03. PeekMessage와 키입력

```
// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

HWND g_hWnd;
HDC  g_hDC;
bool g_bLoop = true;
RECT g_tPlayerRC = { 100, 100, 200, 200 };
```

자주 사용할 변수 전역변수로 선언

<wWinMain>

```
    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    // 화면 그리기용 DC 생성.
    g_hDC = GetDC( g_hWnd );

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_ASSORTROCKWINAPI));

    MSG msg;

    // Main message loop:
    while( g_bLoop )
    {
        if ( PeekMessage( &msg, nullptr, 0, 0, PM_REMOVE ) )
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        // 윈도우 데드 타임일 경우
        else
        {
            static int iCount;
            ++iCount;

            if( iCount == 5000 )
            {
                iCount = 0;
                Run();
            }
        }
    }

    ReleaseDC( g_hWnd, g_hDC );

    return (int) msg.wParam;
}
```

GetMessage가 아닌 PeekMessage 이용

메시지 들어온거 가져오고 큐에서 삭제

데드타임에 게임을 만들어나감.

생성한 DC를 사용이 끝나면 Release

왜 PeekMessage를 이용하는가?

→ GetMessage는 메시지 큐에 메시지가 들어오기 전까지 빠져나오지 않는다. 그렇기 때문에 이벤트가 발생하기 전까지 어떠한 것도 할 수가 없다.

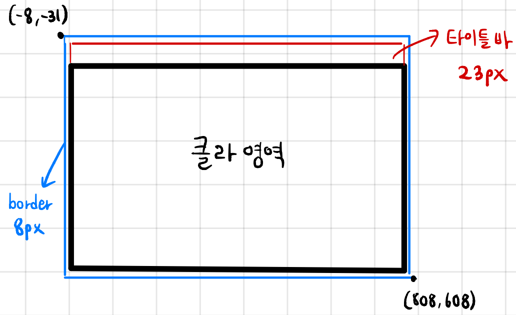
PeekMessage는 메시지가 들어오면 true, 없으면 false를 리턴하고, 메시지가 없을때 false 인 상태 즉, 아무런 이벤트가 없을 때를 윈도우의 데드타임 이라하고, 이 데드타임을 활용해 게임을 만든다.

그리고 메시지 루프는 프로그램 종료시 끝나야 하므로 아래와 같이 윈도우 프로시저의 메시지에서 루프 제어 변수를 바꿔준다.

```
        // 윈도우를 종료시킬 때 들어오는 메시지.
    case WM_DESTROY:
        g_bLoop = false;
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

```cpp
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }
   // 윈도우 핸들 전역변수에 생성한 윈도우 핸들 넣기.
   g_hWnd = hWnd;

   // 실제 윈도우 타이틀바나 메뉴를 포함한 윈도우 크기를 구해줌.
   RECT rc = { 0,0,800,600 };
   AdjustWindowRect( &rc, WS_OVERLAPPEDWINDOW, FALSE );

   // 위에서 구해준 전체 윈도우 크기를 통해 클라이언트 영역의 크기를 원하는 크기로 맞춰줘야 함.
   SetWindowPos( hWnd, HWND_TOPMOST, 100, 100, rc.right - rc.left, rc.bottom - rc.top, SWP_NOMOVE | SWP_NOZORDER );

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

HWND_TOPMOST: 최상단에 띄움 / 100, 100: 띄울 위치 / SWP_NOMOVE: 위치 변경X / SWP_NOZORDER: Z순서유지



따라서 클라 크기가 정확히 800×600이 되려면 크기가
816(808-(-8))×639(608-(-31))
이 되어야 양 옆 border 총 16px
제외 해서 가로 800px
border 16px + 타이틀 바 23px을 제외
세로 600px이 된다.

```cpp
void Run()
{
   RECT wndRect;
   GetClientRect( g_hWnd, &wndRect );

   if( GetAsyncKeyState( 'D' ) & 0x8000 )
   {
      g_tPlayerRC.left += 1;
      g_tPlayerRC.right += 1;
      if( g_tPlayerRC.right > wndRect.right )
      {
         g_tPlayerRC.right = wndRect.right;
         g_tPlayerRC.left = wndRect.right - 100;
      }
   }
   if( GetAsyncKeyState( 'A' ) & 0x8000 )
   {
      g_tPlayerRC.left -= 1;
      g_tPlayerRC.right -= 1;
      if( g_tPlayerRC.left < wndRect.left )
      {
         g_tPlayerRC.left = 0;
         g_tPlayerRC.right = wndRect.left + 100;
      }
   }
   if( GetAsyncKeyState( 'W' ) & 0x8000 )
   {
      g_tPlayerRC.top -= 1;
      g_tPlayerRC.bottom -= 1;
      if( g_tPlayerRC.top < wndRect.top )
      {
         g_tPlayerRC.top = wndRect.top;
         g_tPlayerRC.bottom = wndRect.top + 100;
      }
   }
   if( GetAsyncKeyState( 'S' ) & 0x8000 )
   {
      g_tPlayerRC.top += 1;
      g_tPlayerRC.bottom += 1;
      if( g_tPlayerRC.bottom > wndRect.bottom )
      {
         g_tPlayerRC.bottom = wndRect.bottom;
         g_tPlayerRC.top = wndRect.bottom - 100;
      }
   }

   Rectangle( g_hDC, g_tPlayerRC.left, g_tPlayerRC.top, g_tPlayerRC.right, g_tPlayerRC.bottom );
}
```

GetClientRect → 클라이언트 영역 크기를 얻어오는 함수

GetAsyncKeyState 함수

비동기로 처리.
호출된 시점에서 키 상태를 조사하여 메시지 큐를 거치지 않고 바로 리턴해주므로 키 입력을 바로 읽을 수 있음.
(GetKeyState는 메시지 큐를 거침)
여러 입력을 멀티로 받을 수 있음.

리턴 값

값	설명
0x0000	이전에 누른 적이 없고 호출 시점에도 눌려있지 않은 상태
0x0001	이전에 누른 적이 있고 호출 시점에는 눌려있지 않은 상태
0x8000	이전에 누른 적이 없고 호출 시점에는 눌려있는 상태
0x8001	이전에 누른 적이 있고 호출 시점에도 눌려있는 상태