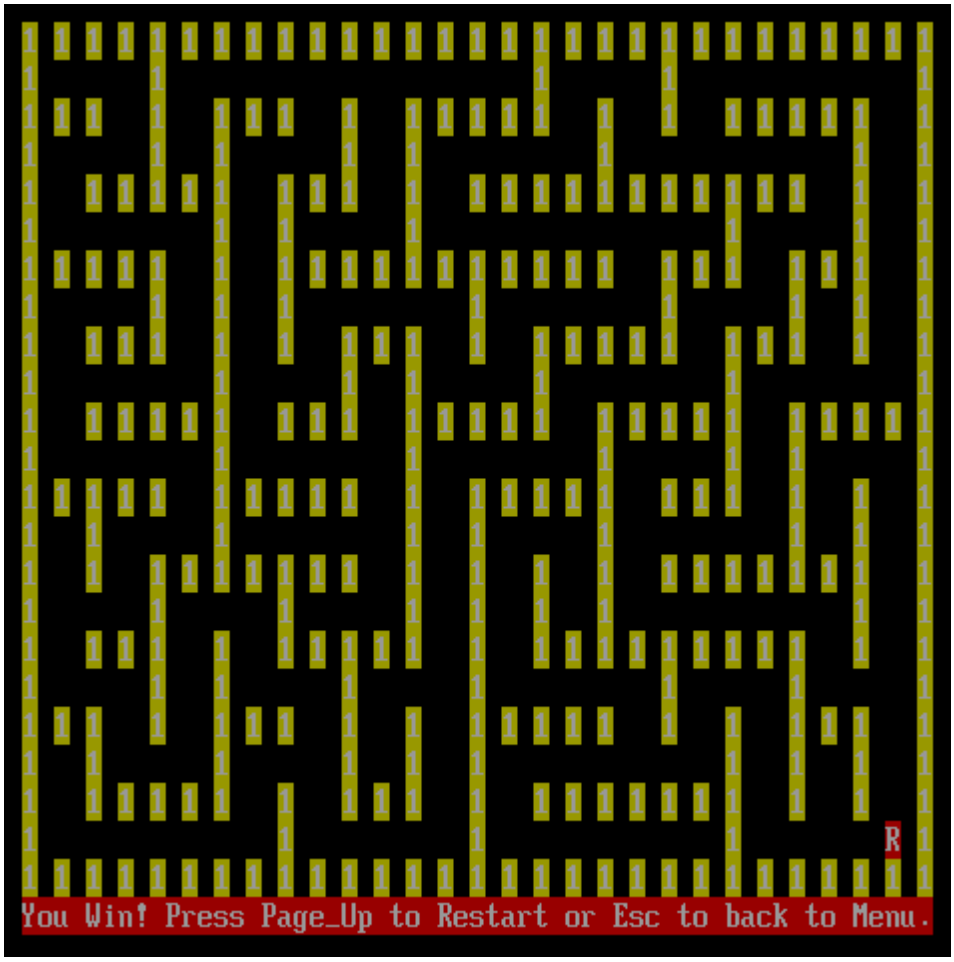


Start Game:

游戏界面如图，人物用红色背景的R字符代替，迷宫出口用蓝色背景的E字符代替。黄色色块表示迷宫的墙壁。



到达终点，显示You Win! 的字段。**PgUp**键重新开始，**Esc** 键返回菜单



3.2 代码分析

3.2.1 控制台入口参数

引用了ShellCEntryLib库，根据该库的定义

```
EFI_STATUS
EFIAPI
ShellAppMain (
    IN UINTN Argc,
    IN CHAR16 **Argv
)
EFI_STATUS
ParseCommandLine (
    IN UINTN Argc,
    IN CHAR16 **Argv
)
```

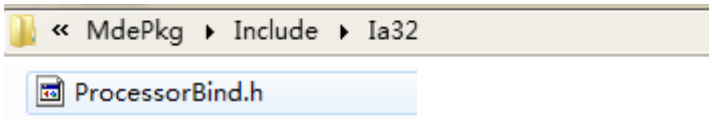
根据这样一个定义，我们就知道了可以解析控制参数的Argc和**Argv

便可以传入函数进行解析。

```
INTN ShellAppMain(UINTN Argc, CHAR16 **Argv )
{
```

3.2.2 变量的基本定义

在ProcessorBind.h里面实际定义了很多UEFI变量，可以参考。




```
typedef signed char           INT8;
#else
  ///
  /// 8-byte unsigned value.
  ///
  typedef unsigned long long  UINT64;
  ///
  /// 8-byte signed value.
  ///
  typedef long long           INT64;
  ///
  /// 4-byte unsigned value.
  ///
  typedef unsigned int        UINT32;
  ///
  /// 4-byte signed value.
  ///
  typedef int                 INT32;
  ///
  /// 2-byte unsigned value.
  ///
  typedef unsigned short      UINT16;
  ///
```

（比如C语言里面的int 型变量，UEFI里面为INT32型）。
同时，实际上C语言里面的结构体在UEFI里同样可以用。

```
typedef enum _STATUS{
    WELCOME, GAMING, PAUSE, WIN, STOP, MENU, EDITMAP, LOADMAP, SAVEMAP, GAME_EXIT
}STATUS;

typedef enum _DIRECTION{
    UP, LEFT, RIGHT, DOWN
}DIRECTION;

typedef struct _SHOWNODE{
    UINT32 x;
    UINT32 y;
    CHAR8 text[3];
}SHOWNODE;
```

3.2.3 输出

改变输出的颜色
gST->ConOut->SetAttribute(gst->ConOut,TextAttribute);
TextAttribute=TextColor|TextBackground
关于颜色的定义见
改变输出的内容
gST->ConOut->OutputString(gst->ConOut,L"字符串");
注意：实际上这里容易出现一个换行的问题，一般的C语言语法的 \n UEFI的输出中并没有那么好用，实际上需要 \n\r 来实现C语言语法的换行。
具体原因见：
http://www.lab-z.com/ 里面有一篇博文进行了说明。

```
/* SHOW WELCOME */
VOID showWelcome(VOID){

    gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_BLACK));
    gST->ConOut->OutputString(gST->ConOut, L"\n\n\r");
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*****************************
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*                           Welc
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*                           By Li
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*                           liute
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*                           PgUP
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*                           Esc E
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t*****************************
}
```

3.2.4 键盘输入

通过gST->ConIn->ReadKeyStroke(gST->ConIn,&Key);
获得ReadKey读取到键盘输入的code

```
EFI_STATUS Status;
EFI_INPUT_KEY Key;

Status = gST->ConIn->ReadKeyStroke(gST->ConIn, &Key);
ASSERT_EFI_ERROR(Status);
if (EFI_ERROR(Status))
    return;
```

、

```
if (Key.ScanCode == SCAN_ESC){
```

key.ScanCode 判断是哪一种输入
常用的有:
SCAN_ESC ESC 键
SCAN_PAGE_UP PgUp键
SCAN_PAGE_DOWN PgDn 键
SCAN_UP 上 键
SCAN_DOWN 下 键
SCAN_LEFT 左 键

SCAN_RIGHT　　右　　键

3.2.5 设置光标位置

其中x,y 对应为x,y 坐标位置。

```
gST->ConOut->SetCursorPosition(gST->ConOut, x, y);
```

3.2.6 产生随机数

通过调用UEFI的时间，来进行随机数的生成

```
static UINT8 RandomResult = 0;
```

```
gRT->GetTime(&Time, NULL);
RandomResult  = Time.Second;
```

调用RuntimeServices获得当前时间，在当前时间中取到当前时间的秒

3.2.7 清屏

通过gST->ConOut->ClearScreen(gST->ConOut)函数;

```
VOID myClear(VOID){
    gST->ConOut->SetAttribute(gST->ConOut,0x1);
    gST->ConOut->ClearScreen(gST->ConOut);
}
```

3.2.8 分配内存

因为引入了 Library/MemoryAllocationLib.h 的库
而该库提供了
VOID *
EFIAPI
AllocatePool (
    IN UINTN AllocationSize
);
进行内存的分配
A * a = AllocatePool(sizeof(A)); 其中A为某结构体等。

3.2.9 字符串拼接

因为没有用C的标准库，所以只能自己写一个函数还实现字符串的拼接（C标准库的Strcpy()函数）

```
CHAR8* myStrcpy(CHAR8 * strDest,const CHAR8 * strSrc){
    UINT32 i;
    CHAR8 *address = strDest;
    for(i = 0; strSrc[i] != '\0'; i++)
        strDest[i] = strSrc[i];
        strDest[i] = '\0';
    return address;
}
```

3.3 逻辑分析

3.3.1 ShellAppMain() 函数

调用init()函数实现游戏资源的初始化。
while(1) 无限循环，只有当游戏状态为GAME_EXIT才退出循环，并清空屏幕。
调用updateKeys()函数，实现键盘输入的监控
render()函数实现刷屏（根据参数控制显示在用户眼前的是哪个页面）

```
INTN ShellAppMain(UINTN Argc, CHAR16 **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);
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->ClearScreen(gST->ConOut);
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_BLACK));
    ASSERT_EFI_ERROR(Status);
    Status = gST->ConOut->SetCursorPosition(gST->ConOut, 0, 0);
    ASSERT_EFI_ERROR(Status);

    init();
    while(1){
        updateKeys(3);
        if(isRefresh){
            render();
        }
        if(sys_gs == GAME_EXIT){
            break;
        }
    }
    // Restore the cursor visibility, position, and attributes
    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;
}
```

3.3.2 updateKeys() 函数

实现对键盘输入检测。
首先判断当前的sys_gs(页面状态是怎么样子的）

如果是WELCOME 那么针对键盘输入调用不同的函数
同理如果是Menu...

```
VOID updateKeys(UINT32 ms){

    EFI_STATUS Status;
    EFI_INPUT_KEY Key;

    Status = gST->ConIn->ReadKeyStroke(gST->ConIn, &Key);
    ASSERT_EFI_ERROR(Status);
    if (EFI_ERROR(Status))
        return;

    switch (sys_gs)
    {
        case WELCOME:
            if (Key.ScanCode == SCAN_ESC){
                exitGame();

            }else if(Key.ScanCode == SCAN_PAGE_UP){
                startMenu(1);
            }
            break;
        case MENU:
            if (Key.ScanCode == SCAN_UP)
            {
                MenuIndex --;
                if(MenuIndex <= 0)
                    MenuIndex = MainMenuNum;
                isRefresh = 1;
            }else if(Key.ScanCode == SCAN_DOWN){
              MenuIndex ++;
              if(MenuIndex > MainMenuNum)
                    MenuIndex = 1;
              isRefresh = 1;
            }else if(Key.ScanCode == SCAN_PAGE_UP){
                switch(MenuIndex){
                 case 1:    //Start
                    startGame();
                    break;
                 case 2:    //Edit
                    //startEdit();
                    break;
                 case 3:    //Load
                    //startLoad();
                    break;
                 case 4:    //Exit
                    exitGame();
                    break;
```

3.3.3 startGame() 函数

设置游戏状态为GAMING 游戏中
清屏
初始化地图
显示一些操作提示
设置Cursor 的位置

```
VOID startGame(VOID){

    BuffIndex = 0;
    isRefresh = 1;
    sys_gs = GAMING;

    myClear();
    initMaze();
    showTips();

    totalSteps = 0;
    mySetCursorPos(0, ROW);
    gST->ConOut->OutputString(gST->ConOut, L"\t\t\t\t\t\t\t\t");

}
```

3.3.4 initMaze() 函数

初始化迷宫各个位置（是墙壁还是空地）
绘制迷宫各个位置
初始化人物位置
绘制人物位置

```
VOID initMaze(VOID){
    initBar(1);
    drawBars();

    initMan();
    showMan();
}
```

3.3.5 initBar() 函数

实际上迷宫的生成是利用了图的深度优先算法，结合随机数进行生成。算法的细节在这不在详述，可参考 《**随机迷宫生成算法浅析**》
http://hi.baidu.com/tunied/item/3f91b4cc0747fa16b77a24e4
需要注意的一点是在UEFI中貌似没有分配一块内存来支持递归的调用，所以如果算法用递归来写的话可能会报出Stack的错误。

```
UINT32 initBar(UINT32 flag){
    UINT32 i = 0, j = 0;
    if(IsEditMap){
        IsEditMap = 0;
        return 1;
    }

    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            BARS[i][j] = 0;
        }
    }
    return createMap(flag);
}
```

3.3.6 drawBars() 函数

实际上是通过自己定义的myStrcpy()函数（用于字符串的拼接）,把需要输出的内容拼接到一个结构体的Char型数组中，然后实现对多个结构体Char型数组的输出。

```
VOID drawBars(VOID){

    UINT32 i = 0, j = 0;
    SHOWNODE node;
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            node.x = j * 2;
            node.y = i;
            if(BARS[i][j])
                myStrcpy(node.text, BG);
            else
                myStrcpy(node.text, BAR);
            showInBuff(node);
        }
    }
    node.x = COL * 2 - 4;
    node.y = ROW - 2;
    myStrcpy(node.text, EXIT);
    showInBuff(node);
}
```

3.3.7 initMan() & showMan()函数

将人的位置存在一个结构体的数组中

```
VOID initMan(VOID){
    CurMan.x = 2;
    CurMan.y = 1;
    myStrcpy(CurMan.text, MAN);
}

VOID showMan(VOID){
    showInBuff(CurMan);
}
```

3.3.8 showGame() 函数

这个函数才实现了真正意义上的迷宫色块的输出，根据是迷宫地图还是人物还是出口来判断输出什么样的颜色。

```
VOID showGame(VOID){
    UINT32 i = 0;
    for(i=0; i<BuffIndex; i++){
        mySetCursorPos(showBuff[i].x, showBuff[i].y);
        if ((showBuff[i].text)[0] == '1')
        {
            /* code */
            gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_YELLOW));
        }if ((showBuff[i].text)[0] == ' ')
        {
            gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_BLACK));
        }if ((showBuff[i].text)[0] == 'E')
        {
            gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_BLUE));
        }if ((showBuff[i].text)[0] == 'R')
        {
            gST->ConOut->SetAttribute(gST->ConOut, EFI_TEXT_ATTR(EFI_LIGHTGRAY, EFI_RED));
        }
        gST->ConOut->OutputString(gST->ConOut,showBuff[i].text);
    }
```

3.3.9 render() 函数

根据游戏的状态判断当前该输出怎样的页面，是欢迎界面还是游戏菜单界面还是游戏界面等。

```
VOID render(VOID){
    isRefresh = 0;
    switch (sys_gs)
    {
        case WELCOME:
            showWelcome();
            mySetCursorPos(0, ROW+1);
            break;
        case MENU:
            showMenu();
            mySetCursorPos(0, ROW+1);
            break;
        case GAMING:
        case WIN:
            showGame();
            mySetCursorPos(0, ROW+1);
            break;
        case PAUSE:
            mySetCursorPos(0, ROW);
            gST->ConOut->OutputString(gST->ConOut,L"Pause! Press Page_Up to Continue.\t\t\t\t");
            mySetCursorPos(0, ROW+1);
            break;
        case STOP:
            mySetCursorPos(0, ROW);
            gST->ConOut->OutputString(gST->ConOut,L"You Win! Press Page_Up to Restart or Esc to back to Menu.\t\t");
            mySetCursorPos(0, ROW+1);
            break;
    }
}
```

4 参考、备注

参考资料:

1.http://www.lab-z.com/
2.UDK2014 的一些.h文件和官方文档。
3.https://github.com/
4. 基于UEFI技术的API性能分析设计与实现 -- 倪兴荣

备注:

实际上UEFI 的中文资料非常的少。包括很多论坛实际上也不能提供很多帮助，如bios人论坛。幸运的是找到了一些国外的网站，如www.lab-z.com和一些代码研究分析，将此项目实现。把编译的过程、代码的函数细节写的比较详细，希望能够帮助到将来一些学习UEFI的新手们。