کاربرد زبان++Cدر بازی سازی – قسمت اول

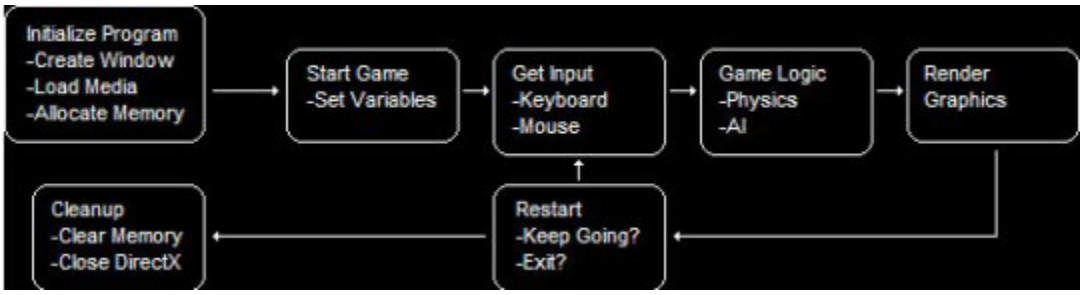
پژوهشگر و گرداورنده : امیر نوری

درس1: قسمت های تشکیل دهنده یک بازی

هفت قسمت از یک بازی!

یک بازی ویدِویی در اصل یک حلقه است که فرمان های پیوسته را انجام میدهد و هر بار اجرای حلقه یک فریم از بازی را ایجاد میکند.

خوب تصویر پایین مرحله های یک بازی را نشان میدهد.(از چپ به راست)



فاز یا مرحله اول : مقدار دهی اولیه برنامه

Initialize Program

اینجا زمان شروع کار است !. شما یک صفحه برای نمایش بازی خودتان میسازید و دایرکت ایکس را برای استفاده از گرافیک ها مدل ها و ... و اختصاص دادن حافظه در اینجا به اصطلاح راه اندازی می کنید.

فاز یا مرحله دوم : شروع بازی

این قسمت بازی شما را آمده شروع و بازی کردن میکند. اینجا قسمتی است که مثلاً شما مپ را انتخاب میکنید. مکان بازیکن ها را مشخص کرده و ترین بازی را میسازید و مقدار های رندم را میدهید(در بخشهای بعدی بیشتر با هر قسمت آشنا مشید).

فاز یا مرحله 3 : گرفتن ورودی از بازیکن

در اینجا شما تمام ورودی های موجود را از کیبورد و موس یا جویستیک میگیرید. این قسمت در مربوط کل به Dircet inputمیشود.

فاز یا مرحله 4 : اجرای منطق بازی مثل فیزیک یا هوش مصنوعی

در این بخش شما عملیات و کارهایی که در دنیای شما انجام میشود را پردازش میکند.چقدر مهمات برای بازیکن شما مونده ؟ آیا دشمنانش در حال آمدن یا رفتن هستند ؟ به چی برخورد میکند؟ و.. تمام این چیزها و خیلی چیزهای دیگه در این بخش تعیین میشود.

© Copyright 2010/ Animationdata.com & Partners

فاز یا بخش 5 : رندر گرافیک ها (تصویرها یا رسم ها نمیدونم ترجمه دقیق چیه همینطوری بهتره!)

اینجا قسمتی است که دایرکت ایکس بیشتر استفاده میشه. در اینجا شما تمام تصاویر(گرافیک ها)ی دو بعدی و سه بعدی را پردازش میکنید و آنهارا بر روی صفحه رندر میکنید.

فاز یا بخش 6 : Restart

برگشت به بخش 3 و انجام دوباره همه چیز...

فاز یا بخش 7 : CleanUp

این آخرین فرصت در برنامه شما برای پاک کردن همه چیز است. چون برنامه در پایان است.در اینجا تمام حافظه ها که برای کارتون اختصاص داده بودید آزاد میکنید.

درس2 : نگاهی کوتاه به ویندوز

hello worldکد پایین برنامه

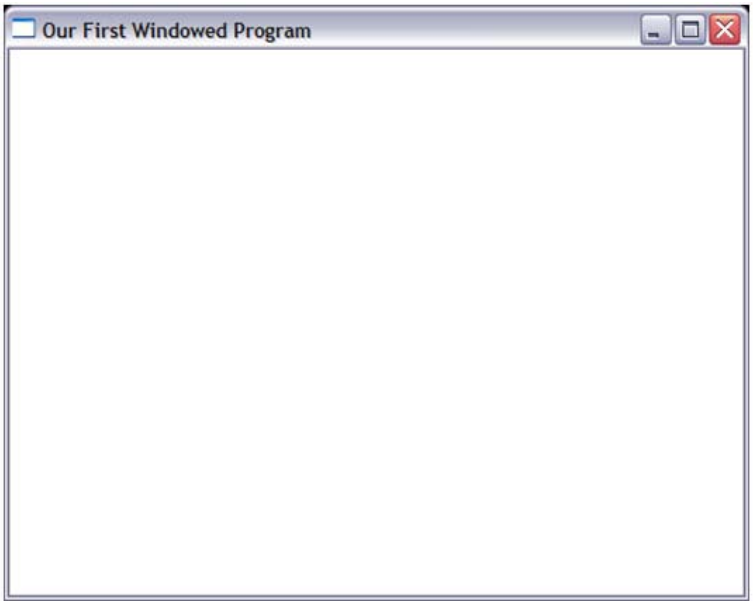
کد زیر نمونه یک برنامه ویندوز است که یک پنجره را میسازد»
عکس 2

```
// include the basic windows header file
#include <windows.h>
#include <windowsx.h>

// the WindowProc function prototype
LRESULT CALLBACK WindowProc(HWND hWnd,
UINT message,
WPARAM wParam,
LPARAM lParam);

// the entry point for any Windows program
int WINAPI WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPSTR lpCmdLine,
int nCmdShow)
{
// the handle for the window, filled by a function
HWND hWnd;
// this struct holds information for the window class
WNDCLASSEX wc;

// clear out the window class for use
ZeroMemory(&wc, sizeof(WNDCLASSEX));
```

```
// fill in the struct with the needed information
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = (WNDPROC)WindowProc;
wc.hInstance = hInstance;
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
wc.lpszClassName = L"WindowClass1";

// register the window class
RegisterClassEx(&wc);

// create the window and use the result as the handle
hWnd = CreateWindowEx(NULL,
L"WindowClass1", // name of the window class
L"Our First Windowed Program", // title of the window
WS_OVERLAPPEDWINDOW, // window style
300, // x-position of the window
300, // y-position of the window
500, // width of the window
400, // height of the window
NULL, // we have no parent window, NULL
NULL, // we aren't using menus, NULL
hInstance, // application handle
NULL); // used with multiple windows, NULL

// display the window on the screen
ShowWindow(hWnd, nCmdShow);

// enter the main loop:

// this struct holds Windows event messages
MSG msg;

// wait for the next message in the queue, store the result in 'msg'
while(GetMessage(&msg, NULL, 0, 0))
{
// translate keystroke messages into the right format
TranslateMessage(&msg);

// send the message to the WindowProc function
DispatchMessage(&msg);
}

// return this part of the WM_QUIT message to Windows
return msg.wParam;
}

// this is the main message handler for the program
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
// sort through and find what code to run for the message given
```

```
switch(message)
{
// this message is read when the window is closed
case WM_DESTROY:
{
// close the application entirely
PostQuitMessage(0);
return 0;
} break;
}

// Handle any messages the switch statement didn't
return DefWindowProc (hWnd, message, wParam, lParam);
}
```

در نهایت ،نتیجه کد بالا ایجاد پنجره پایین میشه:



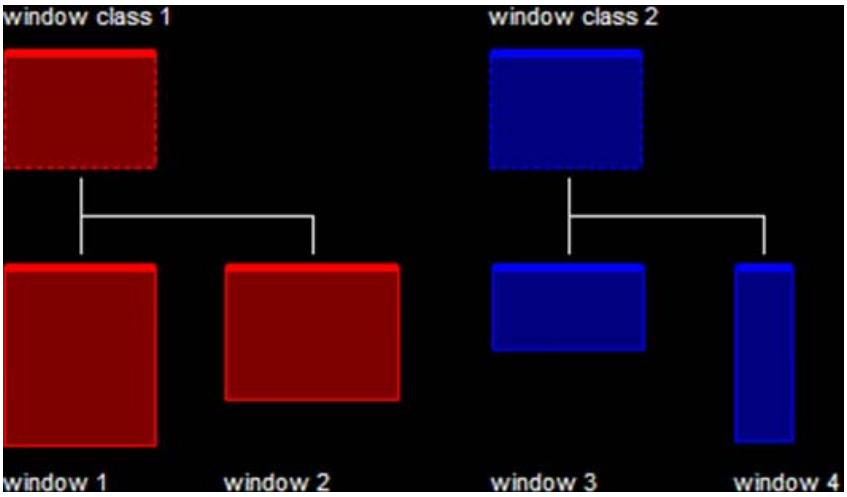
ساخت هر پنجره در برنامه نویسی ویندوز از سه قسمت تشکیل شده

1.Register The windows class
2.Create the window
3.show the window

خوب حالا هر قسمت را توضیح میدیم:

Registering the window class

به طور ساده یک کلاس پنجره یک ساختار پایه ای است که ویندوز برای جمع و جور کردن خواص و عمل پنجره های مختلف استفاده میکنه.یک کلاس پنجره در حقیقت یک قالب از خواص همیشگی و پایه یک پنجره است که برای وجود یک پنجره ساده نیاز است.شکل پایین اینو نشون میده (کلاس اول که به عنوان یک قالب برای بقیه است).



در این نمودار window class1 برای تعریف خواص پایه window1 استفاده شده(همینطور window 2 به همینصورت window class 2برای window3 و window4 هر پنجره دارای خواص خاص خودشه مثل اندازه مکان اجزایی که داخلش و ... ولی خواص پایه در همشون یکیه)
در این مرحله ما یک کلاس ویندوز را ثبت(register)میکنیم که به این معنی که ما به ویندوز میگیم یک پنجره بر پایه اطلاعاتی که ما میدیم بسازه. کد زیر کار ثبت پنجره ما (همراه خواصی که میخوایم) را انجام میده

```
// this struct holds information for the window class
WNDCLASSEX wc;

// clear out the window class for use
ZeroMemory(&wc, sizeof(WNDCLASSEX));

// fill in the struct with the needed information
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = (WNDPROC)WindowProc;
wc.hInstance = hInstance;
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
wc.lpszClassName = L"WindowClass1";

// register the window class
RegisterClassEx(&wc);
```

خوب حالا هر قسمت از این کد را توضیح میدم تا ببینیم چکار میکنه:

WNDCLASSEX wc
این ساختار (struct) که اطلاعات یک کلاس پنجره در اونه.اینجا نمیخوایم همه اجزای اونو توضیح بدم چون زیاداست. فقط قسمتی که به درد ما میخوره مهمه.اگه کاملش را میخواید میتونید توmsdn پیدا کنید .
خوب کلمه EX نشون میده که این ورژن جدید از ساختارWNDCLASS است که تقریباً یکی هستند و چیزایی که اضافه شده به کار ما نمیاد.و برای بقیه EX ها هم همینطور هست.
ZeroMemory(&wc, sizeof(WNDCLASSEX));

ZeroMemoryیک تابع که تمام بلاک حافظه را با نول مقدار دهی اولیه میکنه.آدرسی که در پارمتر اول داده شده تعیین میکنه که از کجای بلاک حافظه باید شروع کنه.پارامتر دوم طول بلاک را تعیین میکنه.که ما این دوتا را با دادن آدرس wc و اندازه نوع ساختار wc پر کردیم.(wndclass)

wc.cbSize = sizeof(WNDCLASSEX);

فکر میکنم این خودش معلومه که چیه.ما اینجا اندازه مورد نیاز ساختار را نشون میدیم که همون اندازه استانداره wndclassexاست اندازه را به وسیله sizeofمیگیریم.

wc.style = CS_HREDRAW | CS_VREDRAW;

در اینجا ما استایل پنجره را تعیین میکنیم.مقدارهایی که اینجا میشه استفاده کرد زیاد ولی ما تقریبا هیچ وقت به اونا تو game programming نیاز نداریم (you can find others in msdn).برای حالا ما از CS_HREDRAW و OR منطقی با CS_VREDRAWاستفاده میکنیم. کاری که اینا میکنه اینه که به ویندوز میگن که پنجره را در صورت حرکت دوباره ترسیم کنه یکیشون برای حرکت افقی و اون یکی برای حرکت عمودیه.این حالت برای یک پنجره مناسب ولی برای بازی نه! بعدا که ما پنجره را full screen کردیم این ها را بر میداریم.

wc.lpfnWndProc = (WNDPROC)WindowProc;

این مقدار به کلاس ویندوز میگه که وقتی پیغامی از ویندوز دریافت کرد(مثل پیام خروج برنامه) از چه تابعی برای پاسخ به اونا استفاده کنه .در برنامه ما اسم این تابع ()WindowProc است ولی میتونه هر اسمی داشته باشه مثله WndProcیا WinProc و یا حتی. ()asdfsadf
wc.hInstance = hInstance;

ما این قسمت را در بخش قبل توضیح دادیم این یک هندل برای کپی برنامه ما است.
wc.hCursor = LoadCursor(NULL, IDC_ARROW);

این عضو تصویر حالت عادی موس ما رو برای کلاس پنجره ذخیره میکنه .و بوسیله استفاده از مقدار بازگشتی ()LoadCursorاین کار را انجام میده. این تابع دارای دو پارامتر است :اولی هندل به وهله(hInstance)برنامه که گرافیک اشاره گر را ذخیره میکنه.که ما نیازی به اون نداریم و اونوNULL رها میکنیم.پارامتر دوم مقداری است که اشاره گر پیش فرض موس را در بر داره .بازم میتونید برید توMSDN برای بقیه انتخابها.

wc.hbrBackground = (HBRUSH)COLOR_WINDOW;

این عضو در بردارنده قلمویی که برای رنگ کردنbackground پنجره ما استفاده میشه.قلموها از بحث فعلی ما فراترند .ولی در اینجا اونا برای تعیین رنگbackground استفاده می شوند COLOR_WINDOW. تعیین میکنه که براش پنجره را به رنگ سفید رنگ کنه.

wc.lpszClassName = L"WindowClass1";

این نام کلاس پنجره است که ما میسازیم در اینجا ما اسم اونوwindowsClass1 گذاشتیم حتی اگر فقط یک کلاس پنجره داشته باشیم . مهم نیست چه نامی برای اون میزارید . حرفL قبل از رشته نام پنجره به کامپایلر میگه که این رشته از 16 bit unicode characterساخته شده به جایه اینکه از مقدار پیش فرض 8bit ansi characterساخته شده باشه.
RegisterClassEx(&wc);

و در آخر ما با این فرمان کلاس پنجره خود را ثبت میکنیم.این تابع فقط دارای یک پارامتر است که اونم آدرس ساختار .بقیه کارم با ویندوزه.!

2. Create the Window

مرحله بعد ساخت یک پنجره است.حالا که ما کلاس پنجره مون را ساختیم می تونیم پنجره هارو بر اساس اون بسازیم.
ما فقط به یک پنجره نیاز داریم پس مسیله مشکلی نیست. برای ساخت یک پنجره ما به کد زیر ادامه داریم

```
// create the window and use the result as the handle
hWnd = CreateWindowEx(NULL,
L"WindowClass1", // name of the window class
L"Our First Windowed Program", // title of the window
WS_OVERLAPPEDWINDOW, // window style
300, // x-position of the window
300, // y-position of the window
500, // width of the window
400, // height of the window
NULL, // we have no parent window, NULL
NULL, // we aren't using menus, NULL
hInstance, // application handle
NULL); // used with multiple windows, NULL
```

خوب فکر کنم اینقدر هم ساده نیست چون کلی پارامتر داره که باید توضیح داده بشه سخته نه؟ نه چون پارامتر ها اینجا سادست.خوب یه نگاهی به نمونه اول تابع میندازیم

```
HWND CreateWindowEx(DWORD dwExStyle,
LPCTSTR lpClassName,
LPCTSTR lpWindowName,
DWORD dwStyle,
int x,
int y,
int nWidth,
int nHeight,
HWND hWndParent,
HMENU hMenu,
HINSTANCE hInstance,
LPVOID lpParam);
```

DWORD dwExStyle,

این پارامتر از وقتی آمد که تابع RegisterClass() به RegisterClassEX() تبدیل شد . و از چهار پارامتر شامل شده که در انتخاب استایل دست ما رو بازتر میکنه و انتخاب های بیشتری داریم که بازم اینجا توضیح نمیدیم میتونید برید MSDN

LPCTSTR lpClassName,

این همون نام کلاس پنجره ای که ساختیم و میخوایم استفاده کنیم.چون فقط یک کلاس داریم از همون استفاده میکنیم یعنی L"WindowsClass1 "این رشته هم از 16bit unicode characterاستفاده میکنه بخاطر L در پشت اسم.

LPCTSTR lpWindowName,

این نام پنجره ای که میسازیم اسمش به صورت عنوان پنجره در بالا نمایش داده میشه. بازم16.... bit

DWORD dwStyle,

این جایی که انتخاب هامون را برای پنجره بکنیم. مثلا میتونید کلید های minimize maximizeرا بردارید و کارایی از این قبیل بازم برای اطلاعات بیشتر برید!! MSDN ما در اینجا از مقدار WS_OVERLAPPEDWINDOW که میانبریه که شامل چندین مقدار با همه که یک استاندار برای یک پنجره عادیه

int x

موقعیت پنجره را برروی محورx ها روی صفحه تعیین میکنه.
int y

موقعیت پنجره را برروی محورy ها روی صفحه تعیین میکنه.
int nWidth,

پهنا یا عرض پنجره را تعیین میکنه.
int nHeight,

طول پنجره را تعیین میکنه.
HWND hWndParent,

این پارامتر به ویندوز میگه که پنجره ما از کدوم پنجره منشا گرفته شده که برای ما از هیچ پنجره منشایی گرفته نشده(پنجره منشا یا parent پنجره که پنجره های دیگه را شامل میشه)چون ما نداریم مقدارشوNULL میزاریم.
HMENU hMenu,

این یک هندل به menu bar که ما بازم اینجا نداریم پس اینمNULL
HINSTANCE hInstance,

اینم یک هندل به وهله hInstance. مقدار دهی کنید
LPVOID lpParam

این یک پارامتر که ما میتونستیم استفاده کنیم در صورتی که چندتا پنجره داشته باشیم. پس اینم اینجاNULL
Return Value

مقدار بازگشتی یک هندل که ویندوز برای پنجره جدید اختصاص میده. ما اینو مستقیما درhWnd ذخیره میکنیم.
hWnd = CreateWindowEx(NULL,
...

3. Show the Window

نشون دادن پنجره آسون ترین قسمت کاره به راحتی ساخت یک جعبه پیام. برای اینکار به یک تابع که دوتا پارامتر داره نیاز داریم. نمونه اولیه:

```
BOOL ShowWindow(HWND hWnd,
int nCmdShow);
```

پارامتر اول هندل پنجره ای که ساختیم میخوایم نشون داده بشه
int nCmdShow

آخرین پارامترWINMAIN یادتونه؟ این همونه که البته ما نیازی بهش نداریم ولی میزاریم اینجا خود ویدوز به ما بگه چکار کنیم.در بازی اهمیتی نداره که ویندوز به ما میگه چکار کنیم چون اونموقع در حالتFull Screen هستیم.
در اینجا بخوایم یا نخوایم مقدارnCmdShow میزاریم.

متاسفانه هنوز کار تموم نشده .

مدیریت رویدادهای ویندوز و پیامها

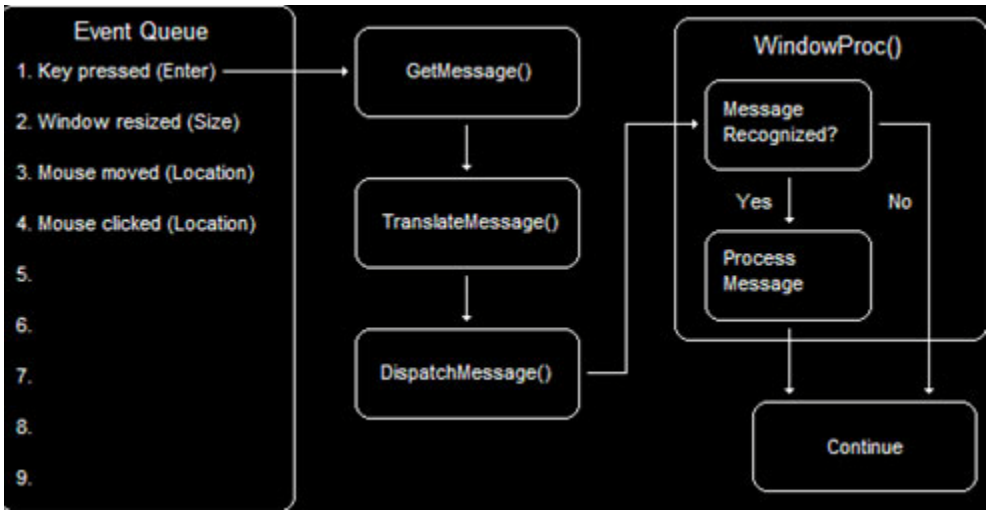
همونطور که قبلا بحث کردیم برنامه نویسی ویندوز بر پایه رویدادهاست به این معنی که پنجره ما فقط وقتی نیاز داره کاری را انجام بده که ویندوز به اون میگه .در حالت دیگه فقط صبر میکنه

زمانی که یک ویندوز یک پیام را به ما میده چند تا چیز به سرعت رخ میده.این پیام در صف رویداد ها قرار میگیره . و ما از تابع ()GetMessage برای گرفتن اون پیام استفاده میکنیم(اون پیام را از صف پیام ها میگیریم) از تابع ()translatemessageاستفاده میکنیم تا پیام را به فرمت مورد نیاز تبدیل کنیم و از تابع()dispatchmessage استفاده

میکنیم و پیام را به WindowProc()میدیم تا در اونجا کد لازم برای جواب به پیام انتخاب بشه

اگر بخوایم به صورت نمودار فرضش کنیم به صورت شکل پایین میشه

سایز این تصویر تغییر یافته است.برای دیدن سایز واقعی آن اینجا کلیک کنید.سایز این تصویر 279*546 میباشد.



مدیریت رویداد ها نصف برنامه ما رو شامل میشه . و خودش به دو قسمت تقسیم میشه

1. The Main Loop
2. The WindowProc() Function

حلقه اصلی شامل تابع های GetMessage TranslateMessage و DispatchMessage است .تابع WindowProc() تنها شامل کد اجرای پیام های حتمی(مورد نیاز همیشه) برنامه است

1. The Main Loop

همونطور که تو نمودارم معلومه این قسمت از سه تابع تشکیل شده . هر تابع تقریبا سادست و ما نیازی به بحث در مورد اونا به صورت دقیق تر نداریم. کد پایین کد مورد نیاز برای حلقه اصلیه برنامست

```
// this struct holds Windows event messages
MSG msg;

// wait for the next message in the queue, store the result in 'msg'
while(GetMessage(&msg, NULL, 0, 0))
{
// translate keystroke messages into the right format
TranslateMessage(&msg);

// send the message to the WindowProc function
DispatchMessage(&msg);
}
```

MSG msg;

MSGیک ساختار که تمام اطلاعات مربوط به یک رویداد تنهای پیام است(Single even message) . به طور عادی نیازی به دسترسی به اجزای داخلش ندارید ولی اعضای اونو این زیر نوشتم

Member Description

HWND hWnd Contains the handle of the window which recieved the message.

UINT message Contains the identifier of the message sent.

WPARAM wParam Contains additional information about the message. The exact meaning depends on what message was sent.

LPARAM lParam Identical to WPARAM, and simply contains more information.

DWORD time Contains the exact time at which the message was posted in the event queue.

POINT pt Contains the exact mouse position, in screen coordinates, when the message was posted.

while(GetMessage(&msg, NULL, 0, 0))

()GetMessageیک تابع که هر پیامی را از صف پیام ها میگیره و داخل ساختار MSGمیزاره. و همیشه مقدار true را بر میگردون به جز زمانی که برنامه میخواد خارج بشه کهfalseمیده.وقتی false میده ما از حلقه خارج میشیم.
تابع ()GetMessage دارای چهار پارامتر است به صورت زیر..

```
BOOL GetMessage(LPMSG lpMsg,
                HWND hWnd,
                UINT wMsgFilterMin,
                UINT wMsgFilterMax);
```

LPMSG lpMsg,

این پارامتر یک اشاره گر به ساختار پیام است مثل همونی که بحث کردیم. تنها کاری که باید بکنیم اینه که آدرس ساختاری که ساختیم اینجا بدیم.

HWND hWnd,

این یک هندل به این پنجره ای که باید پیام از اون بیاد.هرچند ممکنه توجه کنی که ما اینجاNULL گذاشتیم در این پارامتر NULL به این معنی که پیام بعدی برای هر کدوم از پنجره های ما بگیر. اگه بخوایم میتونیم هندل پنجره را بزاریم ولی ما اینجا فقط یکی داریم و نیازی به این نیست.و فقطNULLمیزاریم.

UINT wMsgFilterMin, UINT wMsgFilterMax

این پارامتر میتونه برای محدود کردن نوع پیام هایی که برای ما از صف پیام ها میاد استفاده بشه.باری مثال استفاده از WM_KEYFIRST در wMsgFilterMIn و استفاده از WM_KEYLAST پیام ها را به نوع پیام هایی که مال کی بورد محدود میکنه WM_KEYFIRST .و WM_KEYLAST هم معنی مقدارهای integer اولین و آخرین پیام های کی بوردن.
به همینصورت WM_MOUSEFIRST و WM_MOUSELAST برای محدود کردن به پیام های موس.اگه0 بزاری براش هر پیامی را بدون در نظر گرفتن مقدارش میگیره

TranslateMessage(&msg);

این تابع برای تبدیل عمل فشار کلید های معین را به یک فرمت مشخص انجام میده ولی برای ما در بازی مهم نیست چون در آینده ما از Directinput برای شناسایی فشار کلیدها و ورودی های مختلف مثل موس یاgame pad استفاده میکنیم درمورد این تابع خوبه بعدا خودتون یه مطالعه ای کنید اینجا کم توضیح دادم

DispatchMessage(&msg);

به طور ساده همون کاری را انجام میده که اسمش میگه. این تابع پیام ها را میفرسته به تابع()WindowProc یکه پارامتر هم دارد که آدرس ساختار پیام که ساختیم,

2. The WindowProc() Function

یک خلاصه از کاری که تا اینجا انجام شد این بود که ما پیام ها را گرفتیم (getmessage)ترجمه کردیم (translatemessage)و به WINDOWPROC فرستادیم.(dispatchmessage)
زمانی که تابع WindowProc فراخوانی میشه چهار قسمت از ساختار MSG به اون رد میشه . درزیر نمونه اولیه آمده

```
LRESULT CALLBACK WindowProc(HWND hWnd,
UINT message,
WPARAM wParam,
LPARAM lParam);
```

اگر نگاهی به ساختار msg انداخته باشید میبینید اینه تو اون بود میتونید همونجا بخونید که چی هستن.مرسی!
زمانی که یک پیام داخل windowproc میشه ما میتونیم از شناسه uMSG استفاده کنیم و بفهمیم که کدوم پیام آمده.
بیشتر برنامه نویسا و تمام برنامه نویسان بازی که من میشناسم از()switch برای این کار استفاده میکنن.در پایین مثالش آمده:

```
// sort through and find what code to run for the message given
switch(message)
{
// this message is read when the window is closed
case WM_DESTROY:
{
// ...
// ...
} break;
}
```

در اینجا چک کرده که اگر پیام WM_DESTROY امد چکار کن این پیام وقتی میاد که پنجره در حال بسته شدن و زمانی که بسته شد کد ما اجرا میشه وقتی ما فقط این پیام را چک میکنیم دیگه به بقیه پیام ها کار نداریم و برنامه اونارو نادیده میگیره و در برابر اونا کاری نمیکنه.تمام کاری که ما اینجا میکنیم اینه که به برنامه بگیم ما کارمون با این پیام تمومه و مقدار 0 را بازگشت میدیم و برای نشان دادن اینکه همه چیز تمیز و پاک و باید برنامه خارج بشه ما از کد زیر استفاده میکنیم:

```
case WM_DESTROY:
{
// close the application entirely
PostQuitMessage(0);
return 0;
} break;
```

تابع ()PostQuitMessage پیام WM_QUIT را میفرسته که دارای یک مقدار integer صفر است.همونطور که گفتیم وقتی برنامه خارج میشه که () getmessage مقدار false بازگشت بده و وقتی ما 0 را میفرستیم این تابع مقدار false میده.
در نهایت کاری که این تابع میکنه اینه که به کار تابع winmain پایان میده.

اگر کد برنامه را دیده باشید میفهمید که یک تابع دیگه مونده که باید بگیم.()DefWindowProc.کاری که این تابع میکنه اینه که پیام هایی را که ما کار بهشون نداشتیم را مدیریت میکنه به جای ما. به طور خلاصه پیام هایی را که

ما برای آنها 0 بازگشت ندادیم را مدیریت میکنه .به خاطر همین ما اینو در آخر تابع قرار دادیم تا هرچیزی که ما از دادیم و بگیره...

ادامه دارد...

©Copyright 2010 / Animationdata.com & Partners