WINAPI
-CHAPTER8-

SOULSEEK

목차

GETKEYSTATE

1. GETKEYSTATE

- **WM_KEYDOWN** 명령으로 체크 할 경우 키가 최근에 눌려진 키의 값만을 체크하기 때문에 중복키가 적용이 되지 않는다.
- **GetKeyState**는 **0x8000**의 **16**진수와 가상 키보드의 값을 **&**연산을 통해 **0**인지 아닌지를 통해 키가 눌려져 있는지 체크하게 되므로 중복키도 모두 적용된다.
- **GamePlay**에 적합한 **Key Check** 방식이다.

```
case WM_TIMER:

    if (GetKeyState(VK_LEFT) & 0x8000)
        x -= 10;
    if (GetKeyState(VK_RIGHT) & 0x8000)
        x += 10;
    if (GetKeyState(VK_UP) & 0x8000)
        y -= 10;
    if (GetKeyState(VK_DOWN) & 0x8000)
        y += 10;

    InvalidateRect(hWnd, NULL, false);

return 0;
```

1. GETKEYSTATE

학습과제

1. GetKeyState예제로 주어진 코드를 이용해서 비트맵 클래스에 어울리게 적용해보자
2. 1번 과제를 이용해 특정 키를 누르면 점프하는 코드를 추가해서 구현해보자.
3. 2번 과제를 이용해 상, 하, 좌, 우 움직일 때마다 애니메이션 하게 만들어 보자.

더블 버퍼링

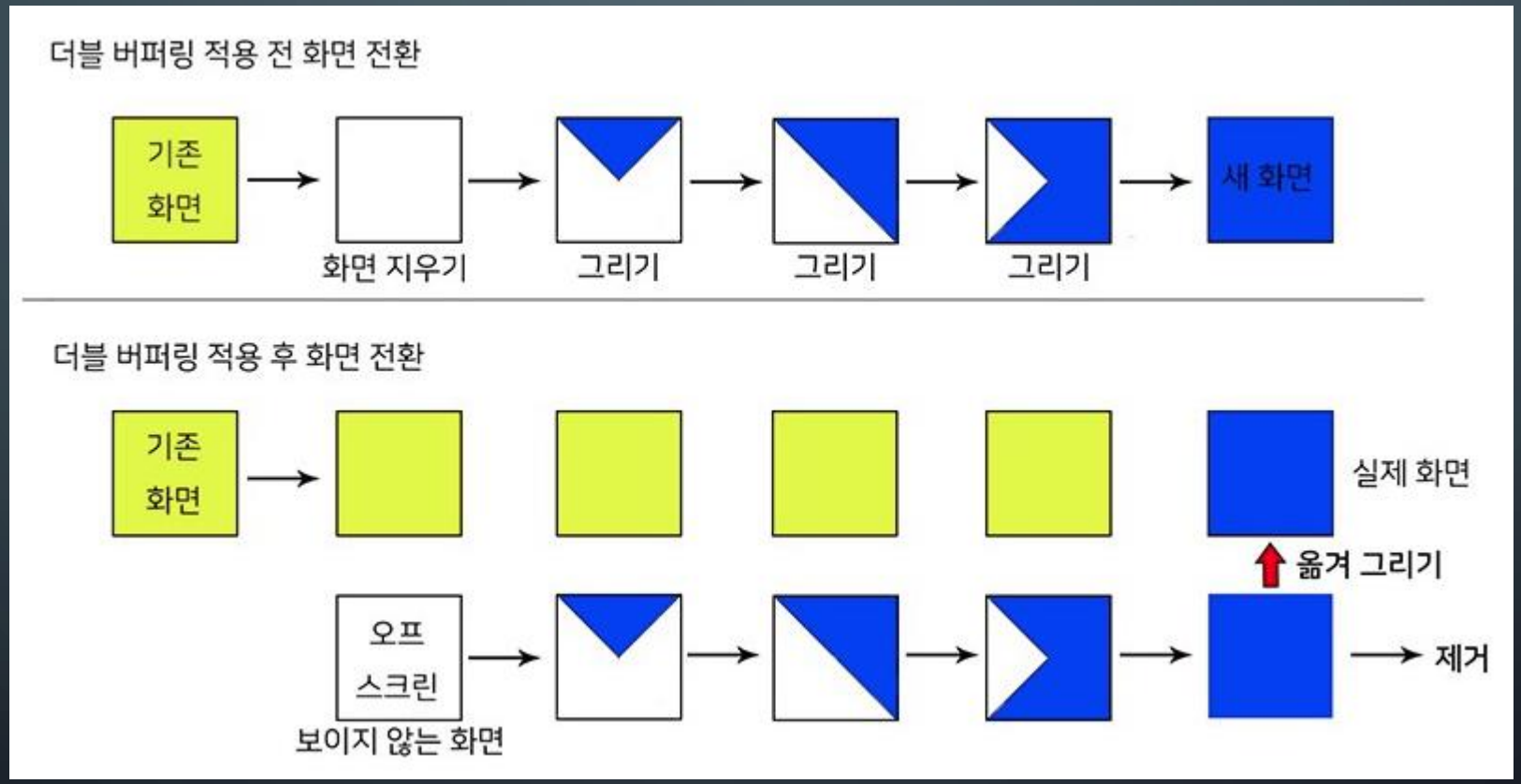
2. 더블 버퍼링

DubleBuffering이 필요한 이유

- 게임다운 게임을 만들기 위해 필요하다.
- WM_PAINT를 호출할 때 특정영역을 다시 그리는 InvalidateRect함수는 특정 영역을 지우고 다시 그리게 된다.
- 백그라운드에 아무것도 남지 않기 때문에 다시 그리는 명령이 올 때마다 깜빡이는 걸 느낄 수 있었을 것이다.
- 기존의 화면을 뿌려주고 그 다음에 뿌려줘야 할 그림은 임시 저장소에 저장을 하는 BackDC에 그린 후 MemDC에 전달해주고 기존 화면에 뿌려준다.

이 순서를 이해하면 지우는 방법으로 처리한다기보다 화면을 뿌려줄 준비가 될 때까지 기존 화면을 유지할 DC를 하나 더 만들어서 유지해 주가다 준비가 완료되면 그 위에 덮어씌우는 것이다.

2. 더블 버퍼링


더블 버퍼링 적용 전 화면 전환
기존 화면
화면 지우기
그리기
그리기
그리기
새 화면
더블 버퍼링 적용 후 화면 전환
기존 화면
실제 화면
옮겨 그리기
오프 스크린
보이지 않는 화면
제거

2. 더블 버퍼링

필요한 변수 선언 부분

HDC g_MemDC[3];
HBITMAP g_hBitmap[3];
HBITMAP g_hOld[3];

- 그림은 2장이지만 2장의 그림을 계속 갱신해서 그려줄 **MemDC가 하나 더 필요**하다.
- 메모리상의 DC가 확보되었지만 도화지 역할을 해줄 공간을 위해 아무것도 없는 공간에 검정색 비트맵을 그려준다 - **Bitmap & OldBitmap이 하나씩 더 필요**하다.

2. 더블 버퍼링

ImageLoad 부분

- 숨겨 그릴 장소(도화지)를 준비한다.
- 그림이 그려질 부분이 필요하기 때문에 검정색의 비트맵을 준비한다.
- 이 부분만 출력해도 정상 작동이 되지 않으니 혼자 출력되는 일이 없도록 하자.

```
g_MemDC[0] = CreateCompatibleDC(hdc);
g_hBitmap[0] = CreateCompatibleBitmap(hdc, 1024, 768);
g_hOld[0] = (HBITMAP)SelectObject(g_MemDC[0], g_hBitmap[0]);
```

- 숨겨 그릴 장소(도화지)에 배경을 그릴 준비를 한다.

```
g_MemDC[1] = CreateCompatibleDC(g_MemDC[0]);
g_hBitmap[1] = (HBITMAP)LoadImage(NULL, "back.bmp", IMAGE_BITMAP, 0, 0
      , LR_CREATEDIBSECTION | LR_DEFAULTSIZE | LR_LOADFROMFILE);
g_hOld[1] = (HBITMAP)SelectObject(g_MemDC[1], g_hBitmap[1]);
```

- 숨겨 그릴 장소(도화지)에 캐릭터를 그릴 준비를 한다.

```
g_MemDC[2] = CreateCompatibleDC(g_MemDC[0]);
g_hBitmap[2] = (HBITMAP)LoadImage(NULL, "char.bmp", IMAGE_BITMAP, 0, 0
, LR_CREATEDIBSECTION | LR_DEFAULTSIZE | LR_LOADFROMFILE);
g_hOld[2] = (HBITMAP)SelectObject(g_MemDC[2], g_hBitmap[2]);
```

2. 더블 버퍼링

실제로 그림이 그려졌다 지워졌다 하는 곳은 g_MemDC[0]

- 숨겨 그릴 장소(도화지)에 배경을 그린다.

```
BitBlt(g_MemDC[0], 0, 0, 1024, 768, g_MemDC[1], 0, 0, SRCCOPY);
```

- 숨겨 그릴 장소(도화지)에 캐릭터를 그린다.

```
TransparentBlt(g_MemDC[0], x, y, 240, 192, g_MemDC[2], 0, 0, 240, 192,
    RGB(255, 0, 255));
```

- 숨겨 그린 것을 원래 보여야 할 **hdc**에 그린다.
- 단지 **hdc**에 그림을 덮어 그리는 행위만 하게 된다.
- 그리는 행위는 하지만 **뒷배경이 공백이 아니고 무언가를 계속 그리기 때문에 그림이 존재해서 그리는 동안 해당 그림을 보여주고 있는 것이라고 생각하면 된다.**

```
BitBlt(hdc, 0, 0, 1024, 768, g_MemDC[0], 0, 0, SRCCOPY);
```

2. 더블 버퍼링

학습과제

1. GetKeyState로 작성한 과제를 더블 버퍼링으로 고쳐보자(클래스로 작업한 것으로).

PEEKMESSAGE

3. PEEKMESSAGE

```
while (GetMessage(&Message, NULL, 0, 0)) //메시지를 구분한다.
{
    TranslateMessage(&Message); // WM_QUIT를 제외한 모든 메시지 전달
    DispatchMessage(&Message); // hWnd에게 메시지가 있다고 호출 전달
}
```

GetMessage()

- **Thread** 메시지 큐에 대기중인 메시지를 꺼내 첫 번째 인수로 전달된 메시지 구조체 복사하고 메시지 큐에서 해당 메시지를 제거한 뒤 **TRUE**를 리턴한다. 단, **WM_QUIT**일 경우 **FALSE**를 리턴한다.
- 복사한 메시지를 **TranslateMessage**에 전달한다.
- 다른 프로세스가 **CPU**를 사용하도록 잠시 대기하다가 메시지가 있을 때만 동작한다.
- 즉, 지속적인 동작체크가 아닌 특정 메시지가 있어야 체크가 가능하다.

TranslateMessage()

- 전달 된 메시지 중 키 입력과 관련된 메시지를 **WM_CHAR**으로 바꿔 주는 역할을 하고 키 입력이 아닌 경우에는 아무 동작을 하지 않는다.

DispatchMessage()

- 함수가 존재하는 **hWnd**가 어디인지 정확히 지정하고 **WinProc**으로 메시지를 전달해주는 역할을 한다.

3. PEEKMESSAGE

```
while(true)
{
	if (PeekMessage(&Message, NULL, 0, 0, PM_REMOVE))
	{
		if (Message.message == WM_QUIT)
			break;

		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	else
	{
		//메시지가 없다면 어떤 일을 처리 할 것인가????
	}
}
```

PeekMessage()

- 항상 메시지가 존재하는지 체크를 하고 있다가 메시지가 존재하면 메시지를 전달하고 **TRUE,** 존재 하지 않으면 **FALSE**를 리턴 하고 빠져 나오게 된다**.**
- 항상 메시지를 감시해줘야 할 필요성이 있기 때문에 무한 반복문안에 존재하다가 조건에 의해 빠져 나오게 해야 한다**.**
- 이런 코드속성을 이용해 메시지를 처리 할 수 없는 상황 즉**, FALSE**인 상황에서는 무한 반복 안에서 윈도우 관련 처리가 없기 때문에 지속적인 업데이트 명령을 내릴 수 있는 상황이 된다**.**

3. PEEKMESSAGE

```
HDC hdc;
hdc = GetDC(hWnd);

while(true)
{
	if (PeekMessage(&Message, NULL, 0, 0, PM_REMOVE))
	{
		if (Message.message == WM_QUIT)
		break;

		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	else
	{
		SetPixel(hdc, rand() % 500, rand() % 400, RGB(rand() % 256, rand() % 256,
			rand() % 256));
	}
}

ReleaseDC(hWnd, hdc);
```

3. PEEKMESSAGE

학습과제

1. PPT의 예제를 돌아가게끔 완성해서 실행해 보고 변형해서 자신이 테스트를 해보자.

GAMEFRAME

4. GAMEFRAME

- PeekMassege의 특성을 이용해서 메시지가 호출되지 않을 시에는 항상 체크해서 윈도우 메시지의 도움 없이 상시 갱신되는 코드를 작성할 수 있다.
- 메시지가 없는 상황에서 갱신처리를 해야 하는 update 함수를 구비해두면 가능하다.

```
//윈도우를 만들고 나면 초기화 해준다.
g_GameFrame.Init(hWnd);

while (true)
{
	// 메시지큐에 메시지가 있으면 메시지 처리
	if (PeekMessage(&Message, NULL, 0, 0, PM_REMOVE))
	{
		if (Message.message == WM_QUIT)
			break;

		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	else
	{
		//메세지가 없을때 업데이트를 진행한다.
		g_GameFrame.Update();
	}
}

//종료직전에 릴리즈 해준다.
g_GameFrame.Release();
```

4. GAMEFRAME

학습과제

1. 예제 처럼 구성하는 방법으로 제공하는 리소스와 함께 서커스 게임 1Stage를 만들어 보자.