



WINAPI

-CHAPTER7-

SOULSEEK

목차

1. 직접적으로 이미지 로드하기
2. 투명처리와 부분그리기
3. 충돌처리 함수

The background is a dark blue gradient with faint, large circular patterns. In the corners, there are white line art illustrations of circuit boards or neural networks, featuring lines and small circles.

직접적으로 이미지 로드하기

1. 직접적으로 이미지 로드하기

LoadImage(HINSTANCE hinst, LPCTSTR lpszName, UINT uType, int cxDesired, int cyDesired, UINT fuLoad);

- 리소스편집기에 등록하지 않고 파일경로를 이용해서 **Bitmap**을 로드한다.
- **1번째 인수 : NULL**고정.
- **2번째 인수 : FileName**(경로 - 파일이름을 포함한 경로, 기본경로는 프로젝트 폴더)
- **3번째 인수 : IMAGE_BITMAP, IMAGE_ICON, IMAGE_CURSOR**
- **4, 5번째 인수 :** 아이콘이나 커서일 경우 너비와 높이, 비트맵이면 **0, 0**
- **6번째 인수 :** 여러가지 플레그 옵션으로 그리기 모드를 바꿀 수 있다.
 - 파일 그대로 표현하는 옵션 - **LR_CREATEDIBSECTION | LR_DEFAULTSIZE | LR_LOADFROMFILE**

2. 투명처리와 부분그리기

```
Bool TransparentBlt(HDC hdcDest, int xoriginDest, int yoriginDest, int wDest,  
int hDest, HDC hdcSrc, int xoriginSrc, int yoriginSrc, int wSrc, int hSrc, UINT  
crTransparent);
```

- **msimg32.lib;** 라이브러리 링크가 필요
- 특정색깔을 안보이게 하는 투명처리가 가능하다.
- 부분 그리기로 텍스트팩으로 스프라이트 애니메이션을 구현 할 수 있다.
- **1번째** 인수 : 이미지를 출력할 위치의 핸들(**hdc**)
- **2, 3번째** 인수 : 이미지를 출력할 위치인 **x, y**좌표
- **4, 5번째** 인수 : 출력할 이미지의 너비, 높이 이미지의 크기가 변경
- **6번째** 인수 : 이미지의 핸들
- **7, 8번째** 인수 : 가져올 이미지의 시작지점인 **x, y**좌표
- **9, 10번째** 인수 : 원본 이미지로부터 해당 크기만큼 잘라낼 이미지의 너비, 높이
- **11번째** 인수 : 투명하게 할 **RGB** 색상 - 분홍색을 지정**RGB(255, 0, 255);**

2. 투명처리와 부분그리기

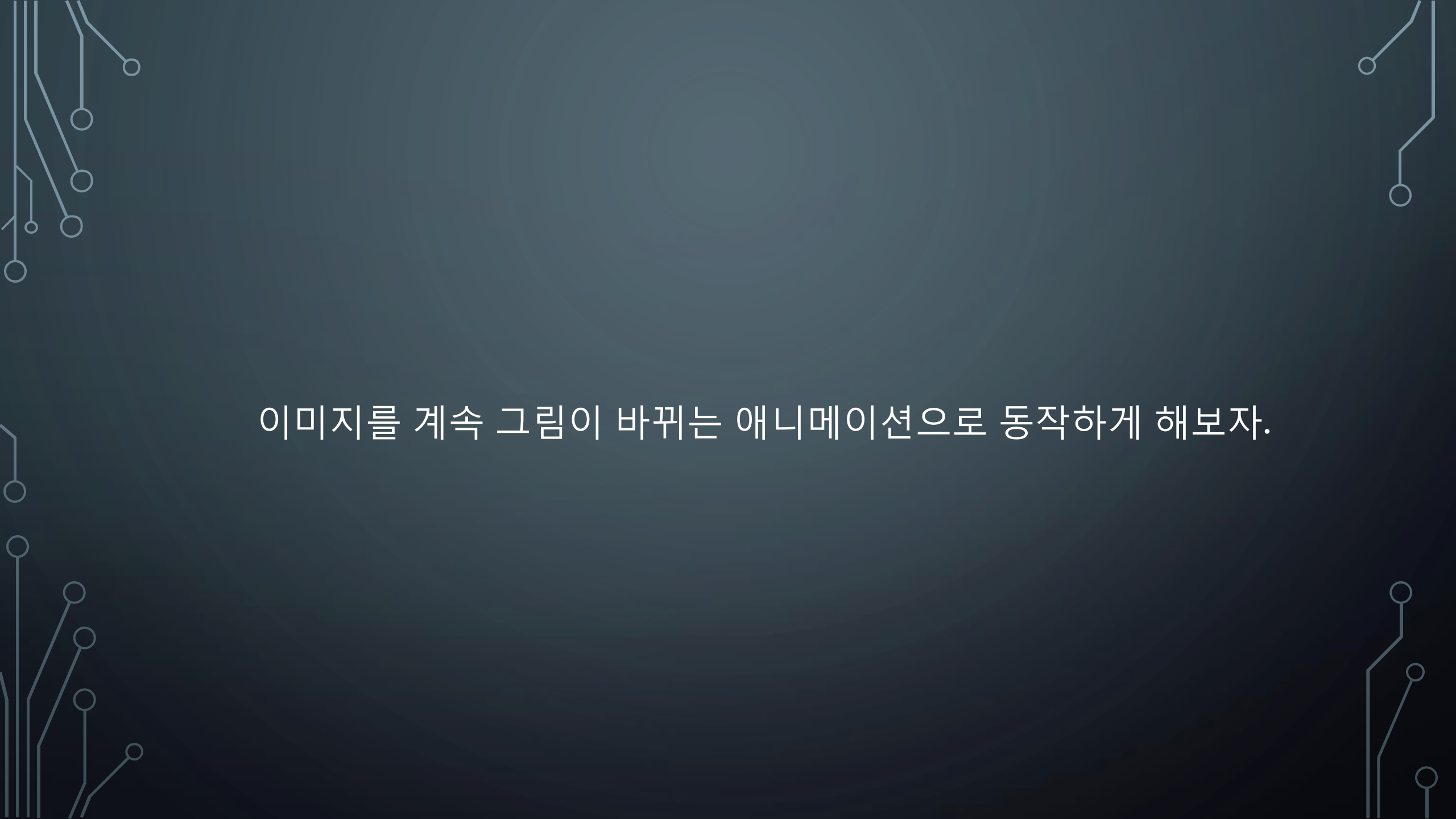
```
TransparentBlt(hdc, 350, 50, bit.bmWidth, bit.bmHeight, hMemDC, 0, 0,  
bit.bmHeight, bit.bmHeight, RGB(255, 0, 255));
```



2. 투명처리와 부분그리기

```
TransparentBlt(hdc, 650, 50, (bit.bmWidth / 4), (bit.bmHeight / 4), hMemDC,  
(bit.bmWidth / 4) * 1, (bit.bmWidth / 4) * 2, bit.bmWidth / 4, bit.bmHeight / 4,  
RGB(255, 0, 255));
```



The background is a dark blue gradient with faint, large circular patterns. In the corners, there are white line art illustrations of circuit boards or neural networks, featuring lines and small circles.

이미지를 계속 그림이 바뀌는 애니메이션으로 동작하게 해보자.

2. 충돌체크

```
BOOL PtlInRect(RECT *lprc, POINT pt);
```

- **POINT**형 변수가 지정한 **RECT**안에 존재하는지 검사할 때 사용
- **Ex)**마우스 포인트가 지정한 **RECT**안에 있다.
- **1번째 인자 : RECT**의 주소
- **2번째 인자 : 비교한 POINT**

```
RECT in_Rect = { 100, 100, 300, 300 };  
POINT in_Pt;
```

```
switch (iMessage)
```

```
{
```

```
    case WM_LBUTTONDOWN:
```

```
        in_Pt.x = LOWORD(IParam);
```

```
        in_Pt.y = HIWORD(IParam);
```

```
        if (PtlInRect(&in_Rect, in_Pt))
```

```
        {
```

```
            MessageBox(hWnd, "안에 있다", "!", MB_OK);
```

```
        }
```

```
    return 0;
```

```
}
```

2. 충돌체크

```
BOOL IntersectRect(RECT* rcTemp, RECT* RECT1, RECT* RECT2);
```

- 두 개의 사각형이 겹치는 교집합 부분이 있는지 체크한다.
- 교집합이 존재하면 **TRUE**를 반환하기 때문에 충돌체크에 사용된다.
- **1번째 인자** : 교집합을 저장할 **RECT**의 주소
- **2, 3번째 인자** : 교집합을 구할 **RECT**의 주소

```
case WM_TIMER:
```

```
    if (!IntersectRect(&rcTemp, &rcRectangle1, &rcRectangle2))  
    {  
        rcRectangle1.left += 10;  
        rcRectangle1.right += 10;  
    }
```

```
    InvalidateRect(hWnd, NULL, TRUE);
```

```
return 0;
```

제공된 리소스를 이용해서 체스게임을 완성하자.