

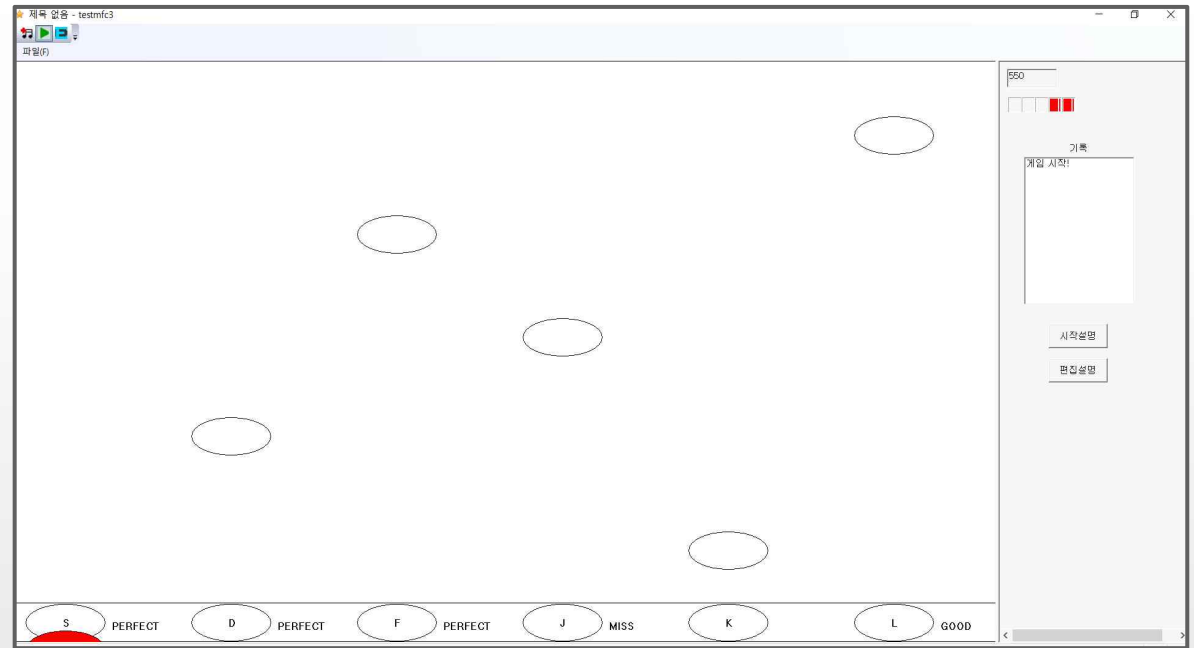
프로그램 개요

MFC를 이용한 리듬게임

- wav파일과 함께 직접 리듬 노드를 만들어 플레이하는 프로그램.
- MFC 프로그래밍을 통해 이벤트 처리를 진행하였습니다.

사용언어: Visual C++

IDE : Visual Studio 2015



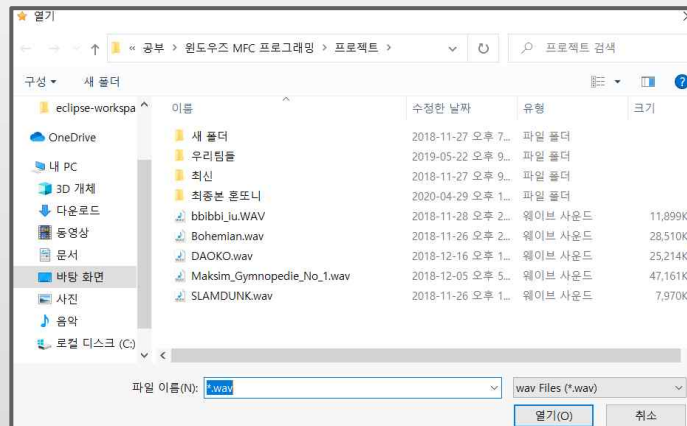
실행과정과 소스코드 소개

MFC를 이용한 리듬게임

- 분할 윈도우를 이용해 각각의 컴포넌트와 뷰를 포함시켜두었습니다.
- 노래는 PlaySound() 함수를 이용해서 .wav음악파일을 틀어줄 수 있습니다. 이때 노래가 나오면서 키보드가 동작을 하도록 비동기화된 상태로 메소드가 실행이 됩니다.

```
void Ctestmfc3View::music()
{
    PlaySound(Music_Path_1, AfxGetInstanceHandle(), SND_ASYNC); // 노래 재생 ... 비동기성
}
```

노래를 선택해두면 그에맞는 노래의 문자열을 Music_Path_1에 저장해두고 뮤직노드가 저장되어있는 .txt파일과 함께 정보를 불러옵니다.

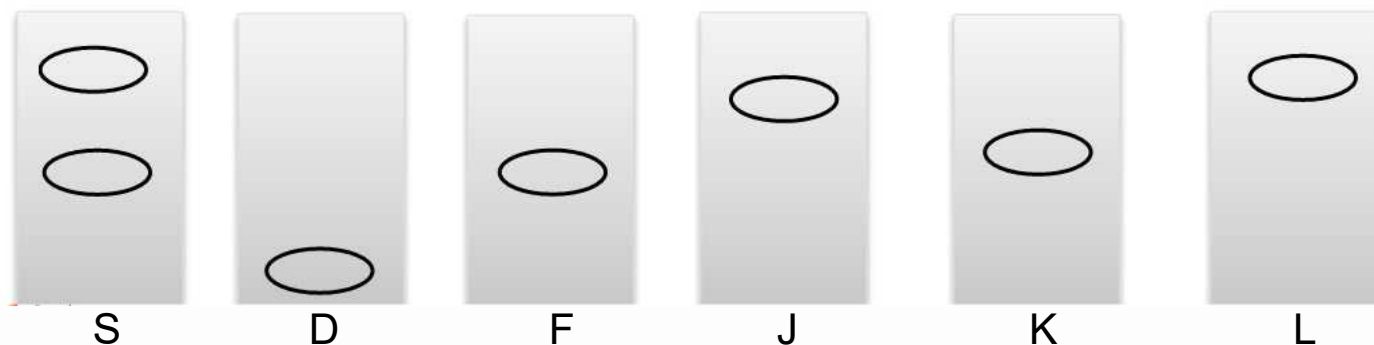


```
if (ins_dlg.DoModal() == IDOK) {
    Music_Path_1 = ins_dlg.GetPathName();

    CString a = ins_dlg.GetFileName();
    a.Append(_T(".txt"));
    pFileName = _tcscdup(a);

    AfxMessageBox(_T("노래 선택을 완료했습니다!"));
    Read_music();
}
```

실행과정과 소스코드 소개



리듬 노드는 편집기모드로 변경을 하고 듣는 노래 박자에 맞추어 키를 눌러주면 됩니다. 해당 박자에 맞는 위치 정보값들을 생성하고 이를 txt파일에 담아 둡니다.

이후, 노래를 실행했을때 해당 txt파일이 있다면 정보를 얻어와 노드를 만들어줍니다.

게임을 시작하면 노드를 미리 만들어 둔 상태에서 위치 값을 하나씩 증가시키며 노드가 내려오는 듯한 연출을 만들 수 있습니다. 주기적으로 위치를 이동하며 해당 노드를 이동을 표현해줍니다.

```
while (true)
{
    flow += 6;

    sred_flow += 6; sblue_flow += 6;
    dred_flow += 6; dblue_flow += 6;
    fred_flow += 6; fblue_flow += 6;
    jred_flow += 6; jblue_flow += 6;
    kred_flow += 6; kblue_flow += 6;
    lred_flow += 6; lblue_flow += 6;

    Invalidate(false);
    Wait(1);

    if (end == true)
    {
        PlaySound(NULL, AfxGetInstanceHandle(), NULL);
        break;
    }
    else if (mode == 0 && flow >= MaxLength()+300) {
        PlaySound(NULL, AfxGetInstanceHandle(), NULL);
        finish = true;
        break;
    }
}
```

실행과정과 소스코드 소개

노드의 이동을 표현하기 위해서는 이전 노드는 지우고 그 다음 위치의 노드를 그려주어야 합니다. 노드를 그려주었다 지웠다는 반복하는 과정을 하다보면 부드럽지 않게 끊기는 현상이 나타나게 됩니다. 그렇기에 **더블 버퍼링** 기법을 사용하도록 했습니다.

2개의 CBitmap 클래스의 객체를 이용해서 하나의 객체로 현재 나타낼 그림을 보여주고 있다면, 다른 객체로는 다음에 그려질 그림을 그려둡니다.

그 다음 그림을 보여주어야 할때 그림을 지워서 다시 교체하는 작업을 하지않고 곧바로 만들어둔 객체를 통해 다음 그림을 이어서 보여줍니다. 그렇게 한다면 부드러운 동작의 그림을 표현할 수 있습니다.

```
void Ctestmfc3View::OnDraw(CDC* pDC)
{
    CRect rect;
    CDC dubleBfDC;
    CBitmap *pOldBitmap, bitmap;

    GetClientRect(&rect);

    dubleBfDC.CreateCompatibleDC(pDC);
    bitmap.CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());

    pOldBitmap = dubleBfDC.SelectObject(&bitmap);
    dubleBfDC.PatBlt(0, 0, rect.Width(), rect.Height(), WHITENESS);

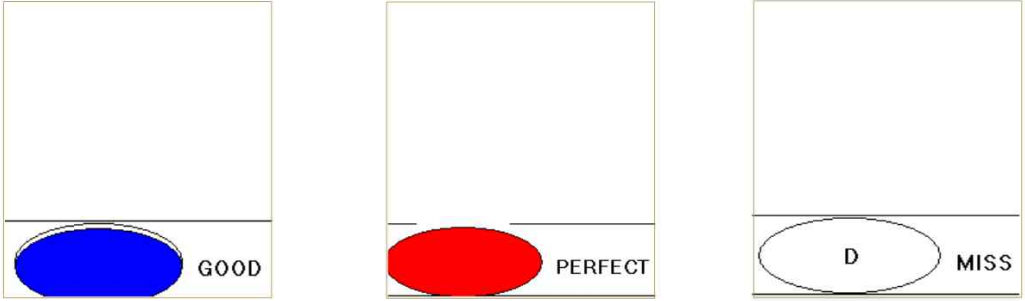
    ImageSet(&dubleBfDC);

    pDC->BitBlt(0, 0, rect.Width(), rect.Height(), &dubleBfDC, 0, 0, SRCCOPY);

    dubleBfDC.SelectObject(pOldBitmap);
    dubleBfDC.DeleteDC();
    bitmap.DeleteObject();
}
```

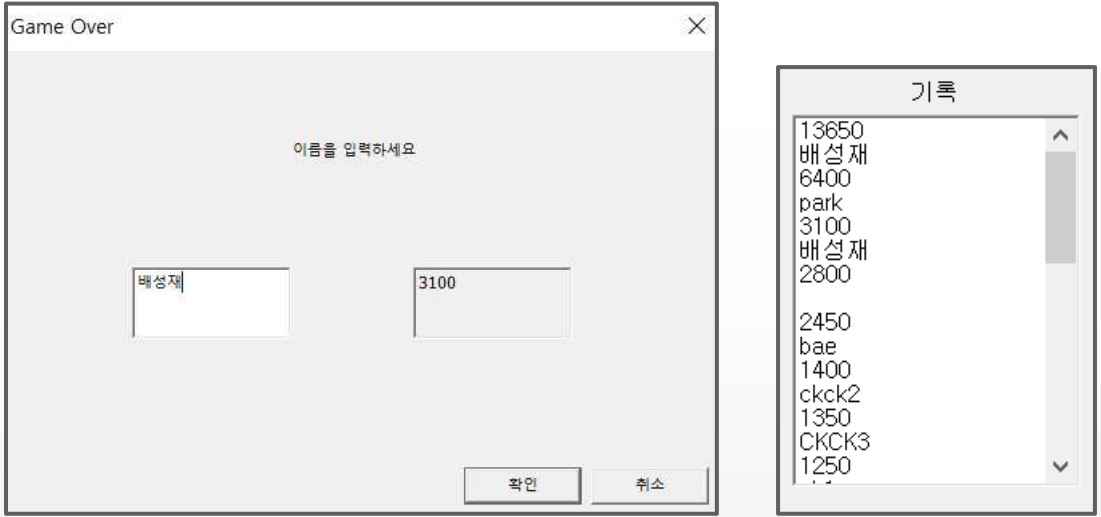
더블 버퍼링을 이용한 OnDraw() 함수

실행과정과 소스코드 소개



게임 진행중 알맞은 위치에 노드가 놓여질 때 키를 누르게 되면 해당 판정이 잘했는지 못했는지 판독을 해줍니다.

현재 노드의 위치와 히트판정 타원의 위치사이의 차이를 표현해서 노드 색을 바꾸어주고 점수를 부여해줍니다.



게임을 전부 클리어 했거나 게임오버가 되었을 때 지금까지 획득한 점수를 보여주며 이름을 올릴 수 있습니다. 기록화면에 보면 지금까지의 랭킹을 보여줍니다.

```
diff = (grade[0][i] + grade[1][i]) / 2;
```

노드의 가운데 위치값

```
if (( diff-60 <= flow) && ( flow <= diff+60) ) {
```

클릭했을 때 Miss인지

```
if (( diff-50 <= flow && flow <= diff-20 )) {
```

클릭했을 때 Good인지 Perfecct인지

실행화면 전체

