**Music- Score player**

****

**휴먼ICT 소프트웨어 공학 Team 6**

**──────────────────────**

|  |  |
| --- | --- |
| 분반 | 월 7,8,9 |
| 교수 | 송인식 교수님 |
| 팀 | Team 6 |
| 팀원 | 20141261 송제웅  20140786 조장연  20140661 신주환  20141001 이지호  20141359 정유석  20146235 김정훈 |

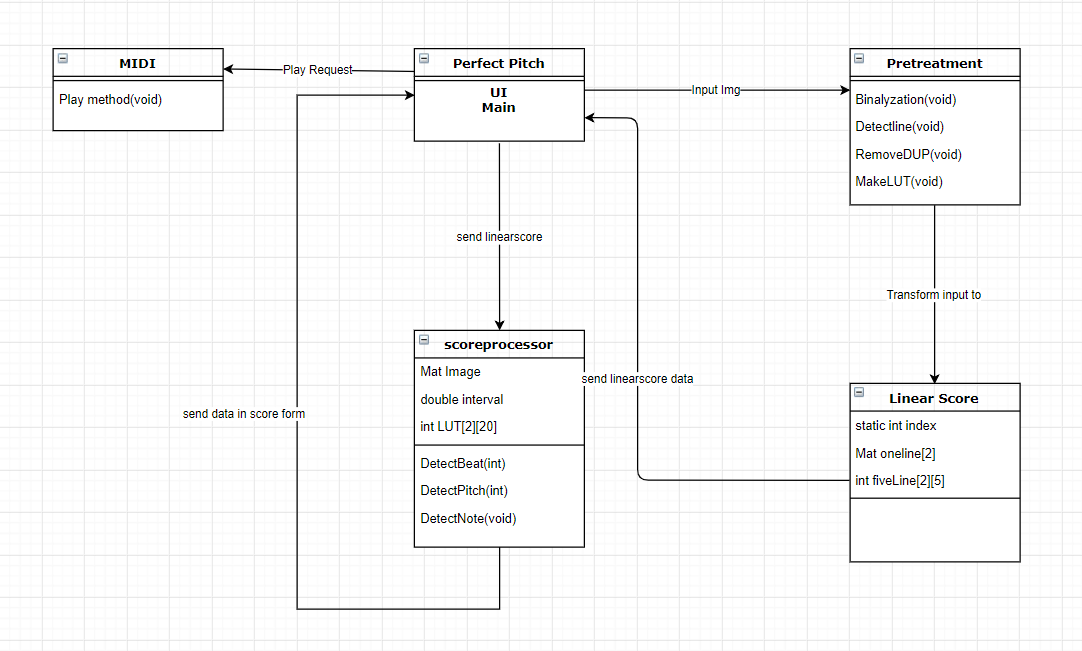
**목차**

**1. System Architecture**

**2. Process**

1. System Architecture

1) Program Class Diagram



- **AdminManager Class** : GUI를 통하여 관리자가 쉽게 프로그램을 사용할 수 있도록 하는 중심 Class이다. 해당 Class를 통하여 전반적인 프로그램이 구동되며 CCTV 화면을 모니터링 하거나 주차공간을 Mapping 하는 등의 프로그램 전반에 대한 Event가 정의된 Class

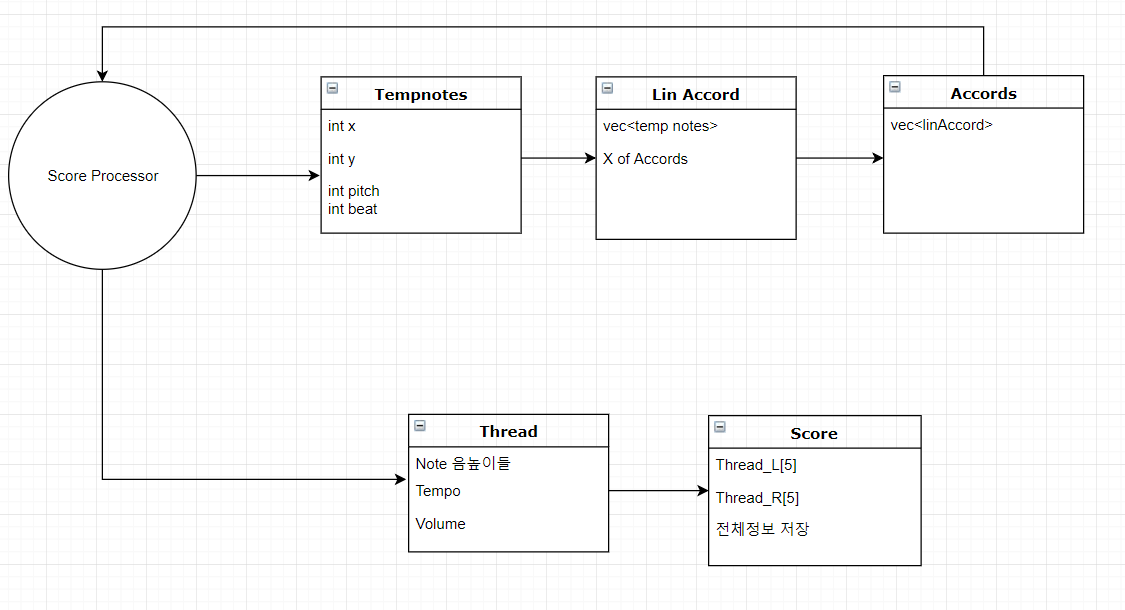
- **MappingFileManager Class** : Mapping File을 관리하고 저장하는 기능을 담당하는 Class로 프로그램 시작 시 checkRecentFile 메소드를 통하여 기존에 저장된 Mapping File이 있는지 확인하고 이를 불러올 수 있도록 한다.

- **ParkingPosition Class** : Mapping 위치를 좌상단, 우하단 좌표로 저장한다.

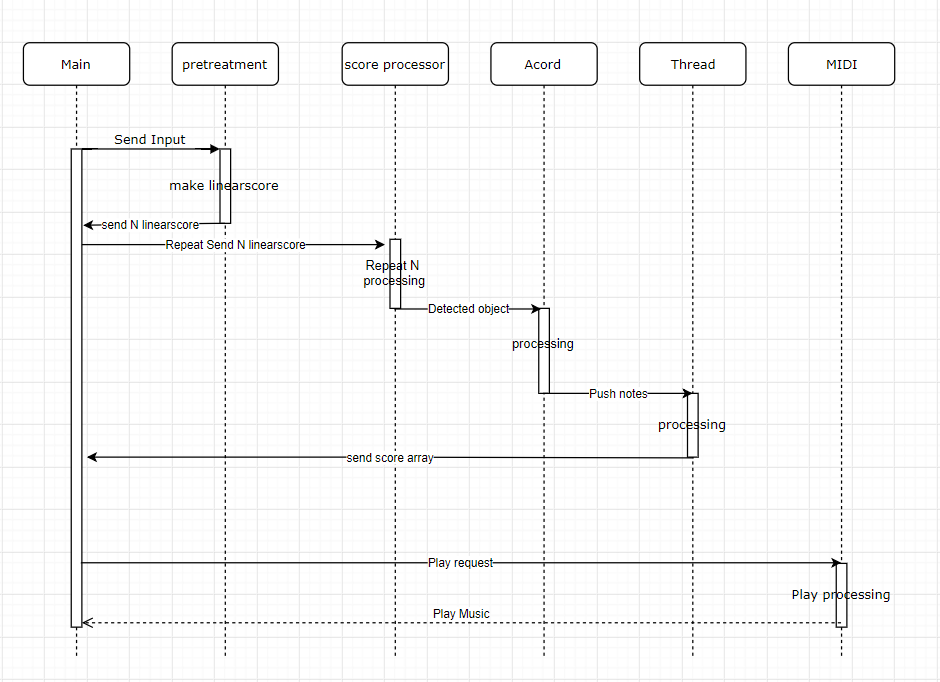
- **ParkingInfo Class**: 관리자가 로그인 시 관리자의 주차장 정보를 DB로 부터 불러와 객체를 생성하고 관리자 정보를 불러오거나 수정이 용이하도록 하기 위한 Class

- **Camera Class** : 연결된 Camera를 관리하는 Class로 각 카메라와 무선으로 연결되므로 Camera의 IP와 각 카메라의 화면에 담긴 주차공간 수를 관리하기 위한 Class

2) Database Structure Diagram



3) Program Sequence Diagram



-Main

2. Process

**[Risk Assessment]**

1. 기술적 문제

* 팀 구성원들이 처음 접해보는 언어나 기술이 많다. 이 위험 관리 방법으로는 팀 구성원들이 각자 파트를 잘 분배하여 자신이 맡은 부분은 책임지고 공부하여 완성시키는 것이다. 가장 많은 경험을 갖고 있는 사람을 팀 리더로 정하였다.

1. 팀 구성원들의 일정 문제

* 팀원들 모두 개인적 일정 및 다른 과목 수업 및 프로젝트가 있기 때문에 각자 맡은 일을 못 할 가능성이 있다. 그렇기 때문에 개인 작업 외에도 매 주 팀원들 모두가 모여 진행 상황을 파악하도록 한다.

1. 악보 인풋 선명도 문제

* 악보의 인풋이 틀어져 있거나 선명하지 않을 경우 영상처리가 제대로 되지 않거나 잘못된 음을 연주 할 수 있다.
* 지속적인 테스트를 통하여 오차를 줄일 수 있도록 한다.

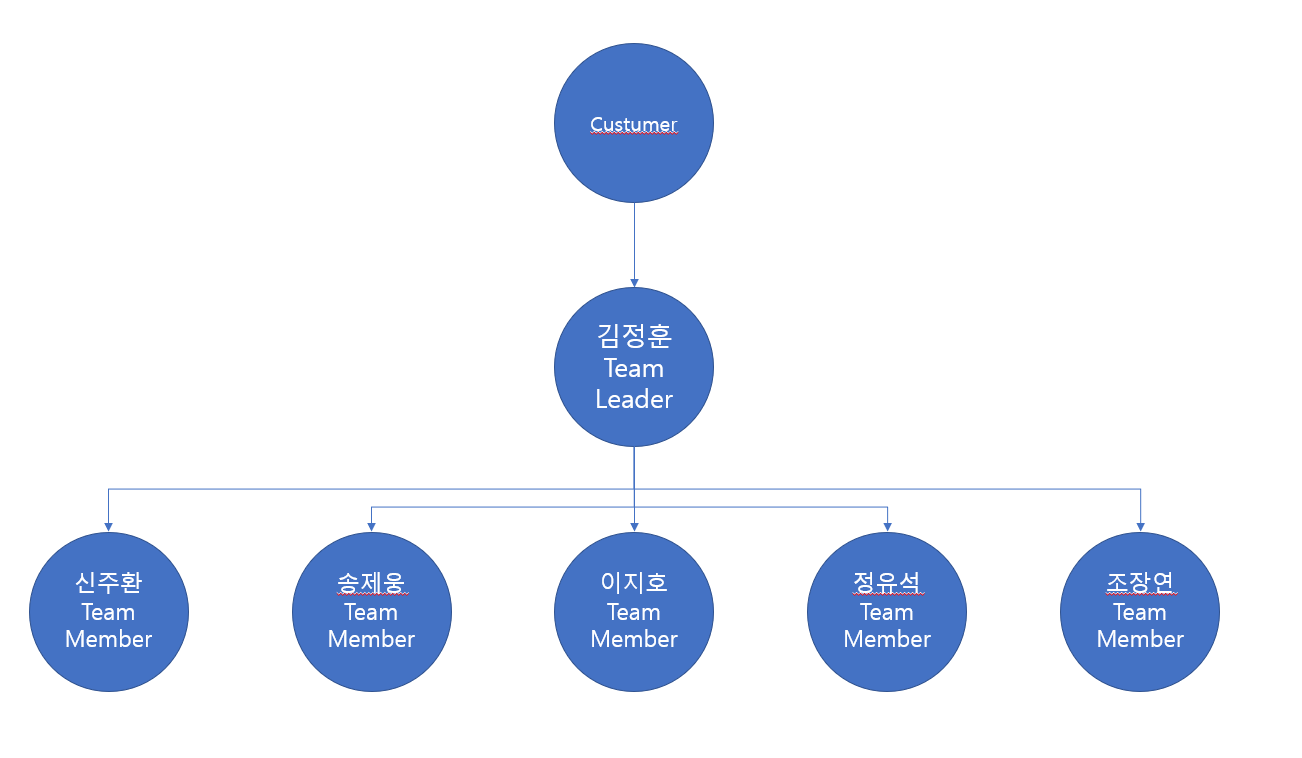
1. 악보 인식률 문제

* 하나의 음표라도 인식이 틀리게 되면 쓰레드가 밀려 정상적인 연주가 불가능하다.
* 지속적인 테스트와 디버깅을 통하여 인식률을 가능한 한 높게 유지시킨다.

**[Project Schedule]**

|  |  |  |
| --- | --- | --- |
| 4월 | 자료구조 및 클래스 설계 | 전체 |
| 5월 1주 | 음악 재생을 위한 midi 구현Thread를 이용한 화음 재생 기능 구현 | 송제웅 , 정유석 |
| 5월 2주 | 전처리 | 이지호, 신주환 |
| 5월 3주 | Head/tail탐색을 통해 음높이 및 박자 탐색 | 송제웅, 정유석, 이지호, 신주환 |
| 5월 4주 | 샵/플랫/내츄럴, 이음줄/붙임줄, 되돌이표 구현 | 이지호, 신주환, 조장연, 김정훈 |
| 6월 | UI처리 | 김정훈, 조장연 |

**[Team Structure]**



**[Test Plan]**

부분별로 나눠 테스트를 진행한다.

**A. Unit test**

-Open CV

1.악보의 이미지를 제대로 이진화 시키는가?

2.이진화 시킨 악보의 음표머리, 꼬리, 오선을 제대로 분류 하는가?

3.전처리한 정보를 DataBase에 잘 저장 시키는가?

- MIDI

1. DataBase에 저장된 정보를 토대로 음악을 재생하는가?

2. 화음을 위한 다중 쓰레드의 동기화가 제대로 되었는가?.

3. 악기의 변경이 의도한대로 작동 하는가?

4. 재생속도의 변화에 따른 쓰레드의 동기화가 제대로 갱신 되었는가?

- DataBase

1. 의도했던 만큼의 효율적인 형태를 유지하는가?

2. 필요한 정보가 모두 저장되어 있는가?

- Console

1. 사용자 편의에 맞게 알맞은 GUI를 제공하는가?

2. 필요한 모든 정보를 DataBase에서 효율적으로 불러오는가?

**B. System testing**

1. Running Time 이 효율적인가?

2. DataBase의 저장공간을 파악하고 있는가? 또한, 저장공간 초과시 어떻게 하는가?

3. System이 외부요소에 의해 부하가 걸릴 일은 없는가?

**[Document Plan]**

1. **Cover page** : Music-Score Player에 관하여 문서화 최신화 날짜, 팀원을 표기.
2. **History** : revision history 를 통해 지난 버젼에서 무엇이 바뀌었는지, 무엇을 작성하였는지 표기.
3. **Contact information** : 각각 requirements 를 구현하는 사람들을 표기.
4. **Summary** : 시스템의 목적은 악기 연주 연습 시 간편하게 청사진을 구현해 주는 것.
5. **Audience** : 악기 연주에 관심이 있는 모든 사람들을 목표로 함.
6. **Staff** : 팀원 전체가 문서화에 참여함.
7. **Time table** : Requirements 에 plan description 으로 표기.
8. **Risk factors** :  Requirements 에 Process description - risk summary 에 표기. 실제 구현 과정 후 테스트를 통해 해결방법 표기.

**[Coding Style Guidelines]**

# 1. 이름 규칙

**1.1 - 변수의 표기는 헝가리안 표기법을 사용합니다.**

(헝가리안 표기법이 타이핑하기 불편하거나 귀찮을 수도 있으나 근본적으로 변수의 명확성을 강화합니다.)

int nNumber, char szName, CString strInfo, CSurface surSportsCar, int \*pnData, CWnd wndMain

**1.2 - 각 변수 및 함수의 이름은 대문자로 시작합니다.**

(소문자만을 이용하거나 '\_'을 이용하거나 또는 소문자 + 첫 단어 대문자를 이용하는 많은 사용 예가 있기는 하지만 결국 여러 스타일을 동일 프로젝트에서 혼용하여 사용하는 것은 통일성과 가독성 등에 저해를 가져옵니다. 때문에 다양한 방법 중 한가지를 선택할 필요성을 느꼈고, 첫 문자를 대문자로 쓰는 것으로 통일하였습니다.)

int nTypeMax, char szFileInfomation, (O)

int ntype\_max, char szfile\_infomation (X)

int GetType(); BOOL SetMaxRange( int CX, int CY ), void SetFont( HFONT hFont )

# 2. 일반

**2.1 - bool, true, false 는 사용하지 않고 BOOL, TRUE, FALSE 로 사용합니다.**

(BOOL등은 4바이트 자료형이고 bool등은 1비트(실제로는 1바이트) 자료형입니다. 32비트 운영체계의 경우 속도와 비교하는데 걸리는 성능 부분에도 유리하고 알아보기도 좋기 때문에 위와 같은 대문자로 된 자료형을 사용합니다.)

**2.2 - 한 라인이 너무 길지 않도록 주의합니다.**

(정확히 컬럼의 크기가 정해진 것은 아니지만 일반적인 프로젝트 팀에서 사용하는 일반적인 해상도(1024~1280)에서 한눈에 보기 좋도록 작성합니다.)

**2.3 - 성능 향상 및 가독성 증가를 위해서 본 가이드라인은 위배 될 수 있습니다. (단! 확고한 근거가 존재해야 합니다.)**

(가이드라인의 목표는 보기 좋고 통일된 코드작성이 가장 중요한 목표입니다. 특수한 경우 가독성이나 성능 향상을 위해서 더 좋은 방법이 있습니다. 그러한 경우에는 본 가이드라인을 따르지 않습니다.)

# 3. 클래스

**3.1 - 클레스의 기본 구조는 다음과 같습니다.**

(생성자, 공개함수, 비공개함수, 멤버변수의 순서로 배열하고 각각은 public, protected 등으로 추가로 명기하여 구분합니다. )

class CSound

{

public:                        : public 함수를 최 상단에 배치합니다.

CSound();                   : 생성자를 최 상단에 배치합니다.

virtual ~CSound();      : 종결자는 일반적으로 virtual 입니다.

void Play();                 : 멤버 함수

void Stop();

: public, protected 등은 1라인 비웁니다.

protected:                  : protected 함수를 배치합니다.

void PlayWav();

void PlayOgg();

: 각각의 함수별 관련성 및 블록이 있다면 1라인 비웁니다.

void StopWav();

void StopOgg();

protected:                   : 멤버 변수를 배치합니다.

char m\_szFileName[256];

int m\_nSize;

};

**3.2 - 모든 멤버 변수는 public으로 선언하지 않습니다.**

(정보은닉과 캡슐화를 위해서 멤버 변수들은 public으로 선언하지 않습니다.)

# 4. 파일 및 디렉토리

**4.1 - 클래스명과 파일명은 'C'를 제외한 동일한 이름을 가집니다.**

클래스명 : CFileName

파일명 : FileName.h, FileName.cpp

**4.2 - 하나의 클래스 2개의 파일(.h .cpp)을 가집니다.**

**4.3 - 클래스의 복잡도에 따라서 여러 개의 cpp 파일을 가질 수 있습니다.**

(종종 멤버 함수가 많다면 함수의 검색 및 수정 등이 불편하여 관리가 어려워집니다. 그러한 경우 비슷한 함수 별로 파일을 만든다면 쉽게 관리가 가능합니다.)

클래스명 : CSurfaceGdi

파일명 : SurfaceGdi.h, SurfaceGdiPut.cpp SurfaceGdiInit.cpp, SurfaceGdiEffect.cpp