2009 종합설계 최종보고서

이미지 분석을 응용한 악보인식과 미디어 제작

<Team. Jugend>

제출자 이름	이탁곤
제출자 학번	52067386
팀원 이름	김도균 (52067372) 오원균 (52067383) 함충민 (52001504)
담당교수	정용주 교수님
제출일시	
평 가	

'2009년도 전기 컴퓨터과학전공 졸업작품

확 인	점 수

보 고 서

제 목 : 이미지 분석을 응용한 악보인식과 미디어 제작

심 사 평	

2009년도 전기 컴퓨터과학전공 졸업작품

목 차

I. 설계 제안

I.1. 시스템 개요

본 프로젝트에서 제안한 시스템은 이미지 파일로 작성된 악보를 인식, 변환해 연주할 수 있는 프로그램이다. 인식되어 분석된 악보는 전자 악기 디지털 인터페이스(이하 MIDI)[1]로 저장된다. 사용자는 시스템이 분석한 악보를 이용해 자신만의 음악을 제작해 사용할 수 있다.

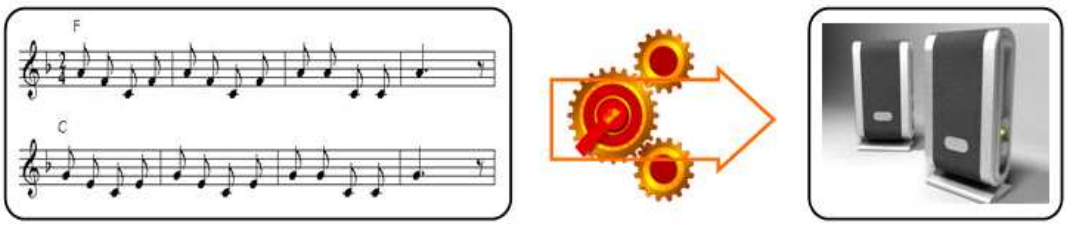


그림 1-1. 개념 흐름도

1. 이미지 파일로 작성된 악보를 인식한다.
2. 인식한 이미지 파일을 분석해 음원으로 변환한다.
3. 변환된 음원을 사용자의 용도에 맞춰 사용한다.

I.2. 주제 선정 과정

I.2.1.선정 동기

하루가 다르게 발전하는 기술들은 우리의 삶을 빠르게 바꿔놓고 있다. 특히 21세기의 가장 큰 화두이자 IT강국인 대한민국에서는 그 발전 속도가 경이로울 정도이다. 인터넷이 대중화되어 생활에 파고든 것은 그리 오래된 일이 아님에도 불구하고 인터넷과 이를 사용하는 우리의 모습은 빠르게 변화하고 있다.

1) MIDI : 정식명칭은 Musical Instrument Digital Interface. 전자악기의 연주정보 등을 상호 전달하기 위해 정해진 H/W 및 통신프로토콜의 국제적 표준 규격이다.

과거 인터넷 사용자가 필요한 것을 “찾기위해” 검색 서비스(Search Engine)등을 이용했다면, 현재의 사용자들은 자신만의 정보, 노하우, 감상 등을 콘텐츠[2]화 시켜 공개, 다른 사용자들과 공유한다. 인터넷에서 정보를 찾는 것에 만족하지 않고 스스로가 콘텐츠를 “만들어” 정보의 바다라고 불리는 인터넷을 더욱 풍성하게 하는 것이다. 즉 오늘날의 인터넷 사용자는 정보를 이용, 열람하는데 만족하지 않고 적극적으로 자신을 표현하고 있는 것이다. 우리들에게 친숙하게 자리 잡고 있는 UCC[3],블로그[4], 미니홈피[5] 등이 좋은 예이다. 이러한 흐름에 따라 현재의 인터넷 사용자들은 온라인상에서도 자신의 개성을 찾는다. 일률적인 것들에서 벗어나 “자신만의 콘텐츠”를 만드는데 관심을 갖게 된 것이다. 이러한 흐름에 기인해 사용자가 자신만의 콘텐츠를 만들 수 있는 시스템을 제안한다.

I.2.2.유사 시스템 소개

프로젝트의 요구사항, 제약사항 등을 분석하기 위해서는 이전에 개발되어 상용화된, 이른바 성공적인 프로젝트를 조사해 보는 것도 하나의 방법일 것이다. 개발에 앞서 유사한 시스템을 조사해 분석한다.

우선 “자신만의 콘텐츠를 제작한다”는 측면에서는 비교적 많은 소프트웨어가 개발되어 배포되고 있다. 대표적인 것이 인터넷 UCC 공유 사이트인 엠엔캐스트(http://www.mncast.com)와 제휴되어 있는 Magic one이다. SM온라인에서 개발한 Magic one은 다양한 동영상을 간편한 조작으로 비교적 자유롭게 편집할 수 있는 기능을 제공하며 제작한 동영상을 엠엔캐스트로 업로드 할 수 있는 기능을 제공한다. 본 프로젝트에서 다루는 콘텐츠 및 개념과는 다소 차이점이 있지만 사용자에게 “자신만의 콘텐츠” 제작을 돕는다는 면에서는 유사한 시스템으로 볼 수 있을 것이다.

---

2) 콘텐츠(Contents) : 인터넷이나 컴퓨터 통신 등을 통하여 제공되는 각종 정보나 그 내용물.
3) UCC : User Created Contents의 줄임말. 사용자가 직접 제작한 콘텐츠를 지칭한다.
4) 블로그(Blog) : 사용자가 자신의 관심사에 따라 자유롭게 글을 올릴 수 있는 웹 사이트. web과 log의 합성어.
5) 미니홈피 : SK커뮤니케이션즈가 운영하는 싸이월드(http://cyworld.com)에서 제공하는 개인 홈페이지 서비스.

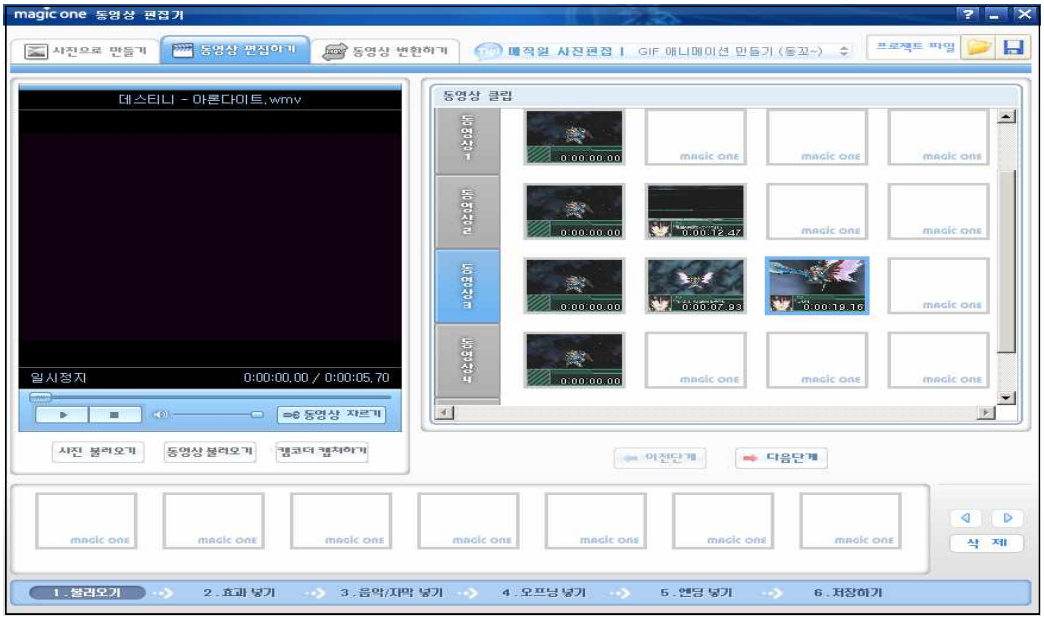


그림 1-2. Magic one

한편 “이미지 파일을 인식 및 분석”한다는 측면에서는 현재 NHN의 대표 검색엔진인 네이버(Naver)에서 제공 중인 이미지 문서 인식 OCR을 들 수 있다. 이는 사용자가 올린 이미지를 분석, 글자와 숫자로 구분해 사용자가 바로 편집할 수 있는 텍스트로 만들어주는 소프트웨어이다.

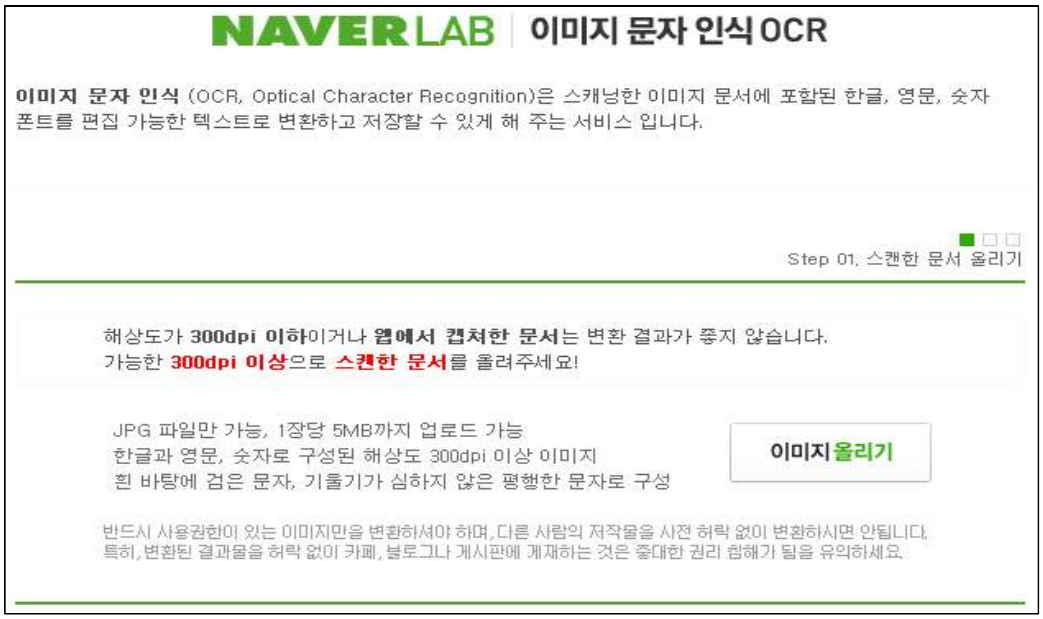


그림 1-3. 네이버 이미지 인식

I.2.3.주제 선정을 위한 회의 내용

주제 선정은 각 팀원이 프로젝트를 제안, 자체 프레젠테이션을 통해 선정했다. 주제 선정과장에서 가장 중요시된 것은 “현재의 트렌드에 맞는 것인가?”라는 요소와 “프로젝트 실현 가능성 여부”였다. 다양한 프로젝트가 제안된 가운데 “사용자의 개성을 살린 콘텐츠 제작”이라는 컨셉과 적용되는 “이미지 분석 기술”이 매력적으로 받아들여져 주제로 선정됐다.

I.3. 시스템의 목적

제안된 시스템은 사용자가 이미지 형태의 파일형식이나 음원 형태의 파일형식에 대한 지식을 갖고 있지 않아도 손쉽게 악보 이미지를 음원으로 변환해준다. 변환된 음원에 대한 편집 기능도 제공, 사용자의 기호에 맞는 콘텐츠를 생성을 돕는다.

I.4. 시스템의 기능

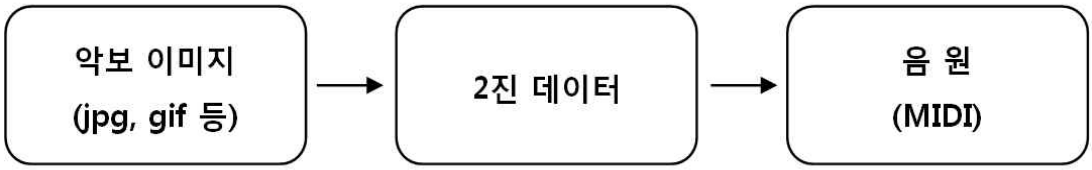


그림 1-4. 데이터 흐름도

시스템은 이미지 파일로 작성된 악보를 분석해 디지털화, 즉 2진 데이터로 변환한다. 변환된 데이터는 패턴 등을 분석해 미리 정해진 음원 데이터와의 매칭과정을 거쳐 MIDI로 변환된다. 변환된 MIDI파일은 사용자의 기호에 따라 다른 악기효과를 낼 수 있으며 볼륨도 변경할 수 있다. 이 기능을 이용해 사용자는 자신만의 콘텐츠를 만들어낼 수 있다. 작성된

MIDI파일은 프로그램에서 재생 및 편집할 수 있으며 사용자가 콘텐츠를 저장, 보관할 수 있도록 돕는다.

변환 대상으로서 MIDI를 선택한 이유는 다음과 같다.

1. MIDI의 플랫폼은 컴퓨터가 사용하는 운영체제(OS)에 독립적이다. 사용을 위해 드라이버 등의 설치과정을 거치지 않더라도 다양한 시스템에서 자유롭게 사용할 수 있다.
2. 확장성이 매우 우수하며 압축률이 높아 다른 음원 데이터와 비교해 작은 용량이다. 이로 인해 웹이 도입되던 시기부터 널리 사용되어 현재에도 대다수의 웹사이트에서 널리 사용되고 있다.
3. 디지털로 변환된 자료의 수정으로 비교적 손쉽게 음원을 수정할 수 있다.

스킨 및 이미지 인식 과정은 다음과 같다.

1. 악보 이미지 전체를 인식한 뒤 이진화 작업을 거친다. 이진화 작업은 분석률을 높이기 위한 준비단계라고 할 수 있다.
2. 수평 히스토그램을 기준으로 각 소절별을 나눈다.
3. 수직 히스토그램을 기준으로 음표의 위치를 찾는다.

1~3. 과정을 통해 얻은 음의 위치 및 높낮이 정보를 조합해 각 음원과 매칭, MIDI파일로 변환한다. 요약하자면 제안된 시스템은 “이미지 분석”과 “MIDI파일 제작”이라는 두 개의 모듈로 나뉜다.

I.5. 시스템 응용가능 분야

본 시스템은 기본적으로 사용자가 전문적인 지식을 갖추고 있지 않더라도 손쉽게 자신만의 콘텐츠를 자유롭게 만들어 내는 것을 돕는다. 사용자의 이러한 활동은 최근 활발하게 대두되고 있는 웹2.0(web2.0)[6]의 초석이라고 할 수 있다.

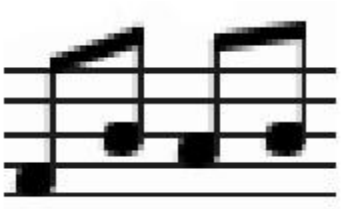
6) 웹2.0(Web2.0) : 데이터의 소유자나 독점자 없이 누구나 손쉽게 데이터를 생산하고 인터넷에서 공유할 수 있도록 한 사용자 참여 중심의 인터넷 환경.

한편 시스템의 중심축이라고도 할 수 있는 이미지 분석 기술 역시 다양한 분야에 활용할 수 있다. 컴퓨터는 사람과 달리 모든 데이터를 0과 1로 처리한다. 사람이 눈으로 보고 이해하는 이미지나 동영상 역시 컴퓨터는 수많은 0과 1의 나열로 처리한다. 이는 각 미디어 나름의 압축방식과 처리방식이 있기 때문에 가능한 것이다. 이미지 분석기술의 핵심은 이미지를 분석하는 것에 멈추지 않고 이미지를 "인식"해 다른 형태의 미디어로 "변환"시킬 수 있다는 것이다. 이러한 기술은 사진으로만 남아 있는 문서를 디지털화 하거나 타블렛 등의 장비와 연동해 전자문서를 제작 할 수 있는 등 다방면으로 활용될 수 있다. 흔히 볼 수 있는 바코드 인식 역시 이 기술을 기반으로 한 것이다.

I.6. 예상 문제점

제안 과정에서 시스템의 컨셉 및 흐름이 결정된 이후 제시된 문제점은 다음과 같다. 문제점은 크게 기능상 문제와 현실적 제약문제로 나눌 수 있다.

I.6.1. 기능상 문제점



제안된 알고리즘을 이용해 왼쪽 그림과 같은 이미지를 분석할 경우 시스템은 수직, 수평 히스토그램을 기준으로 음의 머리부분을 찾는다. 시스템은 음의 위치와 높낮이를 인식하지만 연음이나 이음줄 등은 인식할 수 없다.

한편 악보는 음표 이외에도 다양한 기호가 사용된다. 제안된 알고리즘은 패턴 매칭이 아닌, 히스토그램 분석을 사용하기 때문에 문자 등의 기호로 작성된 기호를 인식할 수 없다. 또 인식여부를 떠나 크레센도(점점세게를 뜻하는 음악기호)등의 기호를 MIDI에서 이러한 것을 표현하는 것은 매우 어렵다.

또 예시와 달리 복잡한 음표라면 인식률이 크게 떨어질 수 있다는 문제점을 내포하고 있다. 인식률이 떨어진다는 것은 결국 악보 이미지와 분석 후 작성된 MIDI간 격차가 크다는 것이다.

I.6.2. 현실적 제약

음악이라는 콘텐츠는 대중에게 생소한 것이 아닌, 이미 우리생활에 뿌리 깊게 관련된 콘텐츠이다. 때문에 사용자의 기호에 맞는 MIDI를 생산해낼 수 있는 본 시스템은 사용에 따라 큰 효과를 기대할 수 있다. 하지만 음원이라는 미디어는 그만큼 저작권 등의 문제에 많이 노출되어 있다. 예를 들어 악보를 구입한다고 해도 악보 구입으로서 얻는 소유권은 음원 사용과는 별개이다. 즉 악보로 음원을 만들어내는 것은 결코 저작권에서 자유로울 수 없는 것이다. 이러한 문제해결을 위해서는 음원 제공업체 등과 마찬가지로 저작권자와의 계약을 필요로 한다. 결국 기능대비 운영비가 비대해질 가능성을 내포하고 있는 것이다.

I.7. 팀 구성

팀 구성은 4명으로 이루어져있다. 구성원 간 관계가 수평적이기 때문에 기본적으로는 분산형 팀 구성 방식을 채택한다. 다만 과반수가 편입생으로 구성된 팀이기 때문에 팀원들 간 프로젝트 경험이 상이함에 따라 부분적으로는 계층형 팀 구성 방식을 차용, 결과적으로는 혼합방식에 가까운 팀 구성 형태를 띠고 있다. 팀 구성 상세는 다음과 같다.

1. 팀 명 : Jugend
2. 팀 원 : 4명
3. 구성원 : 이탁곤(팀장), 함충민, 오원균, 김도균
4. 담당업무 :

이탁곤 – 프로젝트 진행지휘 및 조율, 프로그램 구현(MIDI 제작)
함충민 – 자료 조사, 프로그램 구현(이미지 분석), 테스트
오원균 – 진행상황 기록, 자료조사, 프로그램 구현(MIDI 제작), 테스트
김도균 – 프로그램 구현(이미지 분석), 테스트

I.8. 프로젝트 일정

프로젝트 진행 일정은 각 작업의 구성과 진행일정을 직관적으로 판단할 수 있는 간트 차트(Gantt Chart)를 이용해 기술한다.

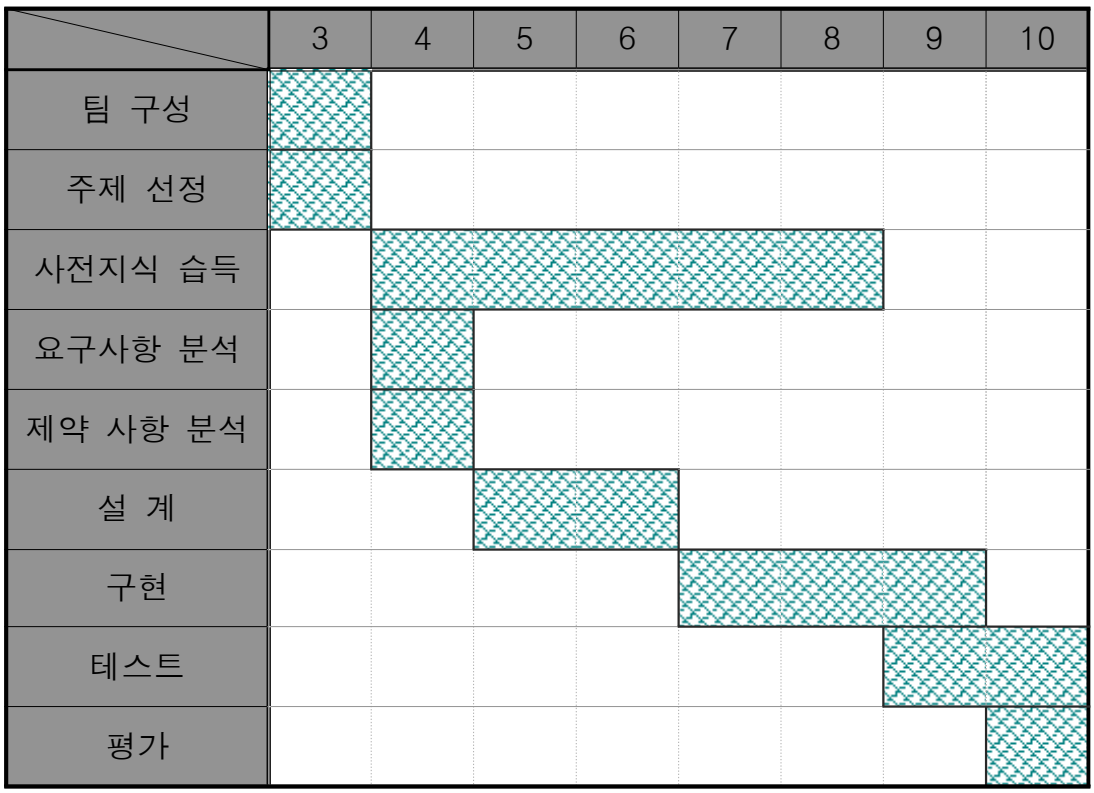


표 1-1. 프로젝트 진행 일정

II. 기초 설계

II.1. 제안 시스템의 입출력 정의

본 프로젝트에서 제안된 시스템은 악보 이미지를 분석해 MIDI로 변환, 사용자만의 미디어 콘텐츠를 제작할 수 있는 소프트웨어를 목표로 한다.

시스템은 크게 “악보 이미지 분석”, “MIDI 변환”으로 나뉜다. 사용자는 자신이 음원으로 변환할 악보 이미지를 선택하며 시스템은 사용자가 선택한 악보 이미지를 MIDI로 변환시켜준다.

그림 2-1. 입출력 형태

입력되는 데이터는 사용자가 선택한 비트맵 형식 이미지[7]이며 시스템의 처리과정을 거친 이후 출력되는 것은 MIDI형식의 미디어 파일이다. 여기서 알 수 있듯이 입력되는 데이터와 처리 후 만들어지는 데이터 형식이 서로 다르다. 두 매체는 “음악”이라는 매체에 대해 약속된 기호로 표기된 시각적 매체, 음원적 매체라는 차이만 있을 뿐 실제로는 같은 것을 표현하고 있다. 프로젝트에서 제안된 것은 “다른 방법으로 표현된 매체를 서로 이어주는 것”이라고 할 수 있겠다. 제안된 시스템의 입력요소는 “비트맵 형식의 이미지(.jpeg등)”, 출력요소는 “MIDI형식의 음원(.mid)”이라고 정의할 수 있다.

II.2. 요구사항 분석

본 프로젝트는 사용자가 선택한 비트맵 이미지 형식으로 작성된 악보를

7) 이미지 파일은 크게 비트맵(bitmap) 형식과 벡터(vector) 형식으로 나뉜다. 각 형식에 따라 특징이 있지만 일반적으로 널리 사용되는 .bmp, .jpg 형식은 비트맵 방식을 사용한 이미지 형식이다. 본 시스템에서는 비트맵 이미지 파일 분석을 전제로 한다.

MIDI파일로 변환해줌으로서 사용자만의 미디어 콘텐츠를 만들어 줄 것을 목표로 하고 있다. 시스템은 크게 “이미지 분석”과 “MIDI파일 제작”으로 나눌 수 있다. 요구사항은 각 파트별로 다르게 제시된다. 파트별 요구사항은 다음과 같다.

1) 이미지 분석
- 비트맵 형식으로 작성된 이미지 파일을 읽을 수 있다.
- 악보 이미지를 분석할 수 있다.
- 사용자는 분석에 대한 상세한 과정을 몰라도 된다.
- 분석결과를 사용자가 열람할 수 있다.

2) MIDI파일 제작
- 분석된 비트맵 이미지가 MIDI형식으로 변환된다.
- 사용자는 작성된 MIDI파일을 감상할 수 있다.
- 사용자는 작성된 MIDI파일을 편집할 수 있다(볼륨 및 사용악기).
- 사용자는 편집한 MIDI파일을 저장할 수 있다.

II.3. 기능 사양

시스템의 개요는 I.1.항을, 시스템에 대한 요구사항은 II.2.항을 참조하면 확인할 수 있다. 시스템이 사용자에게 제공할 서비스 역시 II.2.항에 제시된 요구사항에 기인하며 완성된 시스템은 모든 요구사항을 만족해야 할 것이다.

제안된 시스템은 사용자 개개인을 위한 시스템이다. 목표가 사용자 개개인을 위한, 사용자만의 미디어 파일을 제공하는 것인 만큼 사용자가 원하는 정확한 악보 이미지 분석과 사용자가 원하는 스타일로의 편집 기능을 제공해야 한다. 이 모든 기능은 사용자가 손쉽게 사용할 수 있도록 추상적이지 않고 간결하게 제시되어야 한다.

사용자는 비트맵 이미지 형식의 악보가 MIDI형식의 미디어 파일로 변환

되는 과정을 알 필요가 없다. 오로지 시스템에서 제공하는 인터페이스만으로 자신이 원하는 결과를 확인하기만 하면 되는 것이다. 사용자는 오직 자신이 원하는 이미지를 선택하며 시스템이 변환한 MIDI파일을 확인 및 편집한다.

II.4. 요소 시스템 분석

시스템의 요소 시스템 분석은 유즈케이스 모델링(Usecase Modeling)을 이용한다. 유즈케이스 모델링 방법은 다음과 같다.

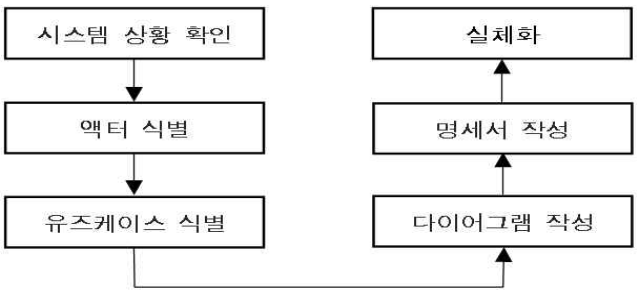


그림 2-2. 유즈케이스 모델링 과정

II.4.1. 시스템 상황 확인

사용자가 악보 이미지를 분석, 변환하기 위해서는 해당 악보 이미지를 선택해야 한다. 선택한 악보이미지는 시스템에서도 확인할 수 있다.

MIDI파일 재생 및 편집은 사용자가 악보 이미지를 선택해 분석해야만 가능하다. 분석과정을 거쳐 MIDI파일이 작성되지 않았다면 MIDI파일의 실행 및 편집, 저장은 불가능하다.

II.4.2. 액터 식별

액터(Actor)는 시스템에 관련된 행위자를 지칭한다. 본 시스템은 사용자 개개인을 위한 시스템이기 때문에 사용자를 제외한 행위자는 존

재하지 않는다.

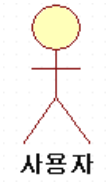


그림 2-3. 액터 식별 결과

II.4.3. 유즈케이스 식별

유즈케이스(Usecase)는 특정 목적의 관점에서 볼 때의 쓰임새와 시스템의 특성 및 동작을 지칭한다. 사용자 요구사항(II.2항) 및 기능 사양서(II.3항)를 기본으로 유즈케이스를 식별하면 다음과 같다.

그림 2-4. 유즈케이스 식별 결과

식별한 유즈케이스의 기능별 범주는 다음과 같다.

사용자	기능 범주	기능 (유즈케이스)
사용자	외부파일(악보이미지) 호출	악보 이미지 열기
	악보 이미지 분석	악보 이미지 분석
	MIDI 파일 제어	재생
		일시정지
		정지
	MIDI 파일 편집	연주악기 수정
		볼륨 수정
		소절 선택
	외부파일(미디어파일) 저장	저장

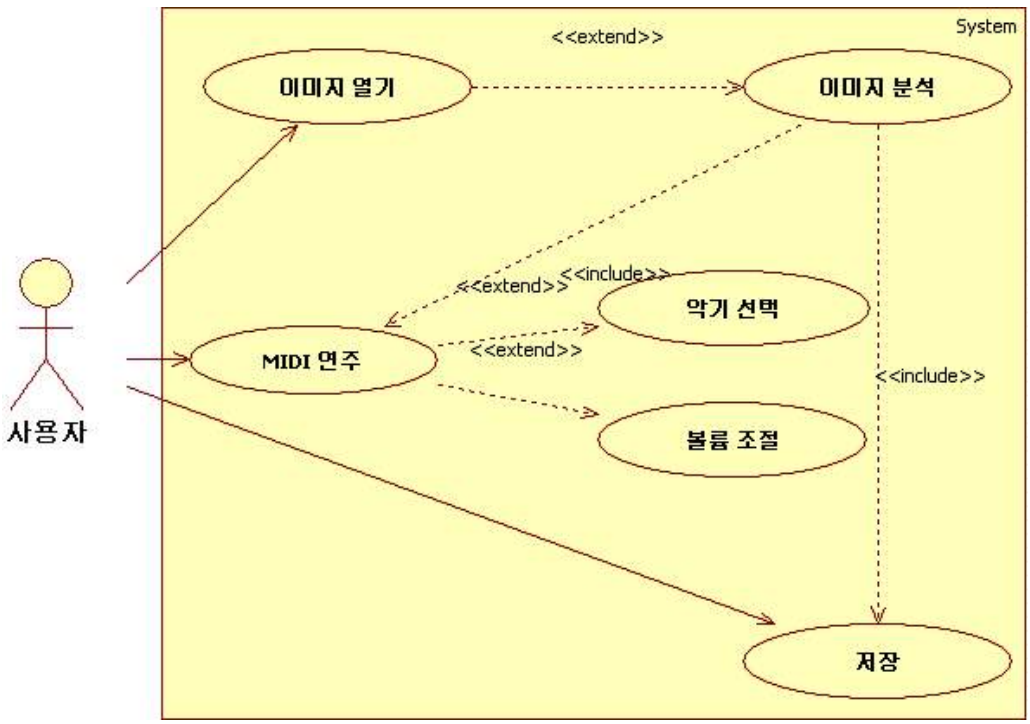
표 2-1. 유즈케이스의 기능별 범주

Ⅲ. 시스템 설계

Ⅲ.1. 요소 시스템 설계

Ⅲ.1.1. 유즈케이스 다이어그램

Ⅱ.4.2항에서 식별한 액터와 Ⅱ.4.3항에서 식별한 유즈케이스를 평가한 뒤 각 유즈케이스 간 의존성을 평가, 관계를 설정한다.



Ⅲ.1.2. 유즈케이스 명세서

❖ 악보 이미지 불러오기

유즈케이스명	악보 이미지 불러오기
액터명	사용자
개요 및 설명	사용자가 자신이 원하는 악보 이미지를 호출한다.
사전 조건	없음.
이벤트 흐름	사용자가 원하는 악보 이미지가 호출된다.

❖ 재생

유즈케이스명	재생
액터명	사용자
개요 및 설명	변환된 데이터를 재생한다.
사전 조건	재생하고자 하는 음악의 악보를 호출한 상태여야 한다.
예상 외 흐름	재생할 악보가 선택되어 있지 않은 경우 “선택하신 악보가 없습니다.“라는 메시지를 출력한다.

❖ 저장

유즈케이스명	저장
액터명	사용자
개요 및 설명	변환된 데이터를 MIDI파일로 저장한다.
사전 조건	사용자가 선택한 악보 이미지가 MIDI형식으로 변환되어 있어야 한다.

❖ 악보 이미지 분석

유즈케이스명	악보 이미지 분석
액터명	사용자
개요 및 설명	사용자가 선택한 악보 이미지를 분석하여 음계의 위치와 길이를 파악해 데이터화 한다.
사전 조건	사용자기 재생할 음악의 악보를 호출한 상태여야 한다.
예상 외 흐름	분석한 이미지가 악보가 아닐 경우 사용자에게 알린다.

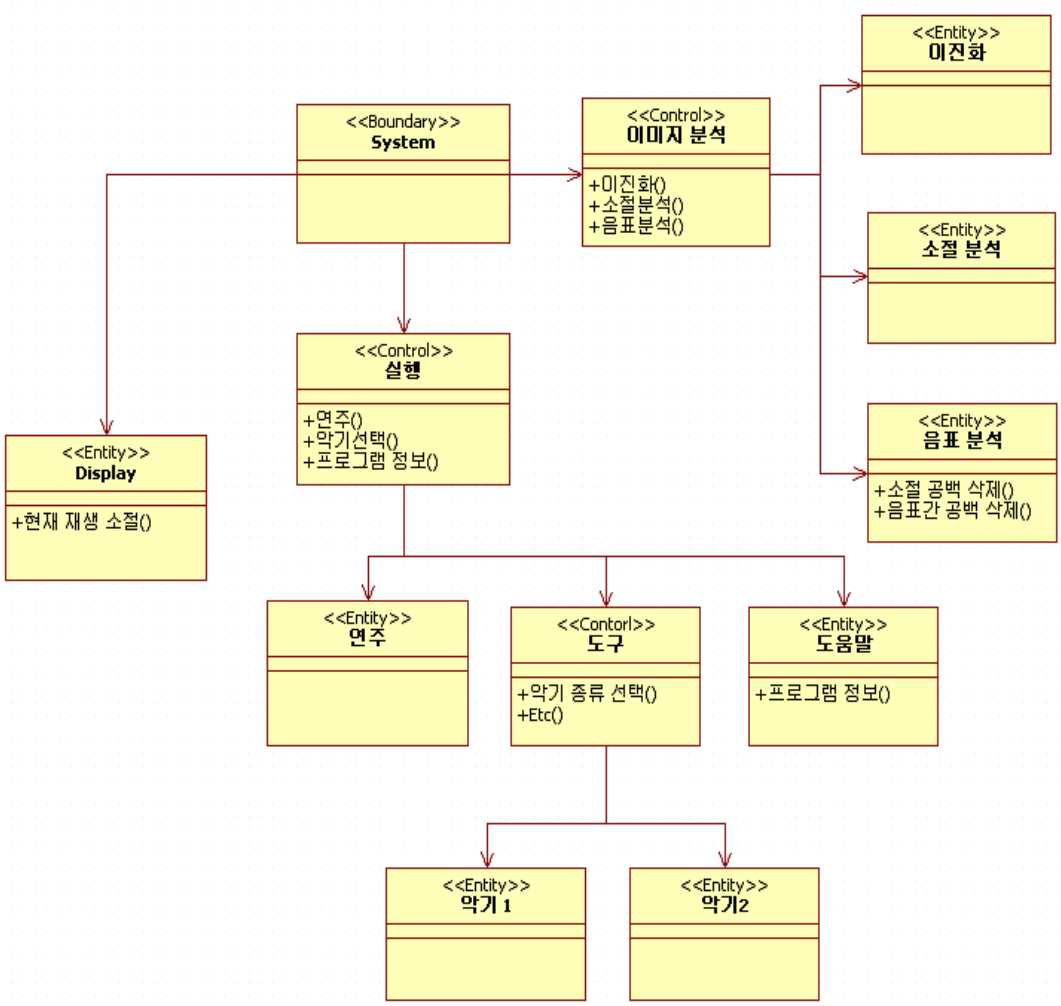
❖ 연주악기 수정

유즈케이스명	연주악기 수정
액터명	사용자
개요 및 설명	재생될 음형을 선택한다.
사전 조건	사용자가 재생할 악보 이미지가 데이터형식으로 변환되어 있어야 한다.

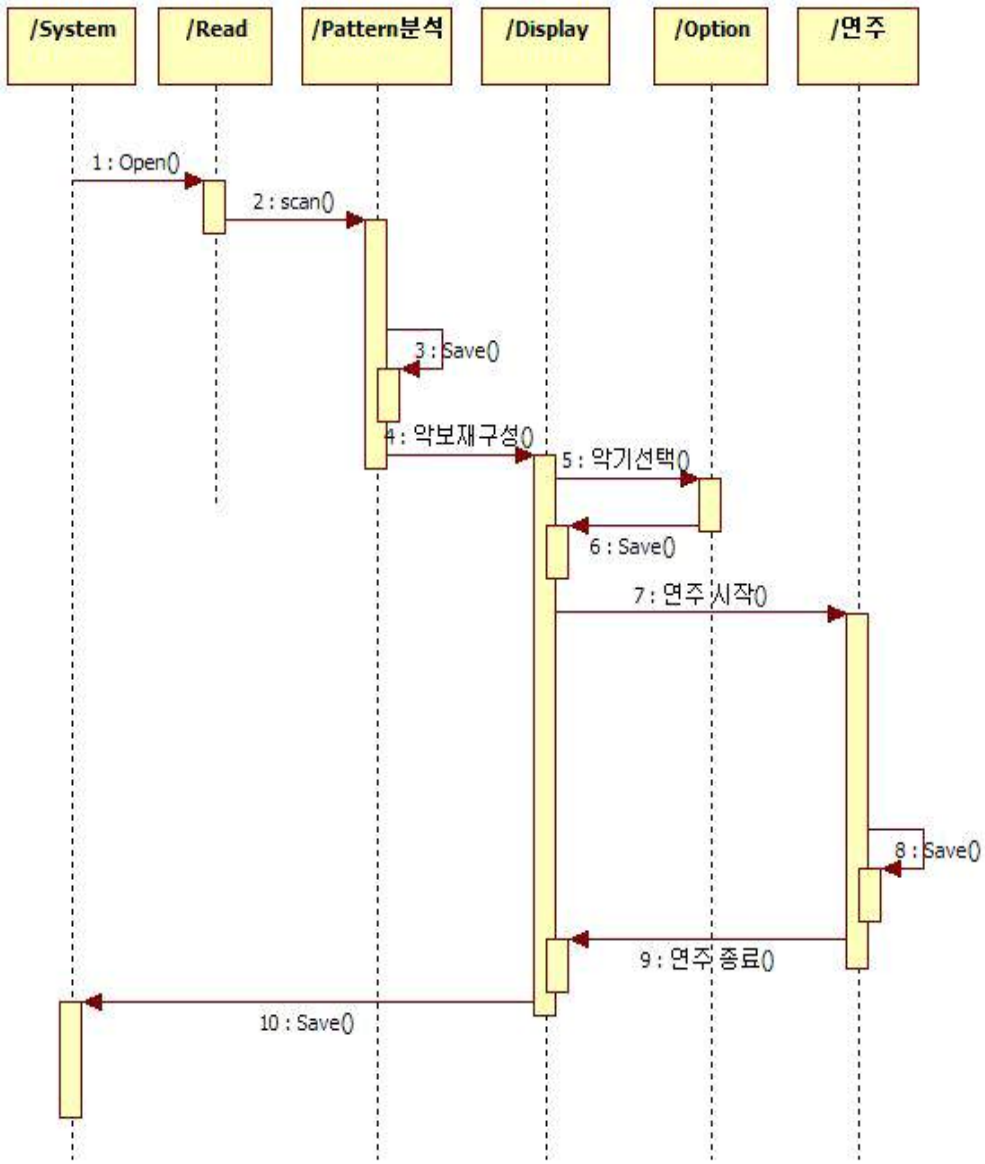
❖ 볼륨 조절

유즈케이스명	볼륨 조절
액터명	사용자
개요 및 설명	변환된 MIDI의 볼륨을 조절한다.
사전 조건	사용자가 재생할 악보 이미지가 데이터형식으로 변환되어 있어야 한다.

III.1.3. 클래스 다이어그램



Ⅲ.1.4. 시퀀스 다이어그램



III.2. 필요 요소 분석

제안된 시스템은 크게 이미지 분석, MIDI로의 변환이라는 2개의 모듈로 나눌 수 있다. 본 절에서는 각 모듈 간 필요 기술을 언급한다. 각 모듈의 성격이 크게 다르기 때문에 절로 나누어 설명한다.

III.2.1. 이미지 분석

컴퓨터를 비롯한 기계는 사람과 다르다. 사람은 연상 등을 이용해 기억하고 추측하고 생각할 수 있지만 기계는 오로지 정형화된 데이터를 이용, 분석해 저장하고 연산한다. 예를 들어 풍경이 담겨있는 사진을 떠올려보자. 우리는 사진에 담긴 풍경을 우리의 경험과 기억, 혹은 지식에 빗대어 이 사진이 어떤 풍경을 그리고 있는지 떠올린다. 이것은 매우 짧은 시간에 이루어지는 것이다. 하지만 컴퓨터라면 어떨까? 컴퓨터는 사진 전체를 분석한 뒤 저장된 데이터와 대조 작업을 거쳐 사진이 어떤 것을 담고 있는지 파악할 것이다. 이것은 사람과 비교해 매우 오랜 시간을 필요로 하며 비효율적이다. 게다가 정확도마저 100%라고 할 수 없다. 이러한 예시가 시사 하는 것은 “컴퓨터는 정확한 데이터를 전달해 주어야 한다.”는 것이다.

제안된 시스템은 악보 이미지를 분석, MIDI로 변환할 것을 목표로 하고 있다. 사람은 기본적인 악보에 대한 지식만 있다면 악보를 읽을 수 있을 것이다. 하지만 컴퓨터는 다르다. 정확하고 효율적인 방법(=알고리즘)이 제시되어야 빠르고 정확한 분석이 가능하다. 그러므로 시스템 개발에 앞서 효율적이고 정확한 방법이 제안되어야 할 것이다. 제안된 시스템이 분석해야 할 이미지는 “악보”라는 형식에 한정되어 있으므로 이에 맞춰 가장 효과적인 악보 이미지 분석 및 인식 방법이 제안되어야 할 것이다. 본 절에서는 악보 이미지 분석 및 인식 방법에 대해 기술한다.

III.2.1.a. 이미지 분석 개요

컴퓨터는 모든 데이터를 0과 1로 구성된 2진수로 처리한다. 컴퓨터의 이용분야가 넓어지면서 우리는 컴퓨터를 통해 음악, 그림, 영

상 등을 접하지만 출력 결과, 즉 사용자가 접하는 결과에 관계없이 컴퓨터는 모든 데이터를 2진수로 처리한다. 즉, 악보 이미지가 선택되면 이것을 특정한 규칙에 맞춰 0과 1로 구별할 필요가 있는 것이다.

III.2.1.b. 평균 이진화

평균 이진화는 영상처리기술에서 가장 단순한 방법으로 제안된 것이다. 구현이 단순하다는 강점이 있지만 제약조건 및 기능도 제한적이라는 단점이 있다.

제안된 시스템이 처리하는 이미지는 악보에 한정되어 있다. 악보는 규칙에 맞춰 기술된 일종의 흐름도이다. 적어도 현재까지 공용되고 있는 악보기술 규칙에 있어 색의 차이로 구분하는 기호는 없다. 또 실제로 사용되는 기호도 50개를 넘지 않는다. 때문에 평균 이진화로 악보 이미지를 분석하더라도 크게 문제가 없을 것으로 추측된다.

평균 이진화 방법을 개략적으로 기술하면 다음과 같다.

1. 선택된 이미지 전체를 스캔, RGB값을 파악한다.
2. 기준 값을 선택, RGB값이 기준 값 이상이면 흰색, 이하면 검은색으로 매핑한다.

이 과정을 거치면 시스템은 미리 선택된 기준 값을 기준으로 검은색과 흰색으로 이미지를 매핑한다.

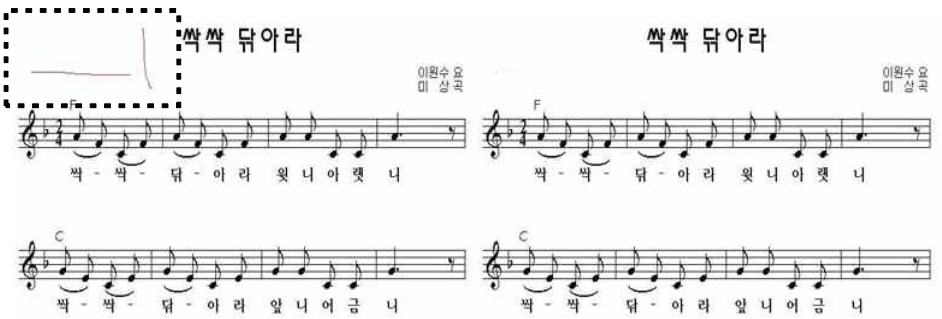


그림 3-1. 이진화 전(왼쪽)과 후(오른쪽)

그림 3-1.은 이진화 전과 이진화 후의 결과이다. 또 다음은 이진화를 구현시킨 코드이다.

```
ImageToBinarization();
    for (int y = 0; y < _bitmap.Height; y++)
    {
            for (int x = 0; x < _bitmap.Width; x++)
            {
            if (_bitmap.GetPixel(x, y).R < 126 && _bitmap.GetPixel(x,
            y).G < 126
            && _bitmap.GetPixel(x, y).B < 126)
            _bitmap.SetPixel(x, y, Color.Black);
            else
            _bitmap.SetPixel(x, y, Color.White);
    }
```

그림 3-1. 왼쪽 악보에서 점선으로 표기된 사각형 내부의 가느다란 선은 이진화 후 사라졌다. 테스트에서 기준이 된 RGB값은 126이다. 사각형 내부의 가느다란 선은 악보에 그려진 오선지나 음표와 비교해 매우 가늘다. 스캔 결과 RGB값이 기준 값인 126보다 컸기 때문에 흰색으로 매핑된 것이다. 이러한 과정을 거쳐 시스템이 제시된 이미지를 이진화한다.

III.2.1.c. 이미지 분석 및 인식 알고리즘

본 절에서는 제안된 알고리즘을 소개한다. 이해를 돕기 위해 예를 들어 설명한다. 예시에 사용된 곡은 “학교종”이다.

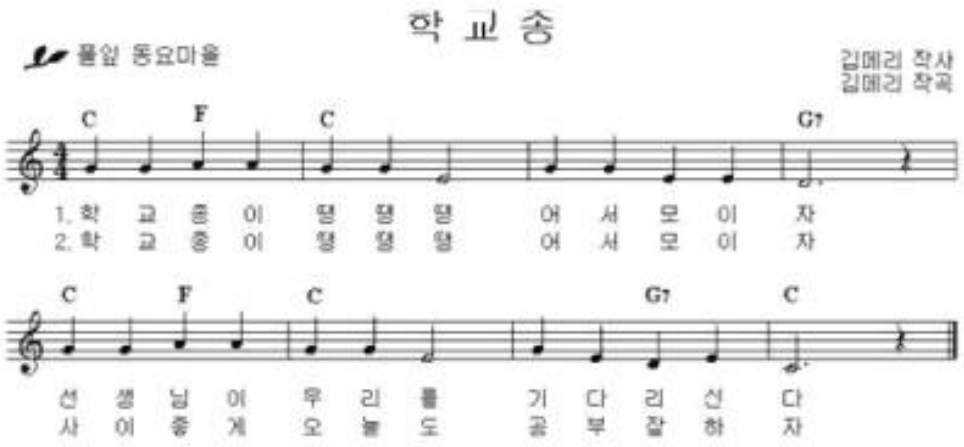


그림 3-2. 분석할 악보 이미지(학교종)

우선적으로 이미지 전체를 검사, 이진화를(III.2.1.b항 참조)를 시작한다. 이진화의 결과로 선택된 이미지가 악보인지 아닌지 판단하며, 악보가 아닐 경우 사용자에게 알린다.

이진화가 끝나면 Y축을 기준으로 이미지 전체를 스캔, 히스토그램[8]을 작성한다. 이것은 빈도를 통해 악보, 소절 등을 구분하기 위한 것이다.

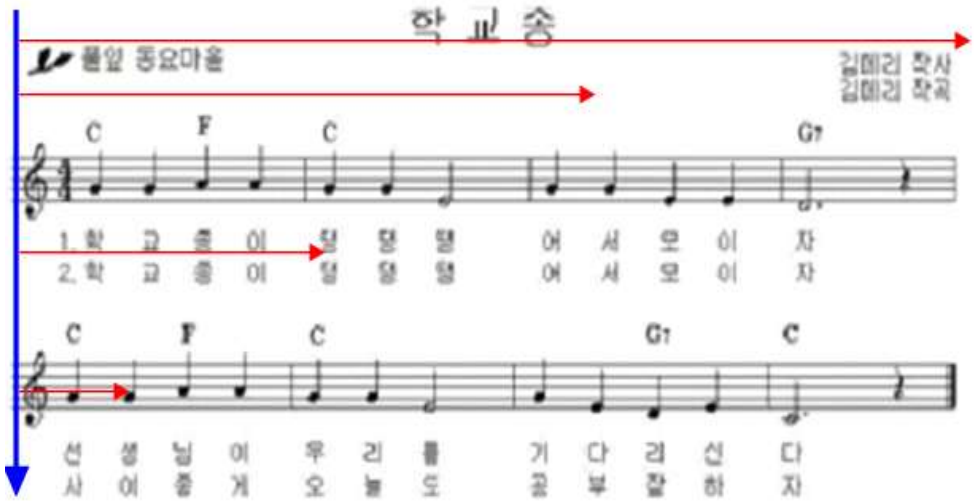


그림 3-3. 히스토그램 작성

그림3-3.과 같이 분석은 Y축을 기준으로 한다. 임의의 점 Y에서 X전체를 스캔한 뒤 Y값을 증가시킨다. 이미지 전체 탐색이 종료되면 작업을 중지하며 결과 값을 기준으로 히스토그램을 완성한다. 히스토그램 과정은 다음과 같다.

```
ImageToHistogram();
 private void ImageToHistogram()
     {
     _nHorHistogram = new int[_bitmap.Height];
     //y축을 기준으로 x축을 돌면서 값 누적하기
     for (int y = 0; y < _bitmap.Height; y++)
     {
             for (int x = 0; x < _bitmap.Width; x++)
             {
```

8) 히스토그램(histogram) : 도수분포를 나타내는 그래프로, 관측한 데이터의 분포의 특징이 한눈에 보이도록 기둥 모양으로 나타낸 것.

```
                if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).G
                == 0 && _bitmap.GetPixel(x, y).B == 0)
                _nHorHistogram[y] = _nHorHistogram[y] + 1;
                }
        }

        _bmphistogram = new Bitmap(_bitmap.Width, _bitmap.Height);
        for (int y = 0; y < _bitmap.Height; y++)
        {
                for (int x = 0; x < _bitmap.Width; x++)
                {
                        if (x < _nHorHistogram[y])
                        _bmphistogram.SetPixel(x, y, Color.Black);
                        else
                        _bmphistogram.SetPixel(x, y, Color.White);
                }

        // 수평 히스토그램 최대값 저장..

                if (_nHistogramMax < _nHorHistogram[y])
                {
                _nHistogramMax = _nHorHistogram[y];
                }
        }
}
```

그림 3-4. 히스토그램 결과

그림3-4.과 같이 작성된 히스토그램을 기준으로 소절의 수, 오선지의 위치, 간격을 찾는다. 소절 단위로 분석 및 변환되기 때문에 각 소절을 확인하기 위해, 또 음의 높이는 오선지를 기준으로 찾기 때문에 선행되는 과정이다.

소절수를 확인, 기준이 될 오선지를 확인한 뒤에는 각 음의 분석

을 위해 오선지를 삭제한다. 히스토그램의 값 중 가장 큰 값(그림 3-4.참조)을 삭제하면 오선지가 삭제된다. 오선지는 히스토그램 결과를 기준으로 위아래 값이 흰색인 경우를 찾아 삭제한다.

```
ImageToSheetDelete();
if (y == ((SheetCheck)_arraySheet[i]).Line[o])
{
for (int x = 0; x < _bitmap.Width; x++)
    {
            if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).G
            == 0 &&
            _bitmap.GetPixel(x, y).B == 0 && _bitmap.GetPixel(x, y - 1).R
            == 255 &&
            _bitmap.GetPixel(x, y - 1).G == 255 && _bitmap.GetPixel(x, y
            - 1).B == 255
            && _bitmap.GetPixel(x, y + 1).R == 255 && _bitmap.GetPixel(x,
            y + 1).G ==
            255 && _bitmap.GetPixel(x, y + 1).B == 255)
                    {
                            _bitmap.SetPixel(x, y, Color.White);
                    }
            }
 }
```

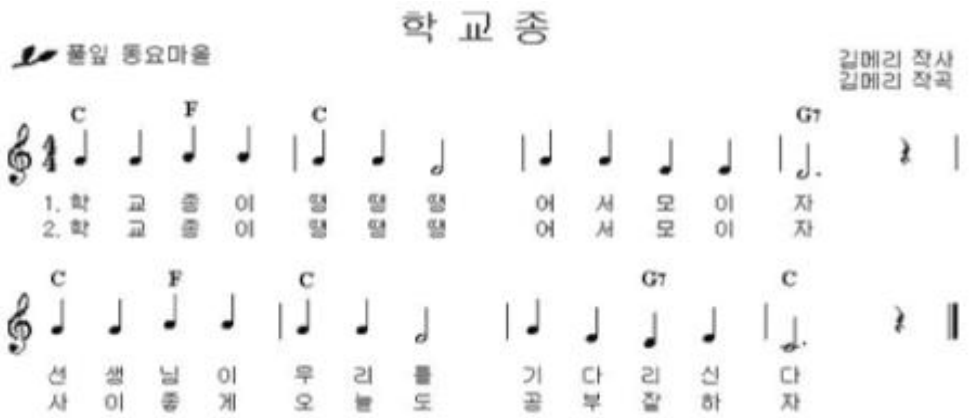


그림 3-4. 오선지 삭제

이후 각 소절별로 X축을 기준으로 탐색, 히스토그램을 작성한다. 이 히스토그램 결과를 토대로 음의 개수를 파악, 공백을 삭제해 각 음 단위로 잘라낸다. 과정과 결과물은 다음과 같다.

```
    private void ImageToNoteCut()
    {
        Bitmap[] bmpNote = new Bitmap[_noteInformation.TotalCount];
        int osun_width = 0;
        int osun_height = 0;
        int osun_count = 0;
        int note_count = 0;

        for (int i = 0; i < bmpNote.Length; i++)
        {
        osun_height = _sheetInformation[osun_count].Downs.Y - _sheetInformation[osun_count]
        .Tops.Y;
        osun_width = _noteInformation.Downs[i].X - _noteInformation.Tops[i].X;
        bmpNote[i] = new Bitmap(osun_width, osun_height);

        _noteInformation.Tops[i] = new Point(_noteInformation.Tops[i].X, _sheetInformation
        [osun_count].Tops.Y);
        _noteInformation.Downs[i] = new Point(_noteInformation.Downs[i].X, _sheetInformation
        [osun_count].Downs.Y);

        for (int y = _sheetInformation[osun_count].Tops.Y; y <_sheetInformation[osun_count]
        .Downs.Y; y++)
        {
                for (int x = _noteInformation.Tops[i].X + 1; x < _noteInformation.Downs[i]
                .X; x++)
                {
                if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).B == 0 && _bit
                map.GetPixel(x, y).G == 0)
                {
                        bmpNote[i].SetPixel(x - _noteInformation.Tops[i].X, y -
                        _sheetInformation[osun_count].Tops.Y, Color.Black);
            }
            else
            {
                bmpNote[i].SetPixel(x - _noteInformation.Tops[i].X, y - _sheetInformation
                [osun_count].Tops.Y, Color.White);
            }
        }
      }

      note_count++;
      if (note_count == _sheetInformation[osun_count].SheetCount)
      {
          note_count = 0;
          osun_count++;
      }

      Save(bmpNote[i], "ImageToNoteCut" + i.ToString());
    }

    ImageToNoteHistogram(bmpNote);
}
```

그림 3-5. 각각의 음 분리

그림 상에서는 표현이 힘들지만 각 음표 하나하나가 독립적으로

생성, 예시의 경우 총 39개의 이미지가 생성된다. 이후 각 음표별로 히스토그램을 작성한다.

그림 3-6. 그림3-5의 히스토그램

그림 3-6과 같이 작성된 히스토그램을 토대로 음의 높이 및 위치를 인식한다. 이와 같은 방식으로 분석된 정보는 미디 데이터와 매칭 되어 사용자의 선택 유무에 따라 재생된다.

III.2.2. MIDI

미디, MIDI는 Musical Instrument Digital Interface의 약자로 직역하면 “디지털 악기 인터페이스”이다. 이는 신디사이저, 전자악기, 컴퓨터간의 인터페이스 역할을 하는 H/W, S/W의 표준이다. 우리가 잘 알고 있는 전자악기가 표준에 따라 만들어졌다면 이 악기들 역시 MIDI를 지원한다고 할 수 있다.

타이밍 정보가 포함된 MIDI는 재사용이 가능하도록 .mid라는 확장자를 갖는 파일로 저장될 수 있다. 이러한 형태의 MIDI를 일반적으로 표준 MIDI(Standard MIDI File), 즉 SMF라고 한다. MIDI파일은 일반적으로 컴퓨터 기반의 시퀀싱 소프트웨어를 이용해 제작한다. 한편 MIDI는 독립적 녹음 및 편집을 위해 여러 개의 출력으로 음을 구성한다. 이를 이용해 특정 채널, 악기 패치를 변경함으로서 손쉽게 편집할 수 있다.

MIDI는 UART(Universal Asynchronous Receiver/Transmitter)라는 칩을 사용해 작동한다. 이 칩은 입력 비트를 10비트 패킷으로 묶는다. 그 중 첫 번째 비트는 항상 0이고 마지막 비트는 1이다. 그리하여 그 내부의 데이터 8비트를 전송한다. 컴퓨터를 비롯한 각종 전자악기는 이 데이터를 디코드 하여 데이터를 얻는다.

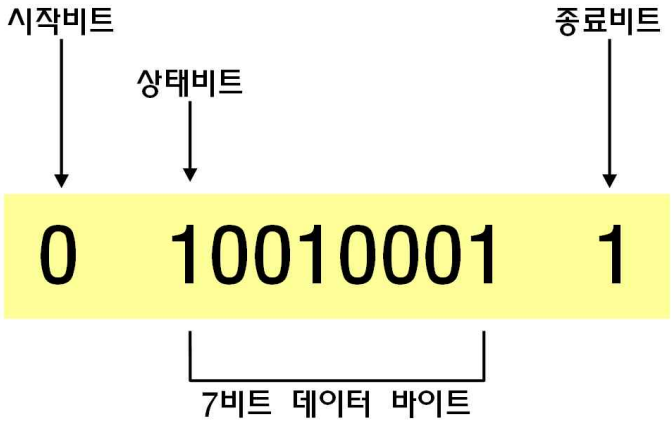


그림3-2. MIDI 데이터 구조

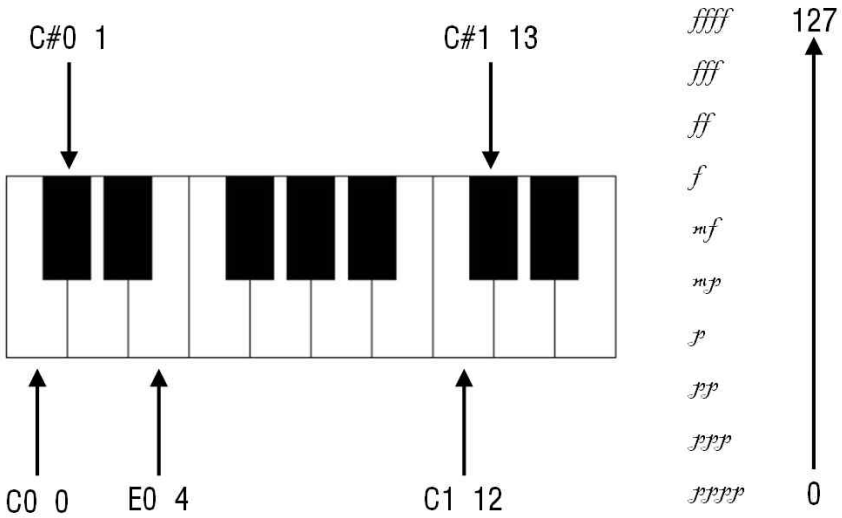


그림3-3. MIDI 데이터의 음계 그리고 소리의 세기

미디는 음향을 출력하는 청각적 데이터이지만 20Hz~20KHz의 가청 주파수 사운드 파일 데이터가 아닌 1과 0으로 구성된 디지털 데이터이다. 이 때문에 MIDI는 신디사이저에 음의 높이(Pitch)와 음의 세기(Velocity) 정보를 전달함으로서 그에 해당하는 소리를 출력한다. MIDI는 8비트로 된 데이터로 0부터 127까지의 데이터를 표현, 즉 128단계로 표현할 수 있다.

IV. 시스템 구현 및 테스트

IV.1. 시스템 구현

개발 코드는 별첨1.을 참조한다.

IV.2. 시스템 실행

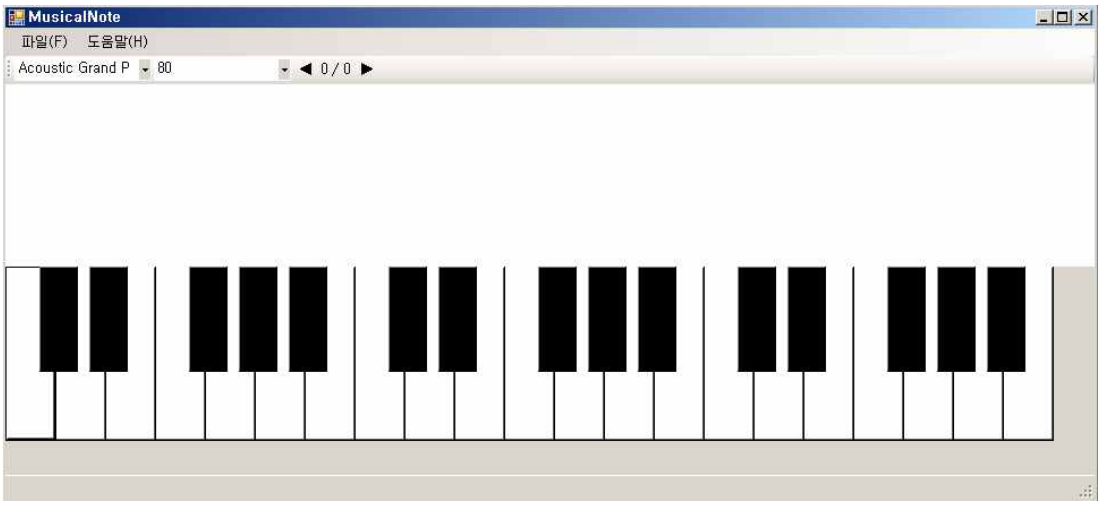


그림4-1. 초기화면

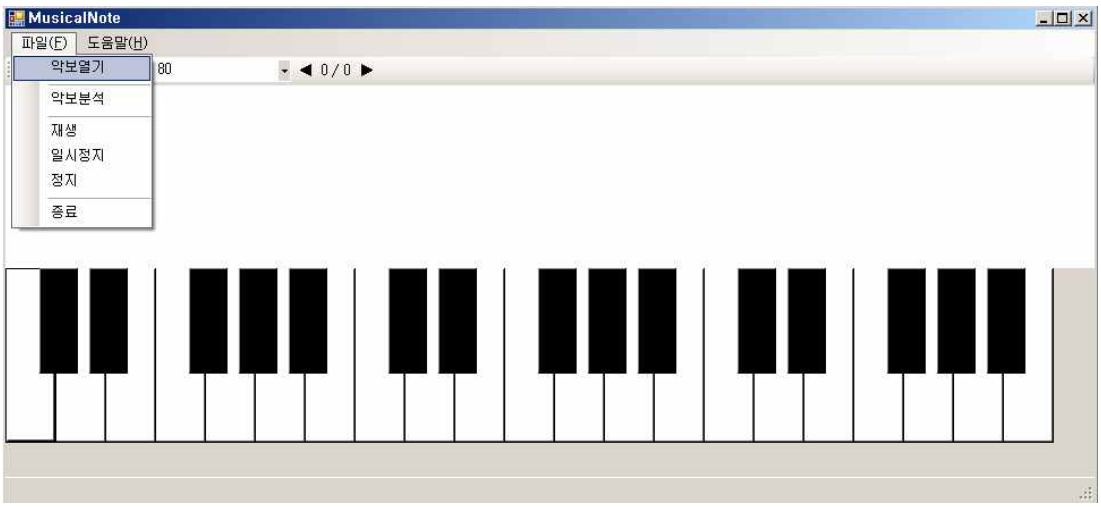


그림4-2. 악보열기

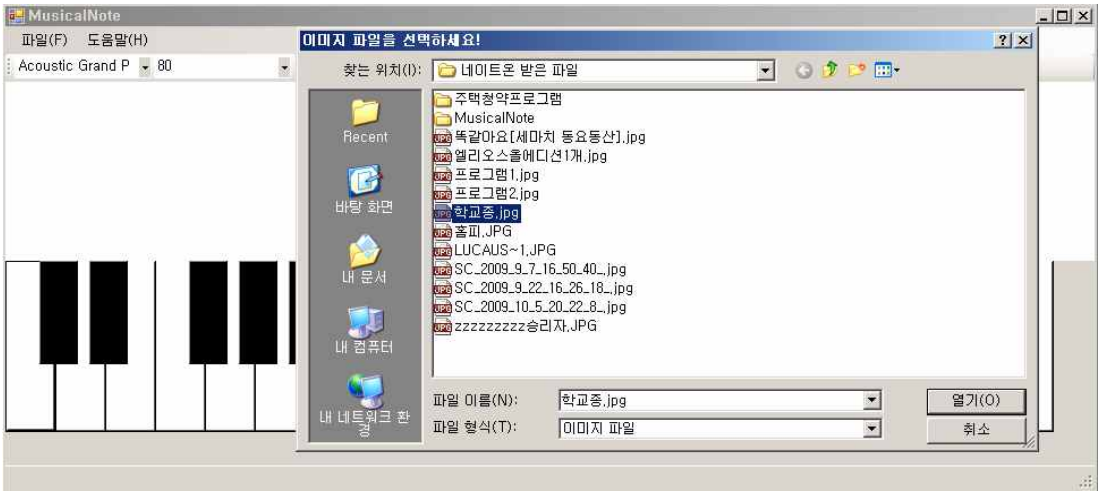


그림4-3. 악보 이미지 선택

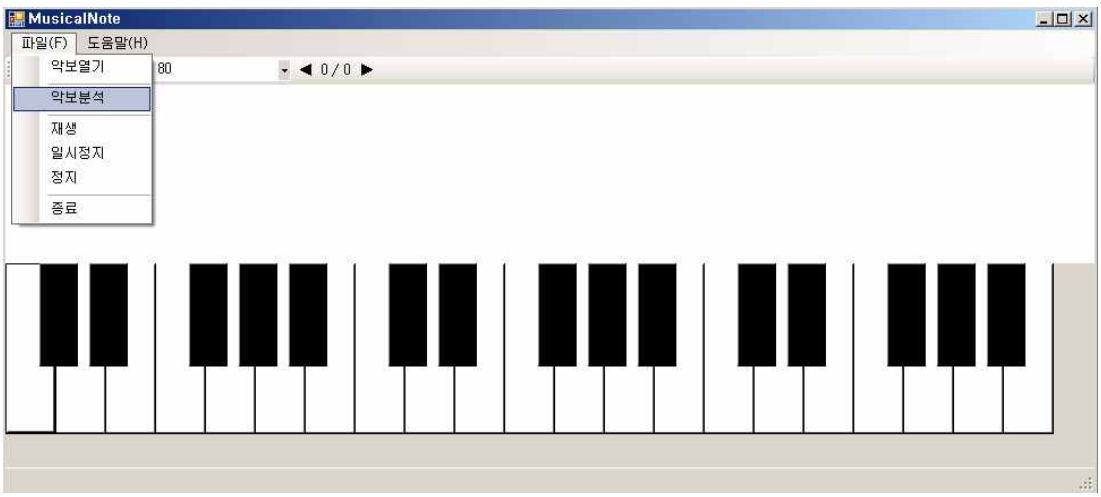


그림4-4. 선택한 악보 이미지 분석

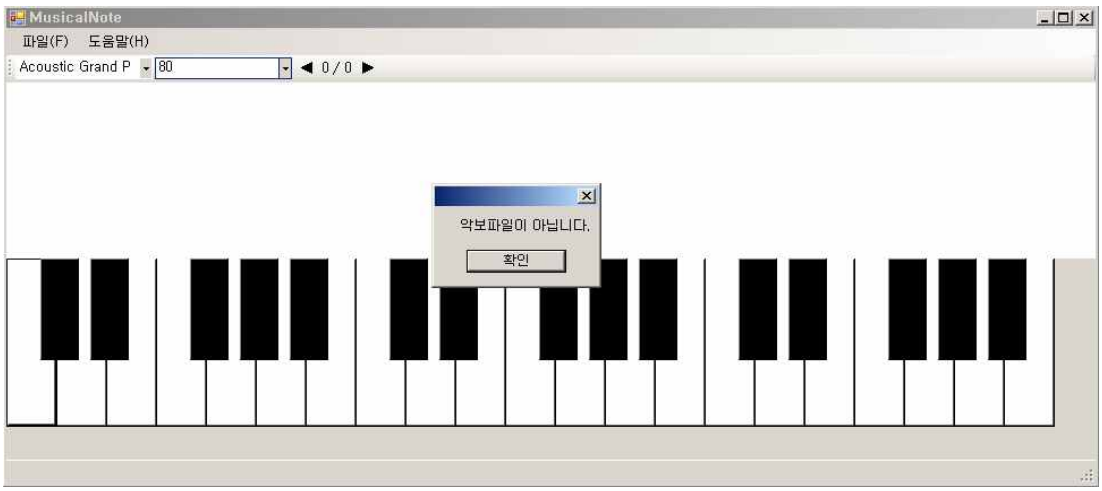


그림4-5. 선택한 이미지가 악보가 아닐 경우

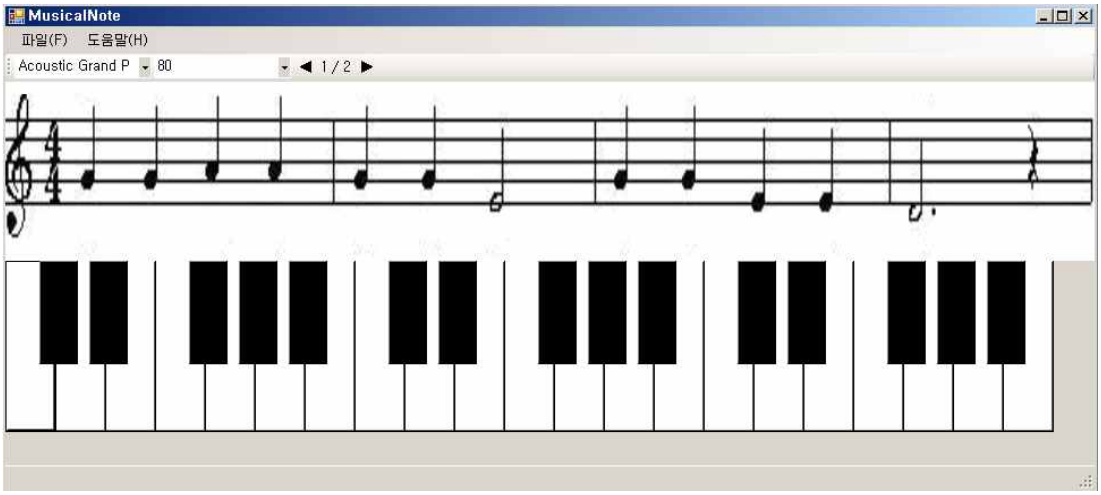


그림4-6. 이미지 분석 결과

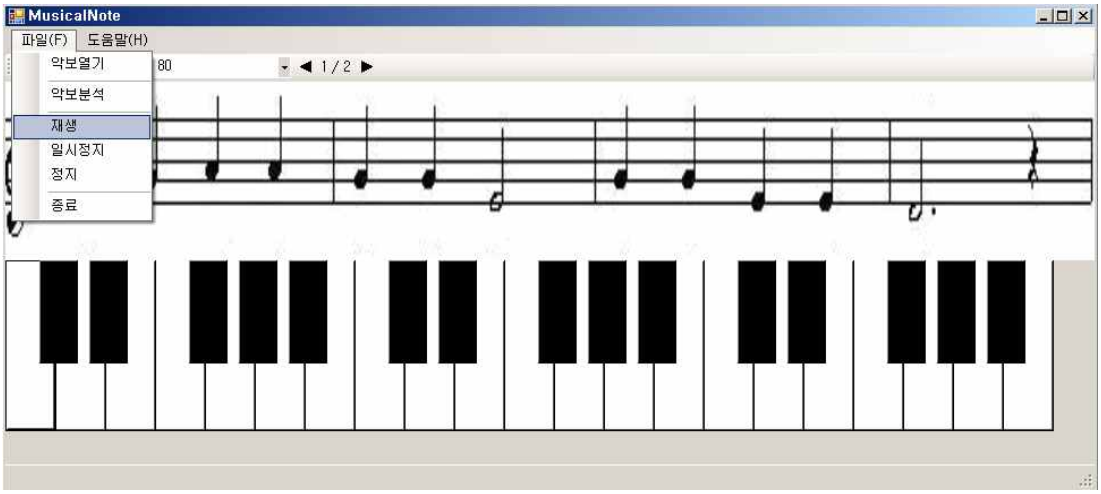


그림4-7. 분석한 악보 이미지 재생

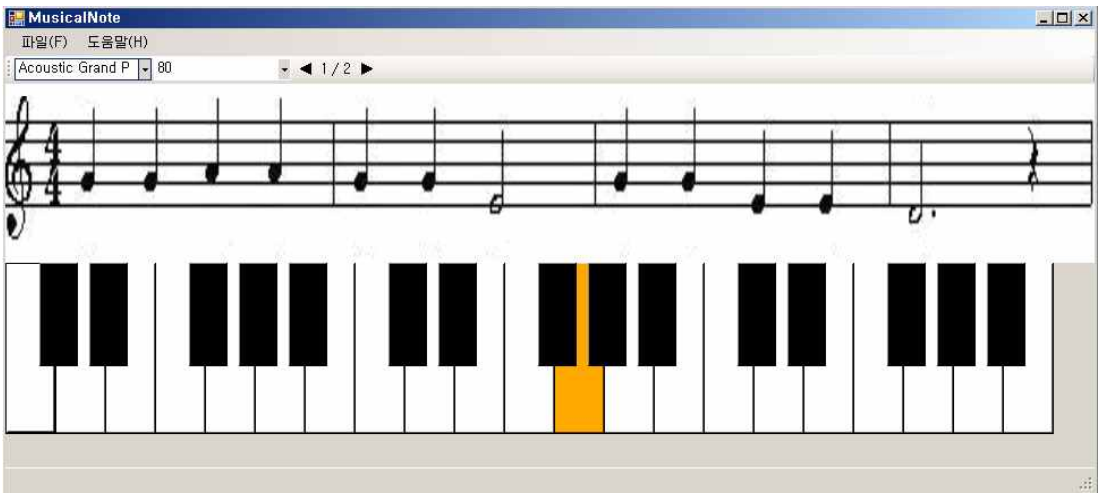


그림4-8. 재생 중

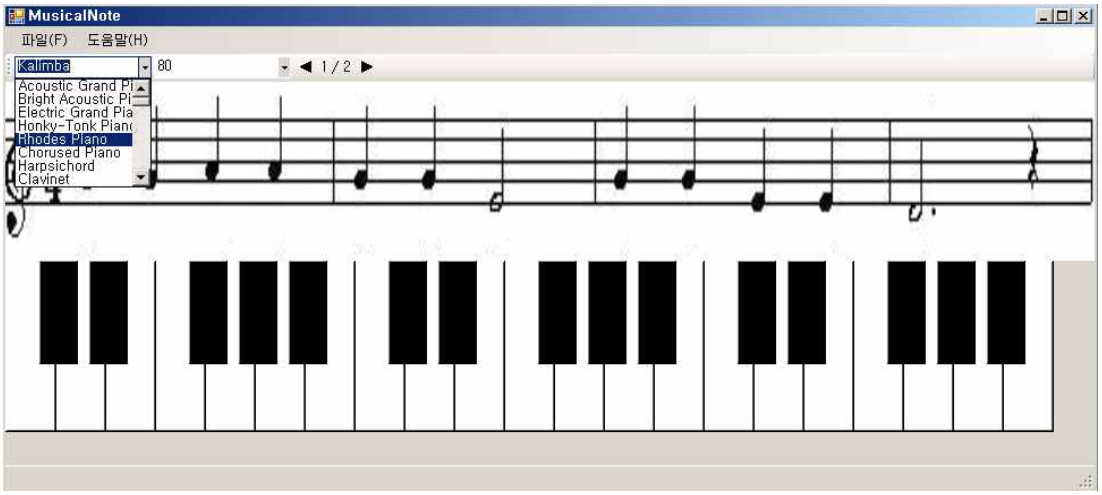


그림4-9. 악기 선택

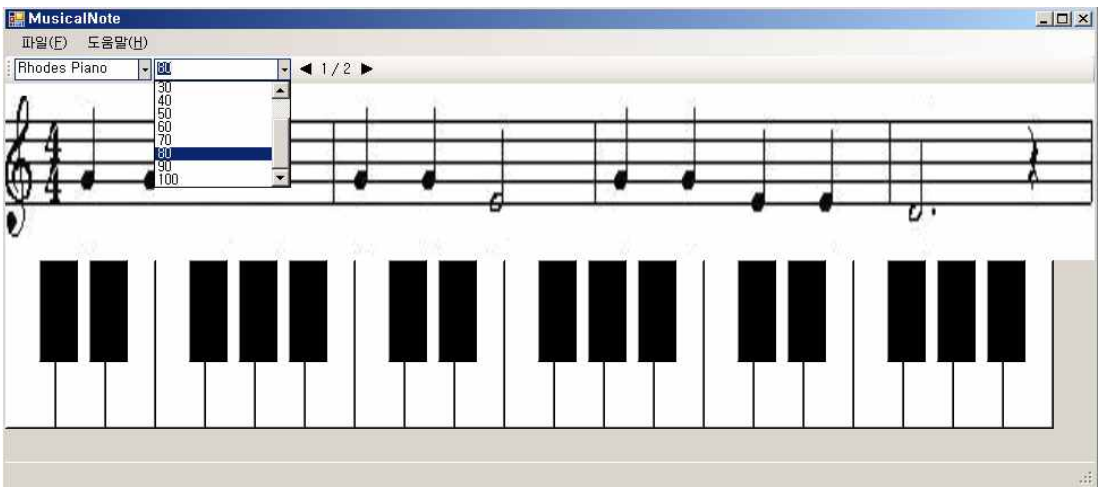


그림4-10. 볼륨 조절

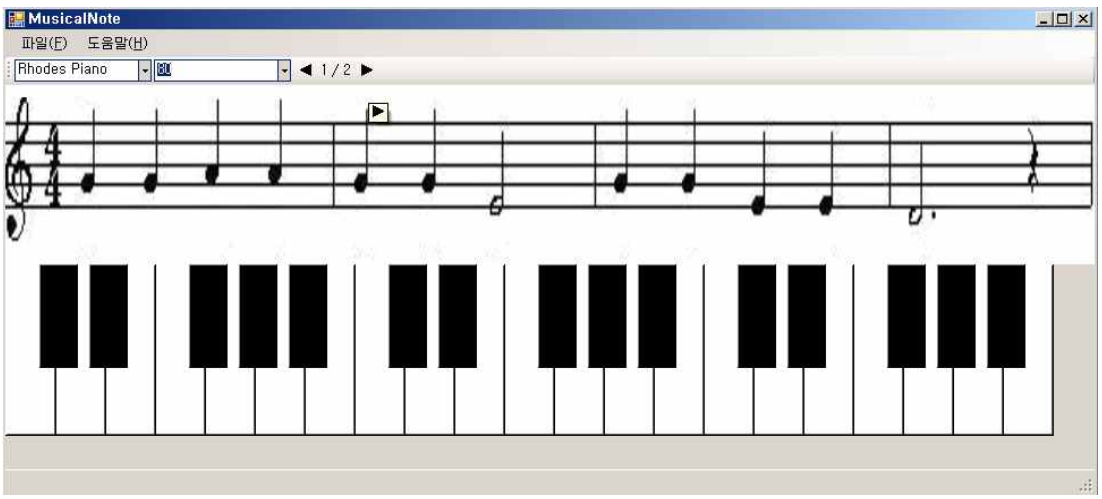


그림4-11. 소절 선택

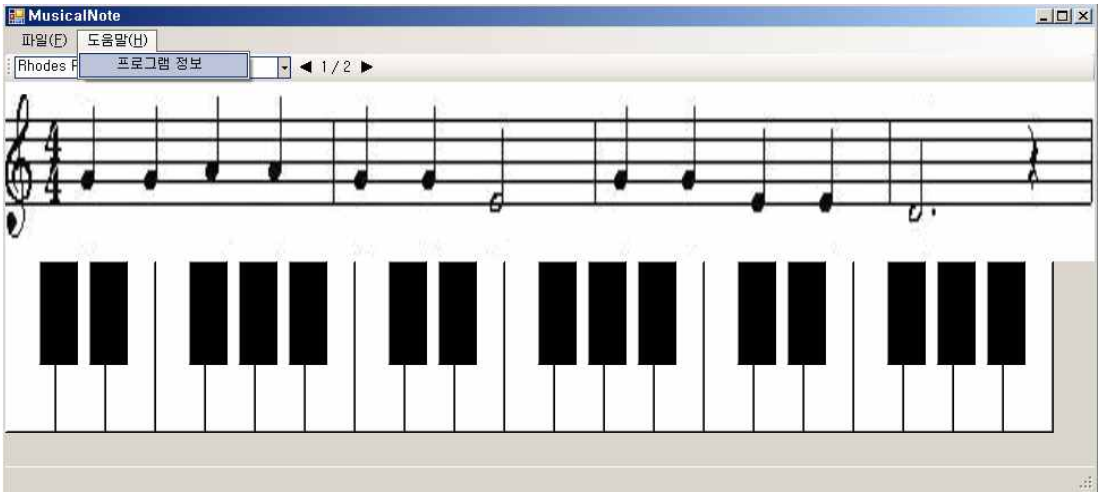


그림4-12. 도움말

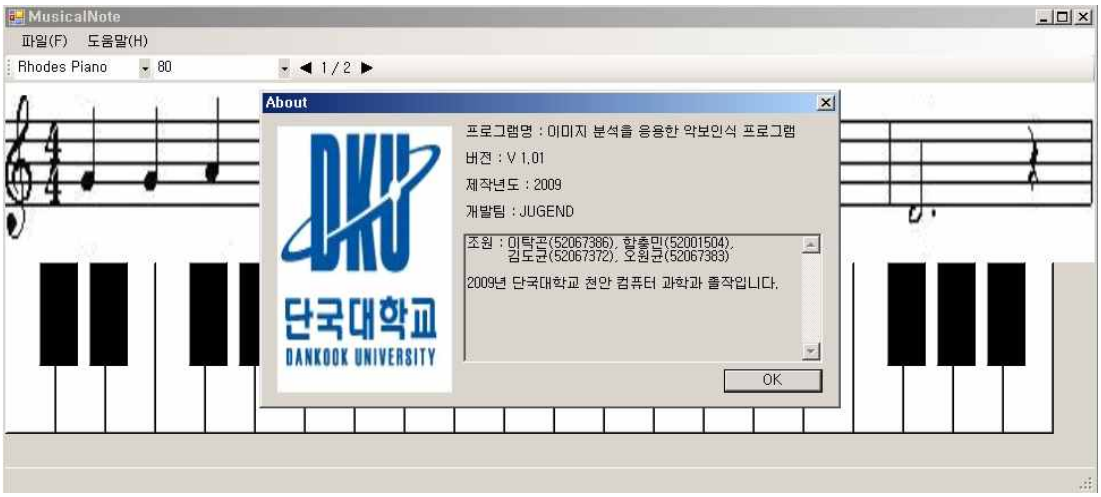


그림4-13. 프로그램 정보

IV.3. 테스트 결과

테스트에는 10곡의 악보가 사용됐다. 다음은 악보별 인식률과 그에 대한 평가이다.

곡 명	분석, 재생 결과	비고
겨울밤	매우 양호	·
곰 세마리	불량	마디에 음표가 많은 경우 인식률이 크게 낮다.
똑같아요	매우 양호	·
비행기	양호	반음을 표현하지 않는다
사과같은 내얼굴	불량	마디에 음표가 많은 경우 인식률이 크게 낮다.
싹싹 닦아라	양호	반음을 표현하지 않는다
자전거	양호	·
짝짜꿍	불량	마디에 음표가 많은 경우 인식률이 크게 낮다. 크기가 큰 쉼표(4분쉼표)를 음표로 인식한다.
학교종	양호	·
ABC노래	양호	·

표5-1. 테스트 결과

II.2항에서 제기된 요구사항에 대한 평가는 다음과 같다.

분류	요구사항	평가
이미지분석	비트맵 형식으로 작성된 이미지 파일을 읽을 수 있다.	양호
	악보 이미지를 분석 할 수 있다.	보통
	사용자는 분석에 대한 상세한 과정을 몰라도 된다.	양호
	분석결과를 사용자가 열람할 수 있다.	양호
MIDI변환	분석된 비트맵 이미지가 MIDI형식으로 변환된다.	보통
	사용자는 작성된 MIDI파일을 감상할 수 있다.	보통
	사용자는 작성된 MIDI파일을 편집할 수 있다(볼륨,악기)	양호
	사용자는 편집한 MIDI파일을 저장할 수 있다.	**불가**

표5-2. 요구사항 평가

테스트 결과 판단된 개선사항은 다음과 같다.

1. 음표를 제외한 기호 인식 추가(패턴인식)
2. 복잡한 악보 분석을 위한 알고리즘의 세밀화
3. 변환된 MIDI의 파일 저장 기능 추가

테스트 결과 비교적 간단하고 적은 음으로 구성된 곡과 비교해 복잡한 곡은 인식 및 변환의 정확도가 크게 떨어졌다. 복잡한 곡의 경우 음표 이외의 기호(4분 쉼표 등)를 하나의 음으로 인식하는 등의 문제가 있었다.

한편 제안된 이미지 분석 알고리즘은 히스토그램을 기준으로 음의 위치와 높이를 찾기 때문에 음 이외의 기호를 인식하지 못한다. 때문에 반올림(#), 반내림(b) 기호를 인식하지 못해 반음 부분, 즉 피아노에서 검은 건반부분이 사용되지 않는다.

또 프로그램의 컨셉 중 하나였던 “분석한 MIDI를 파일로 생성”하는 기능이 개발기간 부족으로 구현되지 못했다.

V. 평가

약 10개월간의 기간에 걸쳐 개발된 프로젝트의 결과물이 가까스로 모습을 드러냈지만 그 결과물은 아직 초라하다. 사전지식 습득 및 효율적인 알고리즘 제시에 난항을 겪었으며 개발 기간에 쫓겨 완성된 프로그램 역시 부족한 부분이 여기저기 눈에 띈다. 앞으로 지속적인 유지보수를 통해 좀 더 완성도 높은 시스템으로 만들어야 할 것이다.

VI. 참고도서 및 문헌

V.1. 서적

영상처리 이론과 실제(Randy Crane)
디지털 영상 처리(Rafael C. Gonzlez 외)
C# 디지털 영상처리(정민영 외)
디지털 영상처리 프로그래밍 따라하기(김학수)
자바로 구현한 영상처리(Nick Efford)

컴퓨터 음악(최영준)
디지털 음악 : 미디와 신디사이저(한진승)

V.2. 웹

OpenCV KOREA 대한민국 최고의 컴퓨터비젼 커뮤니티 (http://cafe.naver.com/opencv)
미디앤 사운드(http://www.mnshome.com/)

별첨1. 프로그램 소스 코드 리스트

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace MusicalNote
{
    public partial class FrmMain : Form
    {
        private Scale _scale = null
        private NoteInformation _noteInformation = new NoteInformation();
        private NoteHorizonHistogram[] _noteHorizonHistogram = new NoteHorizonHistogram[0];
        private VerticalHistogram[] _verticalHistogram = new VerticalHistogram[0];
        private SheetInformation[] _sheetInformation = new SheetInformation[0];

        #region PainoKeys Variable
        private Button[] btnBlackPianoKeys = new Button[0];
        private Button[] btnWhitePianoKeys = new Button[0];

        private Size _szBlackPianoKey = new Size(34, 93);
        private Size _szWhitePianoKey = new Size(45, 154);

        private Point[] _ptBlackPianoKey = new Point[] {
            new Point(30, 0), new Point(74, 0), new Point(162, 0), new Point(206, 0), new Point(250, 0),
            new Point(338, 0), new Point(382, 0), new Point(470, 0), new Point(514, 0), new Point(558, 0),
            new Point(646, 0), new Point(690, 0), new Point(778, 0), new Point(822, 0), new Point(866, 0),
        };
        private Point _ptWhitePianoKey = new Point(0, 0);

        private int[] _nBlackPianoKey = new int[] {
            54, 56, 59, 61, 63,
            66, 68, 71, 73, 75,
            78, 80, 83, 85, 87,
        };
        private int[] _nWhitePainoKey = new int[] {
            53, 55, 57, 58, 60, 62, 64,
            65, 67, 69, 70, 72, 74, 76,
            77, 79, 81, 82, 84, 86, 88,
        };
        #endregion
```

```
        #region Variable
        /// <summary>
        /// 오선에 위치 정보를 가지고 있는다.
        /// </summary>
        private ArrayList _arraySheet = new ArrayList();

        /// <summary>
        /// 악보 이미지
        /// </summary>
        private Bitmap _bitmap = null

        /// <summary>
        /// 수평 Histogram 이미지
        /// </summary>
        private Bitmap _bmphistogram = null

        /// <summary>
        /// 음표를 하나씩 생성한다.
```

```
/// </summary>
private Bitmap[] _bmpNote = new Bitmap[0];

/// <summary>
/// 소절 오선 이미지
/// </summary>
private Bitmap[] _bmpSheet = new Bitmap[0];

/// <summary>
/// MIDI Handler
/// </summary>
private IntPtr _midiHandler = IntPtr.Zero;

/// <summary>
/// 현재 연주 음
/// </summary>
private int _nAfterIndex = 0;

/// <summary>
/// 현재 연주하는 음에 전 음 높이
/// </summary>
private int _nBeforeIndex = 0;

/// <summary>
/// 현재 오선
/// </summary>
private int _nCurrent = 0;

/// <summary>
/// 볼륨크기
/// </summary>
private int _nVolume = 80;

/// <summary>
/// 선택한 미디 장치
/// </summary>
private int _nInstrument = 0;

/// <summary>
/// 현재 보여질 악보
/// </summary>
private int _nSheetIndex = 0;

/// <summary>
/// 악보 총 마디
/// </summary>
private int _nTotalBar = 0;

/// <summary>
/// 현재 타이머 위치
/// </summary>
private int _nTmrCurrent = 0;

/// <summary>
/// 수평 Histogram 최대값
/// </summary>
private int _nHistogramMax = 0;

/// <summary>
/// 수평 Histogram 배열 값
/// </summary>
private int[] _nHorHistogram = new int[0];

/// <summary>
/// 이미지 파일 경로
/// </summary>
private string _filename = ""
#endregion
```

```
        #region Initialize
        /// <summary>
        /// 초기화
        /// </summary>
        private void Initialize()
        {
            tmrPlay.Enabled = false

            _scale = new Scale();
            _noteInformation = new NoteInformation();
            _noteHorizonHistogram = new NoteHorizonHistogram[0];
            _verticalHistogram = new VerticalHistogram[0];
            _sheetInformation = new SheetInformation[0];

            _arraySheet = new ArrayList();

            _nAfterIndex = 0;
            _nBeforeIndex = 0;

            _nCurrent = 0;

            _nHistogramMax = 0;
            _nHorHistogram = new int[0];

            _nVolume = 80;
            _nTotalBar = 0;

            _nTmrCurrent = 0;

            _filename = ""

            InitializeVolumn();
            InitializeMIDI();

            picMusicalNote.Image = null
            tblPlayIndex.Text = string.Format("{0} / {1}", 0, 0);
        }
        #endregion
```

```
        #region InitializeButtonWhitePianoKeys
        private void InitializeButtonWhitePianoKeys()
        {
            btnWhitePianoKeys = new Button[21];

            for (int i = 0; i < btnWhitePianoKeys.Length; i++)
            {
                btnWhitePianoKeys[i] = new Button();
                btnWhitePianoKeys[i].Name = Convert.ToString(btnWhitePianoKeys[i]);
                btnWhitePianoKeys[i].BackColor = Color.White;
                btnWhitePianoKeys[i].Location = _ptWhitePianoKey;
                btnWhitePianoKeys[i].Size = _szWhitePianoKey;
                btnWhitePianoKeys[i].Tag = _nWhitePainoKey[i];

                pnlPianoKeys.Controls.Add(btnWhitePianoKeys[i]);

                _ptWhitePianoKey.X += _szWhitePianoKey.Width - 1;
            }

            this.Width = _ptWhitePianoKey.X + _szWhitePianoKey.Width;
        }
        #endregion
```

```
        #region InitializeButtonBlackPianoKeys
        private void InitializeButtonBlackPianoKeys()
        {
            btnBlackPianoKeys = new Button[15];

            for (int i = 0; i < btnBlackPianoKeys.Length; i++)
            {
                btnBlackPianoKeys[i] = new Button();
                btnBlackPianoKeys[i].Name = Convert.ToString(btnBlackPianoKeys[i]);
                btnBlackPianoKeys[i].BackColor = Color.Black;
                btnBlackPianoKeys[i].Location = _ptBlackPianoKey[i];
                btnBlackPianoKeys[i].Size = _szBlackPianoKey;
                btnBlackPianoKeys[i].Tag = _nBlackPianoKey[i];

                pnlPianoKeys.Controls.Add(btnBlackPianoKeys[i]);
                btnBlackPianoKeys[i].BringToFront();
            }
        }
        #endregion
```

```
        #region InitializeVolumn
        /// <summary>
        /// 볼륨 초기화
        /// </summary>
        private void InitializeVolumn()
        {
            for (int i = 0; i < tlbVolumeList.Items.Count; i++)
            {
                if (tlbVolumeList.Items[i].Equals(Convert.ToString(_nVolume)))
                    tlbVolumeList.SelectedIndex = i;
            }
        }
        #endregion
```

```
        #region FrmMain
        public FrmMain()
        {
            InitializeComponent();

            InitializeButtonWhitePianoKeys();
            InitializeButtonBlackPianoKeys();

            Initialize();
        }

        private void FrmMain_Load(object sender, EventArgs e)
        {
            string[] strNazvyRejstriku = new string[128];
            //Piano:
            strNazvyRejstriku[0] = "Acoustic Grand Piano"
            strNazvyRejstriku[1] = "Bright Acoustic Piano"
            strNazvyRejstriku[2] = "Electric Grand Piano"
            strNazvyRejstriku[3] = "Honky-Tonk Piano"
            strNazvyRejstriku[4] = "Rhodes Piano"
            strNazvyRejstriku[5] = "Chorused Piano"
            strNazvyRejstriku[6] = "Harpsichord"
            strNazvyRejstriku[7] = "Clavinet"
            //Chromatické bicí:
            strNazvyRejstriku[8] = "Celesta"
            strNazvyRejstriku[9] = "Glockenspiel"
            strNazvyRejstriku[10] = "Music Box"
            strNazvyRejstriku[11] = "Vibraphone"
            strNazvyRejstriku[12] = "Marimba"
```

```
strNazvyRejstriku[13] = "Xylophone"
strNazvyRejstriku[14] = "Tubular Bells"
strNazvyRejstriku[15] = "Dulcimer"
// Varhany:
strNazvyRejstriku[16] = "Hammond Organ"
strNazvyRejstriku[17] = "Percussive Organ"
strNazvyRejstriku[18] = "Rock Organ"
strNazvyRejstriku[19] = "Church Organ"
strNazvyRejstriku[20] = "Reed Organ"
strNazvyRejstriku[21] = "Accordian"
strNazvyRejstriku[22] = "Harmonica"
strNazvyRejstriku[23] = "Tango Accordian"
// Kytara:
strNazvyRejstriku[24] = "Acoustic Guitar (nylon)"
strNazvyRejstriku[25] = "Acoustic Guitar (steel)"
strNazvyRejstriku[26] = "Electric Guitar (jazz)"
strNazvyRejstriku[27] = "Electric Guitar (clean)"
strNazvyRejstriku[28] = "Electric Guitar (muted)"
strNazvyRejstriku[29] = "Overdriven Guitar"
strNazvyRejstriku[30] = "Distortion Guitar"
strNazvyRejstriku[31] = "Guitar Harmonics"
// Baskytara:
strNazvyRejstriku[32] = "Acoustic Bass"
strNazvyRejstriku[33] = "Electric Bass (finger)"
strNazvyRejstriku[34] = "Electric Bass (pick)"
strNazvyRejstriku[35] = "Fretless Bass"
strNazvyRejstriku[36] = "Slap Bass 1"
strNazvyRejstriku[37] = "Slap Bass 2"
strNazvyRejstriku[38] = "Synth Bass 1"
strNazvyRejstriku[39] = "Synth Bass 2"
// Smyčce:
strNazvyRejstriku[40] = "Violin"
strNazvyRejstriku[41] = "Viola"
strNazvyRejstriku[42] = "Cello"
strNazvyRejstriku[43] = "Contrabass"
strNazvyRejstriku[44] = "Tremolo Strings"
strNazvyRejstriku[45] = "Pizzicato Strings"
strNazvyRejstriku[46] = "Orchestral Harp"
strNazvyRejstriku[47] = "Timpani"
// Sbor:
strNazvyRejstriku[48] = "String Ensemble 1"
strNazvyRejstriku[49] = "String Ensemble 2"
strNazvyRejstriku[50] = "Synth Strings 1"
strNazvyRejstriku[51] = "Synth Strings 2"
strNazvyRejstriku[52] = "Choir Aahs"
strNazvyRejstriku[53] = "Voice Oohs"
strNazvyRejstriku[54] = "Synth Voice"
strNazvyRejstriku[55] = "Orchestra Hit"
// Žestě:
strNazvyRejstriku[56] = "Trumpet"
strNazvyRejstriku[57] = "Trombone"
strNazvyRejstriku[58] = "Tuba"
strNazvyRejstriku[59] = "Muted Trumpet"
strNazvyRejstriku[60] = "French Horn"
strNazvyRejstriku[61] = "Brass Section"
strNazvyRejstriku[62] = "Synth Brass 1"
strNazvyRejstriku[63] = "Synth Brass 2"
// Dechové:
strNazvyRejstriku[64] = "Soprano Sax"
strNazvyRejstriku[65] = "Alto Sax"
strNazvyRejstriku[66] = "Tenor Sax"
strNazvyRejstriku[67] = "Baritone Sax"
strNazvyRejstriku[68] = "Oboe"
strNazvyRejstriku[69] = "English Horn"
strNazvyRejstriku[70] = "BassoOn"
strNazvyRejstriku[71] = "Clarinet"
// Píšťaly:
strNazvyRejstriku[72] = "Piccolo"
```

```
        strNazvyRejstriku[73] = "Flute"
        strNazvyRejstriku[74] = "Recorder"
        strNazvyRejstriku[75] = "Pan Flute"
        strNazvyRejstriku[76] = "Bottle Bow"
        strNazvyRejstriku[77] = "Shakuhachi"
        strNazvyRejstriku[78] = "Whistle"
        strNazvyRejstriku[79] = "Ocarina"
        // Syntetické 1:
        strNazvyRejstriku[80] = "Lead 1 (square)"
        strNazvyRejstriku[81] = "Lead 2 (sawtooth)"
        strNazvyRejstriku[82] = "Lead 3 (caliope lead)"
        strNazvyRejstriku[83] = "Lead 4 (chiff lead)"
        strNazvyRejstriku[84] = "Lead 5 (charang)"
        strNazvyRejstriku[85] = "Lead 6 (voice)"
        strNazvyRejstriku[86] = "Lead 7 (fifths)"
        strNazvyRejstriku[87] = "Lead 8 (brass + lead)"
        // Syntetické 2:
        strNazvyRejstriku[88] = "Pad 1 (new age)"
        strNazvyRejstriku[89] = "Pad 2 (warm)"
        strNazvyRejstriku[90] = "Pad 3 (polysynth)"
        strNazvyRejstriku[91] = "Pad 4 (choir)"
        strNazvyRejstriku[92] = "Pad 5 (bowed)"
        strNazvyRejstriku[93] = "Pad 6 (metallic)"
        strNazvyRejstriku[94] = "Pad 7 (halo)"
        strNazvyRejstriku[95] = "Pad 8 (sweep)"
        // Efekty:
        strNazvyRejstriku[96] = "FX 1 (rain)"
        strNazvyRejstriku[97] = "FX 2 (soundtrack)"
        strNazvyRejstriku[98] = "FX 3 (crystal)"
        strNazvyRejstriku[99] = "FX 4 (atmosphere)"
        strNazvyRejstriku[100] = "FX 5 (brightness)"
        strNazvyRejstriku[101] = "FX 6 (goblins)"
        strNazvyRejstriku[102] = "FX 7 (echoes)"
        strNazvyRejstriku[103] = "FX 8 (sci-fi)"
        // Různé:
        strNazvyRejstriku[104] = "Sitar"
        strNazvyRejstriku[105] = "Banjo"
        strNazvyRejstriku[106] = "Shamisen"
        strNazvyRejstriku[107] = "Koto"
        strNazvyRejstriku[108] = "Kalimba"
        strNazvyRejstriku[109] = "Bagpipe"
        strNazvyRejstriku[110] = "Fiddle"
        strNazvyRejstriku[111] = "Shanai"
        // Bicí:
        strNazvyRejstriku[112] = "Tinkle Bell"
        strNazvyRejstriku[113] = "Agogo"
        strNazvyRejstriku[114] = "Steel Drums"
        strNazvyRejstriku[115] = "Woodblock"
        strNazvyRejstriku[116] = "Taiko Drum"
        strNazvyRejstriku[117] = "Melodic Tom"
        strNazvyRejstriku[118] = "Synth Drum"
        strNazvyRejstriku[119] = "Reverse Cymbal"
        // Zvukové efekty:
        strNazvyRejstriku[120] = "Guitar Fret Noise"
        strNazvyRejstriku[121] = "Breath Noise"
        strNazvyRejstriku[122] = "Seashore"
        strNazvyRejstriku[123] = "Bird Tweet"
        strNazvyRejstriku[124] = "Telephone Ring"
        strNazvyRejstriku[125] = "Helicopter"
        strNazvyRejstriku[126] = "Applause"
        strNazvyRejstriku[127] = "Gunshot"

        for (int i = 0; i < strNazvyRejstriku.Length; i++)
            tlbInstrumentList.Items.Add(strNazvyRejstriku[i]);

        tlbInstrumentList.SelectedIndex = 0;
    }
    #endregion
```

```
        private void LoadImage()
        {
            OpenFileDialog openFileDlg = new OpenFileDialog();
            openFileDlg.Title = "이미지 파일을 선택하세요!"
            openFileDlg.Filter = "이미지 파일|*.bmp;*.jpg;*.gif;*.tif;*.tiff"

            if (openFileDlg.ShowDialog().Equals(DialogResult.OK))
            {
                _filename = openFileDlg.FileName;
                _bitmap = new Bitmap(_filename);
            }
        }

    //   private void Save(Bitmap bitmap, string method)
    //   {
    //       bitmap.Save(string.Format(@"C:₩aW{0}.bmp", method));
    //   }

        /// <summary>
        /// 악보 이미지를 이진화 한다.
        /// </summary>
        private void ImageToBinarization()
        {
            for (int y = 0; y < _bitmap.Height; y++)
            {
                for (int x = 0; x < _bitmap.Width; x++)
                {
                    if (_bitmap.GetPixel(x, y).R < 126 && _bitmap.GetPixel(x, y).G < 126 &&
_bitmap.GetPixel(x, y).B < 126)
                        _bitmap.SetPixel(x, y, Color.Black);
                    else
                        _bitmap.SetPixel(x, y, Color.White);
                }
            }

        //    Save(_bitmap, "ImageToBinarization");
        }

        /// <summary>
        /// 악보 이미지 전체를 수평기준으로 Histogram 한다.
        /// </summary>
        private void ImageToHistogram()
        {
            _nHorHistogram = new int[_bitmap.Height];
            //y축을 기준으로 x축을 돌면서 값 누적하기
            for (int y = 0; y < _bitmap.Height; y++)
            {
                for (int x = 0; x < _bitmap.Width; x++)
                {
                    if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).G == 0 &&
_bitmap.GetPixel(x, y).B == 0)
                        _nHorHistogram[y] = _nHorHistogram[y] + 1;
                }
            }

            _bmphistogram = new Bitmap(_bitmap.Width, _bitmap.Height);
            for (int y = 0; y < _bitmap.Height; y++)
            {
                for (int x = 0; x < _bitmap.Width; x++)
                {
                    if (x < _nHorHistogram[y])
                        _bmphistogram.SetPixel(x, y, Color.Black);
                    else
                        _bmphistogram.SetPixel(x, y, Color.White);
```

```
        }

        // 수평 히스토그램 최대값 저장..
        if (_nHistogramMax < _nHorHistogram[y])
        {
            _nHistogramMax = _nHorHistogram[y];
        }
    }

//    Save(_bmphistogram, "ImageToHistogram");
}

/// <summary>
/// Histogram을 참조하여 악보 이미지에 오선위치, 두께, 간격, 소절을 정보를 가지고 온다.
/// </summary>
private void ImageToInformation()
{
    SheetCheck tmp_st = null
    SheetCheck sheetCheck = new SheetCheck();

    bool bStarted = false
    bool bBlack = false

    int doogge = 0;
    int gap = 0;
    int count_line = 0;
    int count_gap = 0;
    int barCount = 0;

    for (int y = 0; y < _bmphistogram.Height; y++)
    {
        //흰 -> 검
        if (!bBlack
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).R == 0
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).G == 0
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).B == 0)
        {
            //처음부분이 아니다
            if (bStarted)
            {
                sheetCheck.Gap[count_gap] = gap;
                gap = 0;
                count_gap++;
            }
            //처음 부분이다
            if (!bStarted)
                bStarted = true

            bBlack = true
            sheetCheck.Line[count_line] = y;
            doogge++;
        }

            // 검 -> 검
        else if (bBlack
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).R == 0
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).G == 0
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).B == 0)
        {
            doogge++;
        }
             //검 -> 흰
        else if (bBlack
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).R == 255
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).G == 255
            && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).B == 255)
        {
            bBlack = false
```

```
                    sheetCheck.Width[count_line] = doogge;
                    doogge = 0;
                    count_line++;
                    if (count_line == 5)
                    {
                        count_line = 0;
                        count_gap = 0;
                        barCount++;
                        bStarted = false
                        _arraySheet.Add(sheetCheck);
                        sheetCheck = new SheetCheck();
                    }
                }

                //흰 -> 흰
                else if (!bBlack && bStarted
                    && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).R == 255
                    && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).G == 255
                    && _bmphistogram.GetPixel((_nHistogramMax * 7) / 10, y).B == 255)
                {
                    gap++;
                }
            }

            for (int i = 0; i < _arraySheet.Count; i++)
            {
                tmp_st = (SheetCheck)_arraySheet[i];

                for (int j = 0; j < tmp_st.Line.Length; j++)
                {
                    if (j == 4) continue
                }
            }

            _nTotalBar = barCount;
        }

        /// <summary>
        /// 각 오선에 첫 번째 / 마지막 줄에 위치를 생성 한다.
        /// </summary>
        private void ImageToSheetPosition()
        {
            _sheetInformation = new SheetInformation[_nTotalBar];

            for (int i = 0; i < _nTotalBar; i++)
            {
                _sheetInformation[i] = new SheetInformation();
            }

            for (int i = 0; i < _nTotalBar; i++)
            {
                _sheetInformation[i].Tops = new Point(_sheetInformation[i].Tops.X,
((SheetCheck)_arraySheet[i]).Line[0] - ((((SheetCheck)_arraySheet[i]).Gap[0]) * 2 +
((SheetCheck)_arraySheet[i]).Width[0]));
                _sheetInformation[i].Downs = new Point(_sheetInformation[i].Downs.X,
((SheetCheck)_arraySheet[i]).Line[4] + ((((SheetCheck)_arraySheet[i]).Gap[3]) * 3 +
((SheetCheck)_arraySheet[i]).Width[4] * 2));
            }
        }

        /// <summary>
        /// 악보 이미지에서 오선을 삭제 한다.
        /// </summary>
        private void ImageToSheetDelete()
        {
            for (int i = 0; i < _sheetInformation.Length; i++)
            {
                for (int y = 0; y < _bitmap.Height; y++)
```

```
                {
                    for (int o = 0; o < 5; o++)
                    {
                        if (y == ((SheetCheck)_arraySheet[i]).Line[o])
                        {
                            for (int x = 0; x < _bitmap.Width; x++)
                            {
                                if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).G == 0 &&
_bitmap.GetPixel(x, y).B == 0
                                    && _bitmap.GetPixel(x, y - 1).R == 255 && _bitmap.GetPixel(x, y - 1).G
== 255 && _bitmap.GetPixel(x, y - 1).B == 255
                                    && _bitmap.GetPixel(x, y + 1).R == 255 && _bitmap.GetPixel(x, y + 1).G
== 255 && _bitmap.GetPixel(x, y + 1).B == 255)
                                {
                                    _bitmap.SetPixel(x, y, Color.White);
                                }

                            }
                        }
                    }
                }
            }

         //   Save(_bitmap, "ImageToSheetDelete");
        }

        /// <summary>
        /// 악보 이미지에 각 소절에 Histogram을 생성한다.
        /// </summary>
        private void ImageToBarHistogram()
        {
            for (int i = 0; i < _sheetInformation.Length; i++)
            {
                _sheetInformation[i].VericalHistogram = new int[_bitmap.Width];

                for (int x = 0; x < _bitmap.Width; x++)
                {
                    for (int y = _sheetInformation[i].Tops.Y; y < _sheetInformation[i].Downs.Y; y++)
                    {
                        if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).G == 0 &&
_bitmap.GetPixel(x, y).B == 0)
                        {
                            _sheetInformation[i].VericalHistogram[x] += 1;
                        }
                    }
                }
            }
        }

        /// <summary>
        /// 각 소절을 단위로 생성한다.
        /// </summary>
        private void ImageToBar()
         {
             for (int i = 0; i < _sheetInformation.Length; i++)
             {
                 for (int x = 0; x < _bitmap.Width; x++)
                 {
                     if (_sheetInformation[i].VericalHistogram[x] > 0)
                     {
                         _sheetInformation[i].Tops = new Point(x - 1, _sheetInformation[i].Tops.Y);
                         break
                     }
                 }

                 for (int x = _bitmap.Width - 1; x > 0; x--)
                 {
                     if (_sheetInformation[i].VericalHistogram[x] > 0)
```

```
                {
                    _sheetInformation[i].Downs = new Point(x + 1, _sheetInformation[i].Downs.Y);
                    break
                }
            }
        }
    }

    /// <summary>
    /// 마디의 총 음표 개수를 확인한다.
    /// </summary>
    private void ImageToNoteCount()
    {
        bool isBlack = false

        int tempCount = 0;
        int nuzukCount = 0;

        for (int i = 0; i < _sheetInformation.Length; i++)
        {
            for (int x = 0; x < _bitmap.Width; x++)
            {
                if (isBlack == false && _sheetInformation[i].VericalHistogram[x] > 0)
                {
                    isBlack = true
                }
                else if (isBlack == true && _sheetInformation[i].VericalHistogram[x] <= 0)
                {
                    tempCount++;
                    nuzukCount++;
                    isBlack = false
                }
            }

            _noteInformation.TotalCount += tempCount;
            _sheetInformation[i].SheetCount = tempCount;
            _sheetInformation[i].NoteCount = nuzukCount;
            tempCount = 0;
        }
    }

    /// <summary>
    /// 음표 좌우 포인트를 측정 한다.
    /// </summary>
    private void ImageToNotePosition()
    {
        bool bBlack = false
        int tempCount = 0;

        for (int i = 0; i < _sheetInformation.Length; i++)
        {
            for (int x = 0; x < _bitmap.Width; x++)
            {
                if (!bBlack && _sheetInformation[i].VericalHistogram[x] > 0)
                {
                    _noteInformation.Tops[tempCount] = new Point(x - 1,
_noteInformation.Tops[tempCount].Y);
                    bBlack = true
                }
                else if (bBlack && _sheetInformation[i].VericalHistogram[x] <= 0)
                {
                    _noteInformation.Downs[tempCount] = new Point(x,
_noteInformation.Tops[tempCount].Y);
                    bBlack = false

                    tempCount++;
                }
            }
```

```
            }

            Bitmap originBMP = new Bitmap(_filename);
            _bmpSheet = new Bitmap[_sheetInformation.Length];
            for (int i = 0; i < _bmpSheet.Length; i++)
            {
                _bmpSheet[i] = new Bitmap(_sheetInformation[i].Downs.X - _sheetInformation[i].Tops.X,
_sheetInformation[i].Downs.Y - _sheetInformation[i].Tops.Y);
                for (int y = _sheetInformation[i].Tops.Y + 1; y < _sheetInformation[i].Downs.Y; y++)
                {
                    for (int x = _sheetInformation[i].Tops.X + 1; x < _sheetInformation[i].Downs.X; x++)
                    {

                        _bmpSheet[i].SetPixel(x - (_sheetInformation[i].Tops.X + 1), y -
(_sheetInformation[i].Tops.Y + 1), originBMP.GetPixel(x, y));

                    }
                }

              //  Save(_bmpSheet[i], "ImageToNotePosition" + i.ToString());
            }
        }

       /// <summary>
        /// 음표를 잘라낸다.
        /// </summary>
        private void ImageToNoteCut()
        {
            Bitmap[] bmpNote = new Bitmap[_noteInformation.TotalCount];

            int osun_width = 0;
            int osun_height = 0;
            int osun_count = 0;

            int note_count = 0;

            for (int i = 0; i < bmpNote.Length; i++)
            {
                osun_height = _sheetInformation[osun_count].Downs.Y -
_sheetInformation[osun_count].Tops.Y;
                osun_width = _noteInformation.Downs[i].X - _noteInformation.Tops[i].X;

                bmpNote[i] = new Bitmap(osun_width, osun_height);

                _noteInformation.Tops[i] = new Point(_noteInformation.Tops[i].X,
_sheetInformation[osun_count].Tops.Y);
                _noteInformation.Downs[i] = new Point(_noteInformation.Downs[i].X,
_sheetInformation[osun_count].Downs.Y);

                for (int y = _sheetInformation[osun_count].Tops.Y; y <
_sheetInformation[osun_count].Downs.Y; y++)
                {
                    for (int x = _noteInformation.Tops[i].X + 1; x < _noteInformation.Downs[i].X; x++)
                    {
                        if (_bitmap.GetPixel(x, y).R == 0 &&_bitmap.GetPixel(x, y).B == 0 &&
_bitmap.GetPixel(x, y).G == 0)
                        {
                            bmpNote[i].SetPixel(x - _noteInformation.Tops[i].X, y -
_sheetInformation[osun_count].Tops.Y, Color.Black);
                        }
                        else
                        {
                            bmpNote[i].SetPixel(x - _noteInformation.Tops[i].X, y -
_sheetInformation[osun_count].Tops.Y, Color.White);
                        }
                    }
                }
```

```
                note_count++;
                if (note_count == _sheetInformation[osun_count].SheetCount)
                {
                    note_count = 0;
                    osun_count++;
                }

            //    Save(bmpNote[i], "ImageToNoteCut" + i.ToString());
            }

            ImageToNoteHistogram(bmpNote);
        }

        /// <summary>
        /// 음표에 Histogram을 작성 한다.
        /// </summary>
        private void ImageToNoteHistogram(Bitmap[] bmpNote)
        {
            _noteHorizonHistogram = new NoteHorizonHistogram[_noteInformation.TotalCount];

            for (int i = 0; i < _noteInformation.TotalCount; i++)
            {
                _noteHorizonHistogram[i] = new NoteHorizonHistogram();
                _noteHorizonHistogram[i].TotalHeight = (_noteInformation.Downs[i].Y - 1) -
(_noteInformation.Tops[i].Y + 1) + 1;
            }

            for (int i = 0; i < _noteInformation.TotalCount; i++)
            {
                for (int y = 0; y < _noteHorizonHistogram[i].TotalHeight; y++)
                {
                    for (int x = _noteInformation.Tops[i].X + 1; x < _noteInformation.Downs[i].X; x++)
                    {
                        if (bmpNote[i].GetPixel(x - _noteInformation.Tops[i].X, y).R == 0 &&
bmpNote[i].GetPixel(x - _noteInformation.Tops[i].X, y).G == 0 && bmpNote[i].GetPixel(x -
_noteInformation.Tops[i].X, y).B == 0)
                        {
                            _noteHorizonHistogram[i].HorizonHistogrma[y] += 1;
                        }
                    }
                }
            }

            Bitmap[] tmp_h_bmp = new Bitmap[_noteInformation.TotalCount];

            int note_width = 0;
            int note_height = 0;

            for (int i = 0; i < _noteInformation.TotalCount; i++)
            {
                note_height = _noteHorizonHistogram[i].TotalHeight;

                note_width = _noteInformation.Downs[i].X - _noteInformation.Tops[i].X - 1;

                tmp_h_bmp[i] = new Bitmap(note_width, note_height);


                for (int y = 0; y < note_height; y++)
                {
                    for (int x = 0; x < note_width; x++)
                    {
                        if (x < _noteHorizonHistogram[i].HorizonHistogrma[y])
                        {
                            tmp_h_bmp[i].SetPixel(x, y, Color.Black);
                        }
                        else
                        {
                            tmp_h_bmp[i].SetPixel(x, y, Color.White);
```

```
                    }
                }
            }

        //      Save(tmp_h_bmp[i], "ImageToNoteHistogram" + i.ToString());
        }
    }

    /// <summary>
    /// 수평 Histogram에 위,아래를 잘라내기 한다.
    /// </summary>
    private void ImageToHoriHistogramCut()
    {
        bool isStart = false
        bool isBlack = false
        int new_total_height = 0;
        int original_note_LeftTop_Y = 0;


        for (int i = 0; i < _noteInformation.TotalCount; i++)
        {
            new_total_height = 0;
            original_note_LeftTop_Y = _noteInformation.Tops[i].Y;

            if (_noteHorizonHistogram[i].HorizonHistogrma[0] == 0)
            {
                // 변동사항 없음..
            }
            else if (_noteHorizonHistogram[i].HorizonHistogrma[0] != 0)
            {
                isStart = true
                isBlack = true
                new_total_height++;
                //noteSet1.note_1_LeftTop[i].Y 는  오선에세 받은값 그대로 쓴다.
            }


            for (int y = 1; y < _noteHorizonHistogram[i].TotalHeight; y++)
            {
                if (_noteHorizonHistogram[i].HorizonHistogrma[y] == 0 && isBlack == false)
                {
                    continue       //현재 픽셀이 흰색 && 이전 필셀은 흰색
                }
                else if (_noteHorizonHistogram[i].HorizonHistogrma[y] != 0 && isBlack == false &&
isStart == false)
                {
                    //현재 픽셀이 검은색 && 이전 필셀은 흰색
                    _noteInformation.Tops[i] = new Point(_noteInformation.Tops[i].X,
original_note_LeftTop_Y + y - 1);
                    isBlack = true
                    isStart = true
                    new_total_height++;
                }
                else if (_noteHorizonHistogram[i].HorizonHistogrma[y] != 0 && isBlack == true)
                {
                    //현재 픽셀이 검은색 && 이전 픽셀이 검은색
                    new_total_height++;
                    continue
                }
                else if (_noteHorizonHistogram[i].HorizonHistogrma[y] == 0 && isBlack == true)
                {
                    //현재 픽셀은 흰색 && 이전 픽셀은 검은색
                    _noteInformation.Downs[i] = new Point(_noteInformation.Downs[i].X,
original_note_LeftTop_Y + y);
                    isBlack = false
                    isStart = false
                    _noteHorizonHistogram[i].TotalHeight = new_total_height;
                    break
```

```
                }
            }
        }
    }

    /// <summary>
    /// 수평 Histogram에서 잘래내기한 이미지를 만든다.
    /// </summary>
    private void ImageToBitamp()
    {
        _bmpNote = new Bitmap[_noteInformation.TotalCount];

        int osun_width = 0;
        int osun_height = 0;

        for (int i = 0; i < _bmpNote.Length; i++)
        {
            osun_height = _noteInformation.Downs[i].Y - _noteInformation.Tops[i].Y - 1;
            osun_width = _noteInformation.Downs[i].X - _noteInformation.Tops[i].X - 1;


            _bmpNote[i] = new Bitmap(osun_width, osun_height);

            for (int y = _noteInformation.Tops[i].Y + 1; y < _noteInformation.Downs[i].Y; y++)
            {
                for (int x = _noteInformation.Tops[i].X + 1; x < _noteInformation.Downs[i].X; x++)
                {
                    if (_bitmap.GetPixel(x, y).R == 0 && _bitmap.GetPixel(x, y).B == 0 &&
_bitmap.GetPixel(x, y).G == 0)
                    {
                        _bmpNote[i].SetPixel(x - (_noteInformation.Tops[i].X + 1), y -
(_noteInformation.Tops[i].Y + 1), Color.Black);
                    }
                    else
                    {
                        _bmpNote[i].SetPixel(x - (_noteInformation.Tops[i].X + 1), y -
(_noteInformation.Tops[i].Y + 1), Color.White);
                    }
                }
            }

             //   Save(_bmpNote[i], "ImageToBitamp" + i.ToString());
        }
    }



    //1차 음표들의 수직 히스토그램 표 만들기.

    /// <summary>
    /// 음표을 이용하여 수직 Histogram을 생성한다.
    /// </summary>
    private void ImageToCreateNoteHistogram()
    {
        _verticalHistogram = new VerticalHistogram[_noteInformation.TotalCount];

        for (int i = 0; i < _verticalHistogram.Length; i++)
        {
            _verticalHistogram[i] = new VerticalHistogram();
            _verticalHistogram[i].TotalWidth = _bmpNote[i].Width;

            for (int x = 0; x < _bmpNote[i].Width; x++)
            {
                for (int y = 0; y < _bmpNote[i].Height; y++)
                {
                    if(_bmpNote[i].GetPixel(x, y).R == 0 && _bmpNote[i].GetPixel(x, y).G == 0 &&
_bmpNote[i].GetPixel(x, y).B == 0)
                    {
```

```
                        _verticalHistogram[i].VeriticalHistogram[x]++;
                    }
                }
            }
        }
    }

    /// <summary>
    /// 음계를 분리 한다.
    /// </summary>
    private void ImageToScale()
    {
        int note_height = 0;
        int gap = 0;

        _scale.ScaleCount = _noteInformation.TotalCount;

        for (int i = 0; i < _scale.ScaleCount; i++)
        {
            note_height = _noteInformation.Downs[i].Y - 3;
            gap = ((SheetCheck)_arraySheet[0]).Gap[0];
            for (int osunC = 0; osunC < _sheetInformation.Length; osunC++)
            {
                SheetCheck sheetCheck = ((SheetCheck)_arraySheet[osunC]);
                if (_sheetInformation[osunC].Tops.Y <= note_height && note_height <=
_sheetInformation[osunC].Downs.Y)
                {
                    if (sheetCheck.Line[1] + 1 < note_height && note_height <= sheetCheck.Line[2] - 2)
                    {
                        _scale.ScaleHeight[i] = 77;
                    }
                    else if (sheetCheck.Line[2] - 2 < note_height && note_height < sheetCheck.Line[2] + 2)
                    {
                        _scale.ScaleHeight[i] = 76;
                    }
                    else if (sheetCheck.Line[2] + 1 < note_height && note_height <= sheetCheck.Line[3] - 2)
                    {
                        _scale.ScaleHeight[i] = 74;
                    }
                    else if (sheetCheck.Line[3] - 2 < note_height && note_height <= sheetCheck.Line[3] + 2)
                    {
                        _scale.ScaleHeight[i] = 72;
                    }
                    else if (sheetCheck.Line[3] + 1 < note_height && note_height <= sheetCheck.Line[4] - 2)
                    {
                        _scale.ScaleHeight[i] = 70;
                    }
                    else if (sheetCheck.Line[4] - 2 < note_height && note_height <= sheetCheck.Line[4] + 1)
                    {
                        _scale.ScaleHeight[i] = 69;
                    }
                    else if (sheetCheck.Line[4] + 2 <= note_height && note_height <=
sheetCheck.Line[4] + sheetCheck.Gap[3])
                    {
                        _scale.ScaleHeight[i] = 67;
                    }
                    else if (sheetCheck.Line[4] + sheetCheck.Gap[3] - 2 <= note_height && note_height
<= sheetCheck.Line[4] + sheetCheck.Gap[3] * 2 + 1)
                    {
                        _scale.ScaleHeight[i] = 65;
                    }
                    else
                    {
                        _scale.ScaleHeight[i] = 61;
                    }
                }
            }
        }
```

```
        }

        /// <summary>
        /// 박자를 확인 한다.
        /// </summary>
        private void ImageToBeat()
        {
            int length;

            int max_value; // 배열 원소중 최대값 저장
            int max_index; // 배열 원소중 최대값 가진 인덱스
            int osunC = 0;

            bool isBlack = false
            int x = 0;
            int tailCount = 0;



            SheetCheck sheetCheck = ((SheetCheck)_arraySheet[0]);

            for (int i = 0; i < _verticalHistogram.Length; i++)
            {
                if (i == _sheetInformation[osunC].NoteCount)
                {
                    osunC++;
                    sheetCheck = ((SheetCheck)_arraySheet[osunC]);
                }

                length = _verticalHistogram[i].VeriticalHistogram.Length;

                max_value = 0;
                max_index = 0;

                ///
                /// 음표 하나가 가지고있는 배열원소중 최대값 찾기
                ///
                for (int index = 0; index < _verticalHistogram[i].VeriticalHistogram.Length; index++)
                {
                    if (max_value < _verticalHistogram[i].VeriticalHistogram[index])
                    {
                        max_value = _verticalHistogram[i].VeriticalHistogram[index];
                        max_index = index;
                    }
                }

                if (max_index == _verticalHistogram[i].VeriticalHistogram.Length - 1)
                {
                    if (max_index == 0)
                        _scale.ScaleKind[i] = 0;
                    else
                    {
                        if (_verticalHistogram[i].VeriticalHistogram[max_index / 2] <= (sheetCheck.Gap[0]
+ 1) - 1)
                        {
                            _scale.ScaleKind[i] = 1;
                            _scale.ScaleLength[i] = 35;
                        }
                        else
                        {
                            _scale.ScaleKind[i] = 1;
                            _scale.ScaleLength[i] = 25;
                        }
                    }
                }
                else
                {
```

```
if (_verticalHistogram[i].VeriticalHistogram.Length < 5)
{
    if (_verticalHistogram[i].VeriticalHistogram[0] > 25)
    {
        _scale.ScaleKind[i] = 0;
        break
    }

    else
    {
        int helf;
        _scale.ScaleKind[i] = 3;
        helf = _scale.ScaleLength[i - 1] / 2;
        _scale.ScaleLength[i - 1] = _scale.ScaleLength[i - 1] + helf;
        _scale.ScaleKind[i] = 0;
    }// 점
}

else
{
    if (_verticalHistogram[i].VeriticalHistogram[0] > 5)
    {
        if (_verticalHistogram[i].VeriticalHistogram[0] > 20)
        {
            if (_verticalHistogram[i].VeriticalHistogram[0] > 25)
            {
                if (_verticalHistogram[i].VeriticalHistogram[1] > 25)
                {
                    _scale.ScaleKind[i] = 0;
                    break
                }
                else
                {
                    _scale.ScaleKind[i] = 1;
                    _scale.ScaleHeight[i] = 25;
                }
            }
            else
            {
                _scale.ScaleKind[i] = 0;
            }
        }
        else
        {

            if (_verticalHistogram[i].VeriticalHistogram[3] > 18)
            {
                _scale.ScaleKind[i] = 2;
                _scale.ScaleLength[i] = 25;
            }
            else
            {
                _scale.ScaleKind[i] = 0;
            }
        }
    }

    else
    {
        if (_verticalHistogram[i].VeriticalHistogram[5] <= 7)
        {
            if (_verticalHistogram[i].VeriticalHistogram[2] > 11)
            {
                _scale.ScaleKind[i] = 2;
                _scale.ScaleLength[i] = 25;
            }
            else
            {
```

```
                                if(max_index == _verticalHistogram[i].VeriticalHistogram.Length - 1 ||
                                    (_verticalHistogram[i].VeriticalHistogram[0] == 1 &&
_verticalHistogram[i].VeriticalHistogram[1] == 1 &&
                                    _verticalHistogram[i].VeriticalHistogram[2] == 1))
                                {
                                    x = max_index + 1;
                                    tailCount = 0;

                                    if(_bmpNote[i].GetPixel(x, 0).R == 0 && _bmpNote[i].GetPixel(x,
0).G == 0 && _bmpNote[i].GetPixel(x, 0).B == 0)
                                    { isBlack = true }
                                    else
                                    { isBlack = false }

                                    for (int y = 1; y < _bmpNote[i].Height / 2; y++)
                                    {
                                        //// 검 -> 검
                                        if (isBlack == true && _bmpNote[i].GetPixel(x, y).R == 0 &&
_bmpNote[i].GetPixel(x, y).G == 0 && _bmpNote[i].GetPixel(x, y).B == 0)
                                        { continue }

                                        //// 검 -> 흰
                                        else if (isBlack == true && _bmpNote[i].GetPixel(x, y).R ==
255 && _bmpNote[i].GetPixel(x, y).G == 255 && _bmpNote[i].GetPixel(x, y).B == 255)
                                        { isBlack = false tailCount++; }

                                        //// 흰 -> 검
                                        else if (isBlack == false && _bmpNote[i].GetPixel(x, y).R == 0
&& _bmpNote[i].GetPixel(x, y).G == 0 && _bmpNote[i].GetPixel(x, y).B == 0)
                                        { isBlack = true }

                                        //// 흰 -> 흰
                                        else if (isBlack == false &&_bmpNote[i].GetPixel(x, y).R ==
255 && _bmpNote[i].GetPixel(x, y).G == 255 && _bmpNote[i].GetPixel(x, y).B == 255)
                                        { continue }

                                    }

                                    if (tailCount == 1)
                                    {
                                        _scale.ScaleKind[i] = 1;
                                        _scale.ScaleLength[i] = 15;
                                    }
                                    else if (tailCount == 2)
                                    {
                                        _scale.ScaleKind[i] = 1;
                                        _scale.ScaleLength[i] = 10;
                                    }
                                    else
                                    {
                                        if (_verticalHistogram[i].VeriticalHistogram[max_index / 2] <=
(sheetCheck.Gap[0] + 1) - 1)
                                        {
                                            _scale.ScaleKind[i] = 1;
                                            _scale.ScaleLength[i] = 35;
                                        }// 2분 음표
                                        else
                                        {
                                            _scale.ScaleKind[i] = 1;
                                            _scale.ScaleLength[i] = 25;
                                        }// 4분 음표
                                    }
                                }
                                else
                                {
                                    _scale.ScaleKind[i] = 2;
                                    _scale.ScaleLength[i] = 15;
                                }
```

```
                            }
                        }

                        else
                        {
                            if ((_verticalHistogram[i].VeriticalHistogram[0] == 4 &&
                                _verticalHistogram[i].VeriticalHistogram[1] == 8) ||
                                _verticalHistogram[i].VeriticalHistogram[0] == 2)
                            {
                                _scale.ScaleKind[i] = 0;
                            }
                            else
                            {
                                x = max_index + 1;
                                tailCount = 0;

                                if(_bmpNote[i].GetPixel(x, 0).R == 0 && _bmpNote[i].GetPixel(x, 0).G
== 0 && _bmpNote[i].GetPixel(x, 0).B == 0)
                                { isBlack = true }
                                else
                                { isBlack = false }

                                for (int y = 1; y < _bmpNote[i].Height / 2; y++)
                                {
                                    //// 검 -> 검
                                    if (isBlack == true && _bmpNote[i].GetPixel(x, y).R == 0 &&
_bmpNote[i].GetPixel(x, y).G == 0 && _bmpNote[i].GetPixel(x, y).B == 0)
                                    { continue }

                                    //// 검 -> 흰
                                    else if (isBlack == true && _bmpNote[i].GetPixel(x, y).R == 255 &&
_bmpNote[i].GetPixel(x, y).G == 255 && _bmpNote[i].GetPixel(x, y).B == 255)
                                    { isBlack = false tailCount++; }

                                    //// 흰 -> 검
                                    else if (isBlack == false && _bmpNote[i].GetPixel(x, y).R == 0 &&
_bmpNote[i].GetPixel(x, y).G == 0 && _bmpNote[i].GetPixel(x, y).B == 0)
                                    { isBlack = true }

                                    //// 흰 -> 흰
                                    else if (isBlack == false && _bmpNote[i].GetPixel(x, y).R == 255
&& _bmpNote[i].GetPixel(x, y).G == 255 && _bmpNote[i].GetPixel(x, y).B == 255)
                                    { continue }

                                }

                                if (tailCount == 1)
                                {
                                    _scale.ScaleKind[i] = 1;
                                    _scale.ScaleLength[i] = 15;
                                }
                                else if (tailCount == 2)
                                {
                                    _scale.ScaleKind[i] = 1;
                                    _scale.ScaleLength[i] = 10;
                                }
                                else
                                {
                                    _scale.ScaleKind[i] = 0;
                                    _scale.ScaleLength[i] = 0;
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        #region SheetView
        /// <summary>
        /// 현재 소절 보여주기
        /// </summary>
        private void SheetPreView()
        {
            picMusicalNote.Image = _bmpSheet[_nSheetIndex];
            tblPlayIndex.Text = string.Format("{0} / {1}", _nSheetIndex + 1, _nTotalBar);
        }

        /// <summary>
        /// 이전 소절 듣기
        /// </summary>
        private void BackPlay()
        {
            if (_nSheetIndex - 1 >= 0 && _nSheetIndex - 1 < _sheetInformation.Length)
            {
                _nSheetIndex--;
                _nCurrent = _nSheetIndex;
                btnWhitePianoKeys[_nBeforeIndex].BackColor = Color.White;
                _nBeforeIndex = 0;
                _nTmrCurrent = 0;
                if (_nSheetIndex == 0)
                {
                    _nAfterIndex = 0;
                }
                else if (_nSheetIndex >= 1)
                {
                    _nAfterIndex = _sheetInformation[_nSheetIndex - 1].NoteCount;
                }
            }
            else
            {
                MessageBox.Show("악보 첫 소절입니다.");
            }

            picMusicalNote.Image = _bmpSheet[_nSheetIndex];
            tblPlayIndex.Text = string.Format("{0} / {1}", Convert.ToString(_nSheetIndex + 1),
_nTotalBar);
        }

        /// <summary>
        /// 다음 소절 듣기
        /// </summary>
        private void NextPlay()
        {
            if (_nSheetIndex + 1 >= 0 && _nSheetIndex + 1 < _sheetInformation.Length)
            {
                _nSheetIndex++;
                _nCurrent = _nSheetIndex;
                btnWhitePianoKeys[_nBeforeIndex].BackColor = Color.White;
                _nBeforeIndex = 0;
                _nTmrCurrent = 0;
                _nAfterIndex = _sheetInformation[_nSheetIndex - 1].NoteCount;
            }
            else
            {
                MessageBox.Show("악보 마지막 소절입니다.");
            }
            picMusicalNote.Image = _bmpSheet[_nSheetIndex];
            tblPlayIndex.Text = string.Format("{0} / {1}", Convert.ToString(_nSheetIndex + 1),
_nTotalBar);
        }
        #endregion
```

```
/// <summary>
/// 악보분석
/// </summary>
private void Analysis()
{
    //이진화및 히스토그램생성
    ImageToBinarization();
    ImageToHistogram();

    //Histogram을 참조하여 악보 이미지에 오선위치, 두께, 간격, 소절을 정보를 가지고 온다.
    ImageToInformation();

    //각 오선에 첫 줄 / 마지막 줄을 위치로 소절에 크기를 생성 한다.
    ImageToSheetPosition();

    //오선 삭제
    ImageToSheetDelete();

    //각 소절에 히스토그램 정보를 생성한다.
    ImageToBarHistogram();

    //소절을 단위로 생성
    ImageToBar();

    //음표 개수 파악
    ImageToNoteCount();

    //각 음표 위치 측정
    ImageToNotePosition();

    //각 음표를 잘라낸다.
    ImageToNoteCut();

    //수평 histogram으로 음표에 위, 아래를 잘라낸다.
    ImageToHoriHistogramCut();

    //잘라내기 한 후 이미지를 생성 한다.
    ImageToBitamp();

    //음표를 이용하여 수직 histogram을 한다.
    ImageToCreateNoteHistogram();

    //histogram을 이용하여 음계를 분리 한다.
    ImageToScale();

    //박자를 생성한다.
    ImageToBeat();

    //첫 번째 소절 보여주기
    SheetPreView();
}

private void MusicalNotePlay()
{
    tmrPlay.Enabled = true
}

private void MusicalNoteStop()
{
    tmrPlay.Enabled = false
    _nAfterIndex = 0;
    _nTmrCurrent = 0;

    btnWhitePianoKeys[_nBeforeIndex].BackColor = Color.White;

    _nBeforeIndex = 0;

    InitializeMIDI();
```

```
        SheetPreView();
    }

    private void MusicalNotePause()
    {
        if (tmrPlay.Enabled == false)
        {
            tmrPlay.Enabled = true
            MidiNoteOn(this._midiHandler, _scale.ScaleHeight[_nAfterIndex], _nVolume, 0);
        }
        else
        {
            tmrPlay.Enabled = false

            InitializeMIDI();
        }
    }

    #region MIDI
    private void InitializeMIDI()
    {
        CloseMIDI();

        if (WIN32.midiOutOpen(ref _midiHandler, 0, IntPtr.Zero, IntPtr.Zero, 0) != 0)
        {
            MessageBox.Show("장비오류", Text);
        }
        PlayMidi(0xC0, _nInstrument, 0, 0);
    }

    private void CloseMIDI()
    {
        WIN32.midiOutClose(_midiHandler);
    }

    private IntPtr PlayMidi(int stavovyBajt, int prvniDatovyBajt, int druhyDatovyBajt, int kanal)
    {

        int dwZprava = stavovyBajt | kanal | (prvniDatovyBajt << 8) | (druhyDatovyBajt << 16);
        return WIN32.midiOutShortMsg(_midiHandler, dwZprava);
    }

    private void button2_Click_1(object sender, EventArgs e)
    {
        MusicalNotePlay();
    }

    private IntPtr MidiNoteOn(IntPtr handleMIDIOut, int MIDIVyska, int rychlostStisku, int kanal)
    {
        return PlayMidi(0x090, MIDIVyska, rychlostStisku, kanal);
    }

    private IntPtr MidiNoteOff(IntPtr handleMIDIOut, int MIDIVyska, int rychlostUvolneni, int kanal)
    {
        return PlayMidi(0x080, MIDIVyska, rychlostUvolneni, kanal);
    }
    #endregion
```

```
    #region Menubar
    /// <summary>
    /// 이미지 불러오기
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuLoadMusicalNote_Click(object sender, EventArgs e)
    {
```

```
        Initialize();
        LoadImage();
    }

    /// <summary>
    /// 악보분석
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuAnalysisMusicalNote_Click(object sender, EventArgs e)
    {
        Analysis();
    }

    /// <summary>
    /// 악보 - 재생
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuPlay_Click(object sender, EventArgs e)
    {
        MusicalNotePlay();
    }

    /// <summary>
    /// 악보 - 일시정지
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuPause_Click(object sender, EventArgs e)
    {
        MusicalNotePause();
    }

    /// <summary>
    /// 악보 - 정지
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuStop_Click(object sender, EventArgs e)
    {
        MusicalNoteStop();
    }

    /// <summary>
    /// 프로그램 종료
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void mnuClose_Click(object sender, EventArgs e)
    {
        this.Close();
    }
    #endregion
```

```
    #region toolbar
    private void tlbInstrumentList_SelectedIndexChanged(object sender, EventArgs e)
    {
        _nInstrument = tlbInstrumentList.SelectedIndex;

        InitializeMIDI();
    }

    private void tlbVolumeList_SelectedIndexChanged(object sender, EventArgs e)
    {
        _nVolume = Convert.ToInt32(tlbVolumeList.SelectedItem);
```

```
        }

        private void tlbBackPlay_Click(object sender, EventArgs e)
        {
            BackPlay();
        }

        private void tlbNextPlay_Click(object sender, EventArgs e)
        {
            NextPlay();
        }
        #endregion

-------------------------------------------------------------------------------------------------------------

        #region Timer
        private void tmrPlay_Tick(object sender, EventArgs e)
        {
            if (_nAfterIndex == _scale.ScaleCount)
            {
                tmrPlay.Enabled = false
                _nCurrent = 0;
                _nBeforeIndex = 0;
                _nAfterIndex = 0;
                _nTmrCurrent = 0;

                _nSheetIndex = 0;
                SheetPreView();
            }
            else if (_nAfterIndex == _sheetInformation[_nCurrent].NoteCount && _nCurrent + 1 <
_sheetInformation.Length)
            {
                _nSheetIndex = _nCurrent + 1;
                SheetPreView();
            }


            if ((_nAfterIndex != _scale.ScaleCount) && _scale.ScaleLength[_nAfterIndex] == _nTmrCurrent)
            {
                MidiNoteOff(this._midiHandler, _scale.ScaleHeight[_nAfterIndex], _nVolume, 0);
                btnWhitePianoKeys[_nBeforeIndex].BackColor = Color.White;
                _nAfterIndex++;
                _nTmrCurrent = 0;
                return
            }
            else if ((_nAfterIndex != _scale.ScaleCount) && _scale.ScaleKind[_nAfterIndex] == 0)
            {
                _nAfterIndex++;
            }
            else if ((_nAfterIndex != _scale.ScaleCount) &&_scale.ScaleKind[_nAfterIndex] == 1 &&
_nTmrCurrent == 0)
            {
                MidiNoteOn(this._midiHandler, _scale.ScaleHeight[_nAfterIndex], _nVolume, 0);

                for (int i = 0; i < btnWhitePianoKeys.Length; i++)
                {
                    int ntag = Convert.ToInt32(btnWhitePianoKeys[i].Tag);

                    if (ntag.Equals(_scale.ScaleHeight[_nAfterIndex]))
                    {
                        _nBeforeIndex = i;
                        btnWhitePianoKeys[i].BackColor = Color.Orange;
                        break
                    }
                }
            }
            else if ((_nAfterIndex != _scale.ScaleCount) && _scale.ScaleKind[_nAfterIndex] == 2 &&
_nTmrCurrent == 0)
```

```
            {
                MidiNoteOff(this._midiHandler, _scale.ScaleHeight[_nAfterIndex], _nVolume, 0);
                btnWhitePianoKeys[_nBeforeIndex].BackColor = Color.White;
            }

            _nTmrCurrent++;

            if (tmrPlay.Enabled == false)
            {
                _nAfterIndex = 0;
                _nTmrCurrent = 0;
            }
        }
        #endregion

    }//FrmMain class
```

```
    #region VerticalHistogram
    public class VerticalHistogram
    {
        private int _nTotalWidth = 0;
        private int[] _nVerticalHistogram = new int[0];

        public VerticalHistogram()
        {
            TotalWidth = 0;
        }

        public int TotalWidth
        {
            get
            {
                return _nTotalWidth;
            }
            set
            {
                _nTotalWidth = value

                VeriticalHistogram = new int[TotalWidth];
            }
        }

        public int[] VeriticalHistogram
        {
            get
            {
                return _nVerticalHistogram;
            }
            set
            {
                _nVerticalHistogram = value
            }
        }
    }
    #endregion
```

```
    #region NoteHorizonHistogram
    public class NoteHorizonHistogram
    {
        private int _nTotalHeight = 0;
        private int[] _nHorizonHistogram = new int[0];

        public NoteHorizonHistogram()
        {
```

```
            TotalHeight = 0;
        }

        public int TotalHeight
        {
            get
            {
                return _nTotalHeight;
            }
            set
            {
                _nTotalHeight = value

                HorizonHistogrma = new int[TotalHeight];
            }
        }

        public int[] HorizonHistogrma
        {
            get
            {
                return _nHorizonHistogram;
            }
            set
            {
                _nHorizonHistogram = value
            }
        }
    }
    #endregion
```

```
    #region SheetInformation
    public class SheetInformation
    {
        private Point _ptTop = new Point();
        private Point _ptDown = new Point();

        private int _noteCount = 0;
        private int _sheetCount = 0;
        private int[] _nVeritcalHistogram = new int[0];

        public Point Tops
        {
            get
            {
                return _ptTop;
            }
            set
            {
                _ptTop = value
            }
        }

        public Point Downs
        {
            get
            {
                return _ptDown;
            }
            set
            {
                _ptDown = value
            }
        }

        public int NoteCount
```

```
        {
            get
            {
                return _noteCount;
            }
            set
            {
                _noteCount = value
            }
        }

        public int SheetCount
        {
            get
            {
                return _sheetCount;
            }
            set
            {
                _sheetCount = value
            }
        }

        public int[] VericalHistogram
        {
            get
            {
                return _nVeritcalHistogram;
            }
            set
            {
                _nVeritcalHistogram = value
            }
        }


    }
    #endregion
```

```
    #region SheetCheck
    public class SheetCheck
    {
        private int[] _nGap = new int[4];
        private int[] _nLine = new int[5];
        private int[] _nWidth = new int[5];

        public SheetCheck()
        {
            Gap = new int[4];
            Line = new int[5];
            Width = new int[5];
        }

        public int[] Gap
        {
            get
            {
                return _nGap;
            }
            set
            {
                _nGap = value
            }
        }

        public int[] Line
```

```
        {
            get
            {
                return _nLine;
            }
            set
            {
                _nLine = value
            }
        }

        public int[] Width
        {
            get
            {
                return _nWidth;
            }
            set
            {
                _nWidth = value
            }
        }
    }
    #endregion
```

```
    #region NoteInformation
    /// <summary>
    /// 음표 정보를 저장 한다. (음표개수, 위치(사각형)등 저장)
    /// </summary>
    public class NoteInformation
    {
        private int _nTotalCount = 0;
        private Point[] _ptTop = new Point[0];
        private Point[] _ptDown = new Point[0];

        public NoteInformation()
        {
            TotalCount = 0;
        }

        public int TotalCount
        {
            get
            {
                return _nTotalCount;
            }
            set
            {
                _nTotalCount = value

                Tops = new Point[TotalCount];
                Downs = new Point[TotalCount];
            }
        }

        public Point[] Tops
        {
            get
            {
                return _ptTop;
            }
            set
            {
                _ptTop = value
            }
        }
```

```
        public Point[] Downs
        {
            get
            {
                return _ptDown;
            }
            set
            {
                _ptDown = value
            }
        }
    }
    #endregion
```

```
    #region Scale
    /// <summary>
    /// 악보 이미지에 음표, 박자에 정보를 가지고 있는다.
    /// </summary>
    public class Scale
    {
        private int _nScaleCount = 0;
        private int[] _nScaleKind = new int[0];
        private int[] _nScaleHeight = new int[0];
        private int[] _nScaleLength = new int[0];

        public Scale()
        {
            ScaleCount = 0;
        }

        public int ScaleCount
        {
            get
            {
                return _nScaleCount;
            }
            set
            {
                _nScaleCount = value

                ScaleKind = new int[ScaleCount];
                ScaleHeight = new int[ScaleCount];
                ScaleLength = new int[ScaleCount];
            }
        }

        public int[] ScaleKind
        {
            get
            {
                return _nScaleKind;
            }
            set
            {
                _nScaleKind = value
            }
        }

        public int[] ScaleHeight
        {
            get
            {
                return _nScaleHeight;
            }
            set
```

```
            {
                _nScaleHeight = value
            }
        }

        public int[] ScaleLength
        {
            get
            {
                return _nScaleLength;
            }
            set
            {
                _nScaleLength = value
            }
        }
    }
    #endregion
}
```

별첨2.진행보고서

종합설계 진행보고서

제출일 : 2009 년 3 월 16 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 03 월 01 일 ~ 2009 년 03 월 15 일	
진행사항	1. 팀 구성, 지도교수 배정 2. 팀 오리엔테이션 - 팀원의 관심분야, 특기분야(역량) 파악. - 팀원간 친밀도 증진. 3. 팀명 선정 : Jugend(“젊음”을 뜻하는 독일어) - 각 팀원이 팀명을 제안한 뒤 투표를 거쳐 선정. 4. 주제 선정 : 이미지 분석을 응용한 악보인식과 미디어 제작. - 팀원별로 주제선정 이후 다른 팀원들에게 발표. - 브레인스토밍(brainstorming) 과정을 거쳐 선정.	
참여인력별 진행사항	이탁곤	매주 회의 진행. 주제 선정 및 발표.
	함충민	오리엔테이션, 팀명 선정 참가. 주제 선정 및 발표.
	김도균	오리엔테이션, 팀명 선정 참가. 주제 선정 및 발표.
	오원균	오리엔테이션, 팀명 선정 참가. 주제 선정 및 발표.
문제점	각 팀원의 관심사와 요구사항이 모두 달랐기 때문에 주제 선정 과정이 길었다. 팀원간의 갭을 줄이기 위해 자신이 생각해온 것을 다른 팀원에게 발표하는 과정을 거쳤지만 주제 선정 및 의견 조율에 난항을 겪었으며 결국 지도교수님에게 조언을 구함으로서 해결했다.	
기타	팀 클럽 개설 : club.cyworld.com/Jugend1 회의일시 : 2009년 3월 11일.	

종합설계 진행보고서

제출일 : 2009 년 3 월 30 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 03 월 16 일 ~ 2009 년 03 월 30 일	
진행사항	**1. 요구사항 분석** - 선정된 주제에 대한 요구사항 분석. **2. 프로젝트 계획 수립** - 프로젝트 종료(10개월)까지의 계획수립. - 간트 차트(Gantt chart) 이용. **3. 역할 분담** - 오리엔테이션시 얻은 팀원 간 장단점, 관심사, 능력을 고려해 역할분담. **4. 제약사항 확인** - 현실적 제약 조건 확인. - 구현 가능 여부 확인. **5. 자료조사**	
참여인력별 진행사항	**이탁곤**	매주 회의 진행. 역할 분담.
	함충민	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	김도균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	오원균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
문제점	“실제로 구현이 가능한가?” 실제로 우리 팀의 능력이 명확히 수치화 된 것이 아니기 때문에 토론시 의견조율이 쉽지 않은 부분 이였다. 회의 결과 어느 정도 프로젝트 규모가 조정되었으며 요구사항에 대한 만족여부는 프로젝트 완성 이후 테스트 및 평가 과정에서 알 수 있을 것이다.	
기타	회의일시 : 2009년 3월 18일, 3월 25일. 참고서적 : 디지털 영상처리(허영호 외 3명), 영상처리 이론과 실제(Randy crane)	

종합설계 진행보고서

제출일 : 2009 년 9 월 15 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 09 월 02 일 ~ 2009 년 09 월 13 일	
진행사항	**1. 설계 바탕 자료 수집** - 설계를 바탕으로 참고 할 수 있거나, 도움이 될 만한 자료수집 및 공유 **2. 변경 사항 조정** - 최초 사용 계획했던 프로그램의 진행과정에서 문제 발생에 따라 차후 진행과정에 대한 회의 - 사용 프로그램을 Java에서 C#으로 조정. **3. 스킨 부분 디자인 회의** - 프로그램의 스킨 디자인에 대한 회의. **4. 자료조사**	
참여인력별 진행사항	**이탁곤**	매주 회의 진행. 역할 분담.
	함충민	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	김도균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	오원균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
문제점	초기 계획시 구현하려고 했던 프로그램 구성의 자료 부족과 운용능력 부제로 계획의 차질 발생. 기초 프로그램 변경하기로 결정. 이진화 부분에서 Debugging문제 발생.	
기타	회의일시 : 2009년 9월 4일, 9월 8일. 참고 Site : http://cafe.naver.com/opencv	

종합설계 진행보고서

제출일 : 2009 년 9 월 29 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 09 월 14 일 ~ 2009 년 09 월 28 일	
진행사항	**1. 기존 코딩 소스 변환** - 기존에 구성해 둔 Java 코드를 C#에 맞게 변환시킴 **2. 스킨 디자인 코딩** - 이전 주에 결정한 디자인 소스에 따라 스킨 소스를 코딩 시작 - 옵션 및 기타 부분에 대한 세부 사항 회의 **3. 자료조사**	
참여인력별 진행사항	**이탁곤**	매주 회의 진행. 역할 분담.
	함충민	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	김도균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	오원균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
문제점	전체적인 구성은 비슷하나 몇몇 차이점에 따라서 기존 제작된 코드를 C#에 맞도록 재 편집함. 스킨 구성시 버튼의 크기와 폰트가 다르게 적용되는 문제 발생.	
기타	회의일시 : 2009년 9월 16일, 9월 22일. 참고 Site : http://cafe.naver.com/opencv	

종합설계 진행보고서

제출일 : 2009 년 10 월 12 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 09 월 30 일 ~ 2009 년 10 월 11 일	
진행사항	1. Body 부분의 코딩 시작 - 기존에 구성해 둔 Java 코드를 C#에 맞게 변환 이후에 추가적으로 재구성된 코드를 제작하기 시작. - 이진화의 구성으로 이미 대부분의 작업은 처리가 된 상태임. 2. 스킨 보정 및 버그 수정 - 기능의 삭제와 추후 연결 문제에 따라 스킨 다지아니 소폭 수정 - 스킨에 글씨 폰트 문제점 수정. 3. 자료조사	
참여인력별 진행사항	이탁곤	매주 회의 진행. 역할 분담.
	함충민	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	김도균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	오원균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
문제점	코딩중 중간 문제점 체크 과정에서 MS framework와 기타 여건에 의해서 시동 차체가 안 될 수 있다고 판단됨. 이진화 과정에서 분석 작업에 순서 지정에 대한 의견 충돌로 회의 실행. 스킨 폰트 문제점 수정.	
기타	회의일시 : 2009년 9월 30일, 10월 6일. 참고 Site : http://cafe.naver.com/opencv	

종합설계 진행보고서

제출일 : 2009 년 10 월 27 일

( )조	설계주제	이미지 분석을 응용한 악보인식과 미디어 제작
조원명단	이탁곤(52067386), 함충민(52001504), 김도균(52067372), 오원균(52067383)	
지도교수	정용주 교수님	
진행기간	2009 년 10 월 14 일 ~ 2009 년 10 월 26 일	
진행사항	**1. 전체 코드의 연결 및 연주 부분 코딩 시작** - 악보 분석 후 이진화된 정보를 분석하여 기존에 입력해 둔 값과 동일하게 추정되는 값의 경우 지정된 키를 연주하도록 코딩. - 기존에 구성해둔 스킨 코드와 키 음원 간의 연결 코딩. **2. 자료조사**	
참여인력별 진행사항	**이탁곤**	매주 회의 진행. 역할 분담.
	함충민	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	김도균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
	오원균	요구사항 분석, 역할분담, 제약사항, 자료조사 참여.
문제점	전반적인 프로젝트 진행의 속도가 더디고, 생각 보다 인식률이 떨어질 것으로 판단되어지고 있음. 완실한 모습을 패턴 인식하기 보다 검은 점의 수로 인식하는 경향이 더 강하다고 생각됨.	
기타	회의일시 : 2009년 10월 14일, 10월 20일. 참고 Site : http://cafe.naver.com/opencv	