프로젝트 결과보고서

1. 프로젝트 개요

o 기본정보

프로젝트 명	스마트카 레이싱 게임 개발
참여 학생	김현익, 박진, 황석현
개발 기간	2013. 09. 11 ~ 2013 12. 3

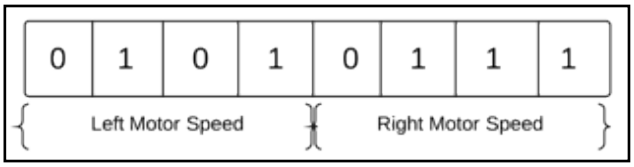
o SW개발 환경

구분	세부 내용
사용언어	Phython, C++, Java, Arduino Script
HW	한백전자 스마트카
실습장비	갤럭시 노트 10.1
기타	cocos2dx, Google App Engine 사용

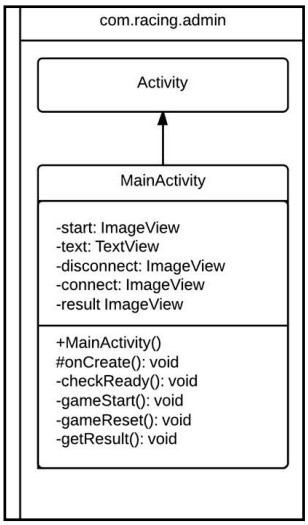
o 시스템 구조도

시스템 구조도

1. 펌웨어 속도 패킷 구조



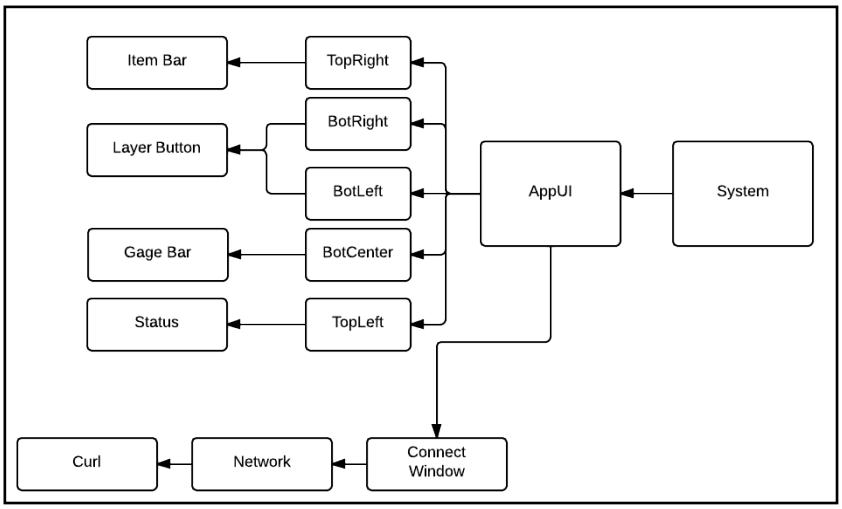
2. 어드민 앱 구조



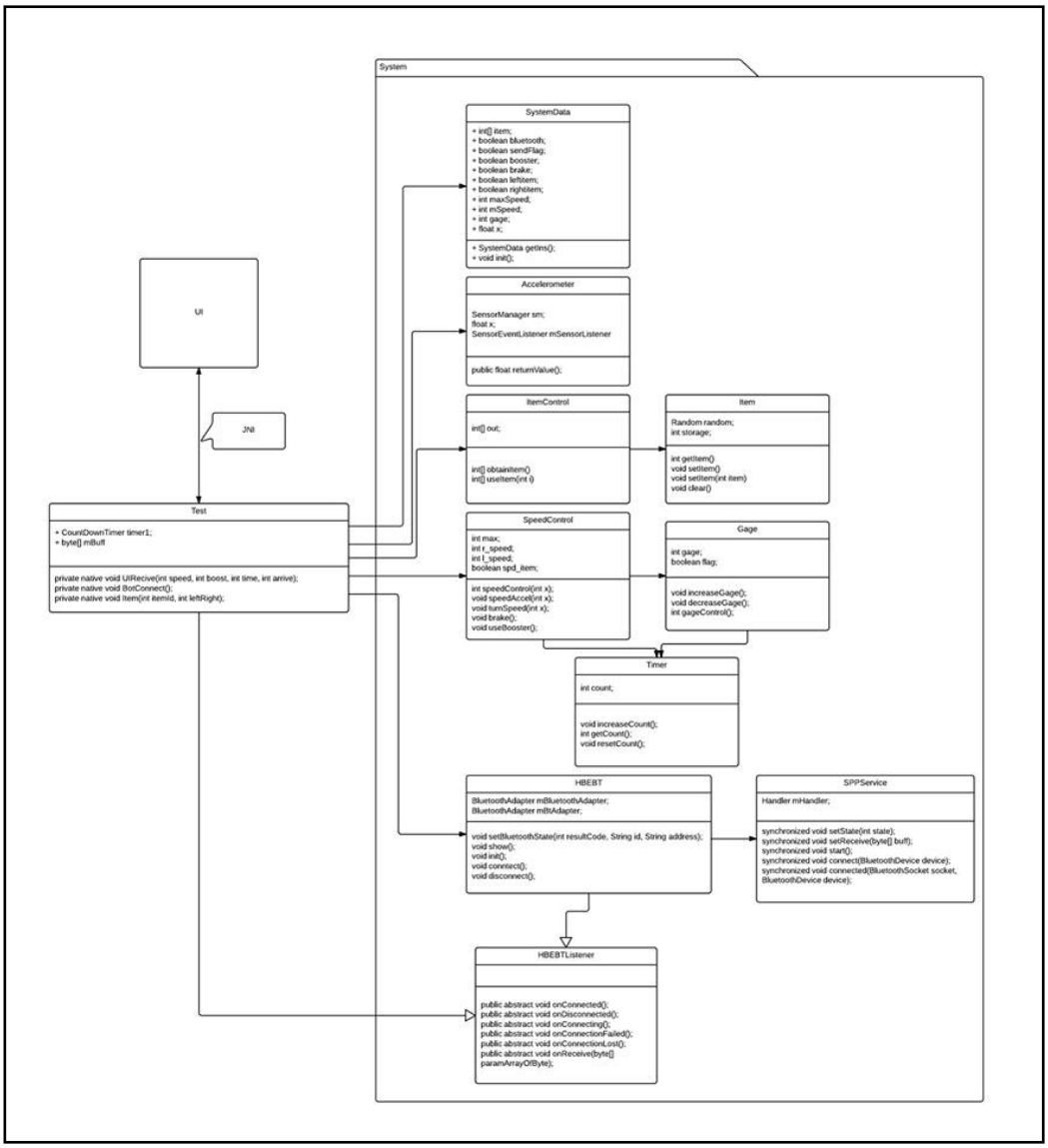
3. 웹 서버 구조

1. http://dkumse02.appspot.com/
   설명 : 현재 서버의 스테이터스를 한눈에 보여줍니다.
2. http://dkumse02.appspot.com/admin/reset
   설명 : 서버의 데이터를 모두 초기화 시킵니다.
3. http://dkumse02.appspot.com/admin/start
   유저 1과 유저 2 모두가 레디를 했다면 start가 출력됩니다.
   만약 한명이라도 레디가 되어 있지 않다면 fail이 출력됩니다.
4. http://dkumse02.appspot.com/admin/getarrive
   유저들의 도착을 판별합니다.
   만약 스마트카1 도착, 스마트카2가 도착하지 않았다면,
   bot 1 : arrive
   bot 2 : running
   라고 뜨게 됩니다.
5. http://dkumse02.appspot.com/admin/getarrivetime
   유저들의 도착시간을 보여줍니다.
   출력은 아래와 같습니다.
   bot 1 : 32
   bot 2 : 50
6. http://dkumse02.appspot.com/admin/getwin
   어떤 유저가 이겼는지 보여줍니다.
   1번 유저가 이겼다면 bot 1을 출력, 아니라면 bot 2를 출력합니다.
   만약 두 유저 모두 같은 시간에 도착하였다면 draw를 출력합니다.
7. http://dkumse02.appspot.com/admin/check_all_ready
   플레이어 모두 레디를 한 상태인지 체크합니다.
   모두 레디를 하였다면, all_ready, 아니라면 fail을 출력합니다.
8. http://dkumse02.appspot.com/user/getbotid
   플레이어가 로그인 합니다. 순서에 따라 해당 아이디를 서버에서 보내주며, 플레이어는 이 아이디를 계속 가지고 있어야 합니다.
9. http://dkumse02.appspot.com/user/ready?botid=%d
   해당 아이디에 따라 레디 합니다.
   성공시 on ready를, 실패하였다면 ready fail을 출력합니다.
   실패한 경우에는, 미리 로그인을 안할 경우가 있습니다.
10. http://dkumse02.appspot.com/user/start
    어드민이 출발을 눌렀는지 안눌렀는지 체크합니다.
    출발 상태라면 start를 출력하고 아니라면 fail을 출력합니다.
11. http://dkumse02.appspot.com/user/arrive?botid=%d&time=%d
    도착한 유저의 아이디와 시간을 등록합니다.
    플레이어가 도착점에 도착하면 사용합니다.
    제대로 된 아이디가 등록되었다면 success를, 아니라면 fail을 출력합니다.
12. http://dkumse02.appspot.com/user/win?botid=%d
    유저가 이겼는지 졌는지 출력해줍니다. 이겼다면 win을 졌다면 lose를 출력합니다.
    만약 도착 시간이 같다면 draw를 출력하며 아이디가 틀릴 경우에는 fail을 출력합니다.

4. 스마트카 - 플레이어 구조 UI



5. 스마트카 - 플레이어 구조 System



o 팀원 역할분담

팀원명	주요 역할
김현익	어드민 앱 개발, 트랙 제작
박진	프로젝트 설계 및 관리, 스마트카-플레이어 UI 개발, 웹서버 개발
황석현	스마트카 펌웨어 수정, 스마트카-플레이어 시스템 개발, 트랙 제작

o 추진 일정

세 부 과 제	월별 프로젝트 일정							
	년도	2013						
	월	9		10		11		12
기획								
개발								
문서 작성								

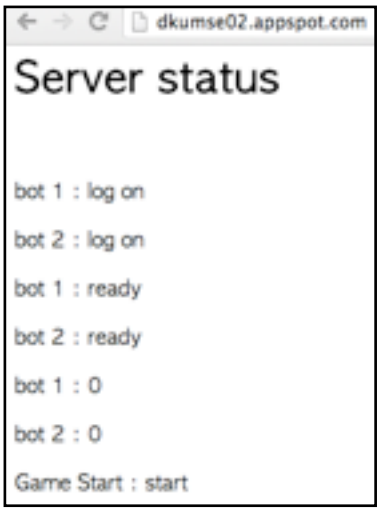
2. 추진성과

o 기술능력

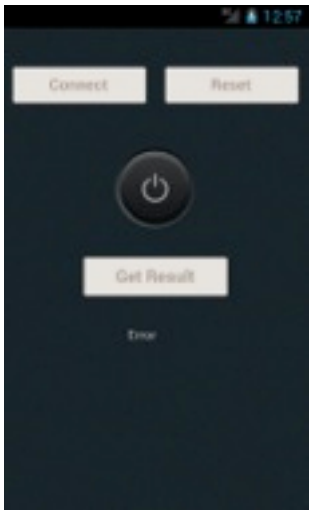
1. **프로젝트 구현**

처음에 정한 기획에서는 어느정도 구현된 상태입니다. 스마트카의 크기가 크다보니, 2인용으로 구현은 하였지만 스마트 카의 크기가 예상보다 너무 큰 상태였기 때문에, 2인형태로 트랙 세팅을 할 수 없었습니다. 그래서 트랙의 크기를 약간 줄이고 아이템을 싱글 플레이만 가능하도록 일부분만 구현하였고, 서버는 처음 기획대로 2인용으로 전부 가능하도록 구현하였습니다.

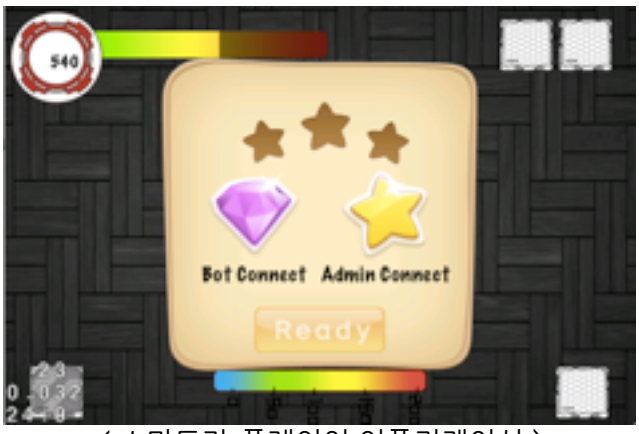
아래는, 현재까지 진행한 프로젝트의 스크린샷 및 사진입니다.



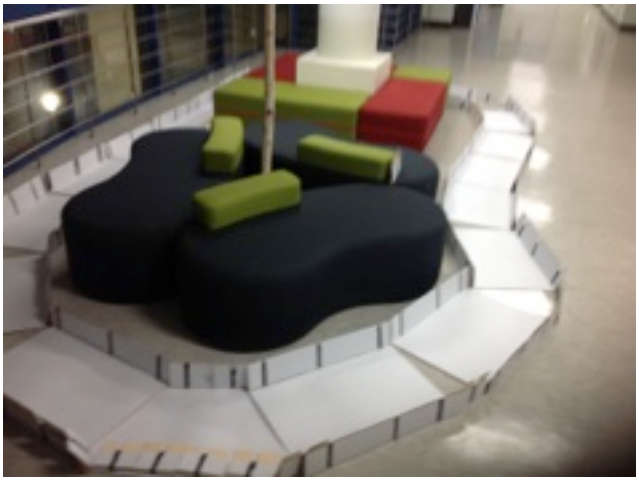
〈 웹 서버 현재 상태 표시 〉



〈 어드민 앱 〉



〈 스마트카-플레이어 어플리케이션 〉


〈 트랙 사진 〉

2. **오류 해결**

펌웨어에 적외선 센서 값을 읽어와서 어플로 전송하여 어플에서 값을 출력해보는 구문을 추가하였습니다. 이 때 예상되는 출력값은 흰색 위에 있을경우는 600이상의 값이, 검정색 위에 있을때는 200~300 정도의 값이어야 하는데, 출력되는 값은 흰색위에 있어도, 검은색 위에 있어도 비슷한 150~250 정도의 값만 계속 출력되었습니다.

그래서 검정색과 흰색을 판별하는 threshold를 정하기가 애매해서 고민하고 있었는데 패킷이 char형으로 전송이 된다는 것을 확인하고 사이즈문제로 255보다 큰 값은 패킷에 손실이 생겨서 이러한 값이 출력된다는 것을 알게 되었습니다.

그래서 적외선 센서로 읽어들인 값을 쉬프트 연산을 이용하여 2개로 나누어 2패킷에 걸쳐 전송하여 안드로이드 어플에서 값을 출력해본 결과 예상값과 비슷하게 나오는 것을 확인하였습니다.

그래서 최종적으론 펌웨어 내에서 400보다 큰 값을 읽어들인경우 1을 전송하고 아닐 경우엔 0을 전송하여 흰색일경우엔 1, 검정색일경우엔 0의 값을 전송하여 적외선 센서로 바닥을 인식하다가 8개 모든 센서가 검정색을 인식 즉, 0이 전송되었을 경우 finish로 인식하고 주행을 정지하도록 구현 하였습니다.

o 개발 지식

1. **전문 지식**

Cocos2dx

cocos2dx는 iOS, And, Win32, Blackberry 등 다양한 플랫폼에서 사용가능한 오픈소스 2D 게임 엔진입니다. 언어는 기본적으로 C++을 지원하며, 상황에 따라 루아같은 스크립트 언어들을 지원합니다. 저희는 이 엔진을 통하여 스마트카-플레이어 앱의 UI 개발 시간을 줄이기 위하여 사용하였습니다.

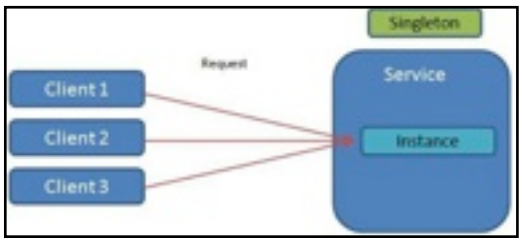
2. **신규 습득**

Google App Engine

Google App Engine은, 구글에서 2008년 발표한 웹 서버 엔진입니다.
언어는 대표적으로 Java와 Phython을 제공합니다.
이 웹 서버 엔진이 특이한 점은, 굳이 특별한 교육이 없어도 지원하는 언어와 몇 장 안되는 개발 문서만 읽으면 쉽게 개발할 수 있습니다. 또한, 클라우드 서버 또한 구글에서 지원해주기에 공인IP를 따로 사용할 필요가 없습니다. 주로 소규모 팀 프로젝트에서 프로토타입 개발용으로 선택 합니다.

3. **이론의 실무적용**

싱글톤은 디자인 패턴종류 중의 하나로 클래스를 오직 하나의 인스턴스만 생성하여 어디서 접근을 하던지 통일되게 제공하는 형식입니다. 싱글톤 패턴은 클래스 자체가 자신의 유일한 인스턴스로의 접근 방법을 자체적으로 관리하므로 전역 변수를 이용하여 하나의 인스턴스만을 만든 클래스에 접근하는 방법보다 훨씬 쉽게 관리할 수 있습니다.

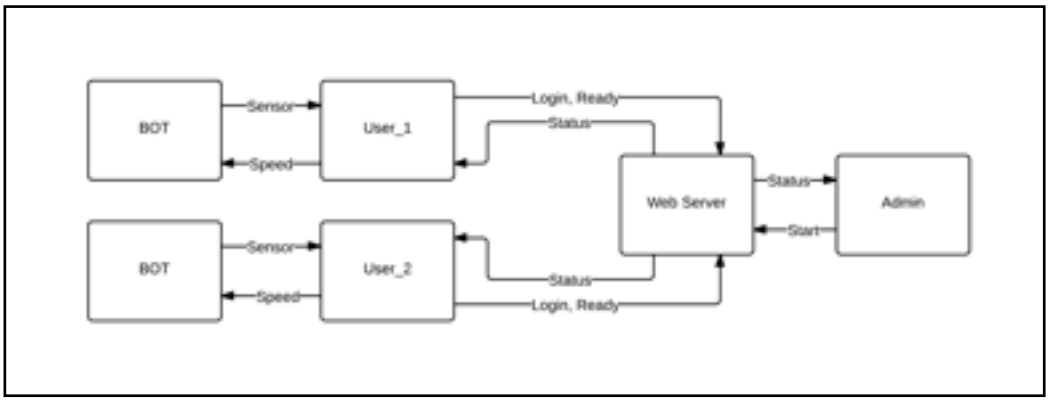


3. 기획 능력

o 프로젝트 설계

시스템 구조

아래 그림은 현재 프로젝트가 간단히 구동되는 방식을 간단하게 표현된 그림 입니다.



유저가 각각 한명씩 존재하고, 유저들은 어드민과 통신하기 위하여 웹 서버를 거치는 구조입니다. 아래서부터는 이 구조의 핵심만 간략하게 설명하고 넘어가도록 하겠습니다.

1. 웹서버

우선 첫 기획때, 웹 서버라는 존재가 없고, 그 자리에 Admin이 들어가서 직접 User들과 통신하는 구조로 기획하였습니다.

하지만, 직접 개발을 하게 되면서 어드민과 유저와의 통신 개발이 예상과 달리 오래 걸리게 되었습니다. 그래서 실시간으로 어드민과 유저의 통신보다는 우선 전체적인 프로젝트의 완성도를 높혀야겠다고 생각이 들어 현재 통신을 갈아엎고 웹 서버로 대체하게 되었습니다.

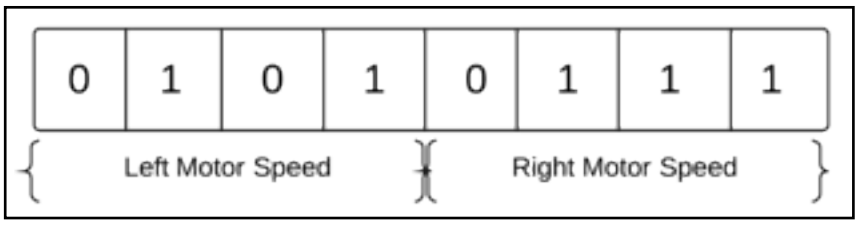
물론, 웹 서버가 기존 통신보다는 실시간으로 처리한다거나 응답 속도가 늦습니다. 하지만, 저희가 프로젝트를 진행하고 시연하고 마무리를 할 장소는 빠른 Wifi가 전역으로 깔려있는 학교이기 때문에 이 단점은 문제가 되지 않았습니다.

그래서 이러한 구조로 어드민과 플레이어가 웹서버를 통하여 서로 통신하게 되었습니다.

2. 스마트카 - 속도 패킷

저희는 프로젝트를 진행하면서 스마트카의 바퀴 속도를 유동적으로 설정하여 자연스러운 컨트롤과 부드러운 속도를 표현하고 싶었기에 약간의 펌웨어와 속도 패킷 데이터를 약간 수정하게 되었습니다.

아래는, 속도 패킷(1byte)를 어떻게 효율적으로 쓸 것인지에 대한 그림입니다.

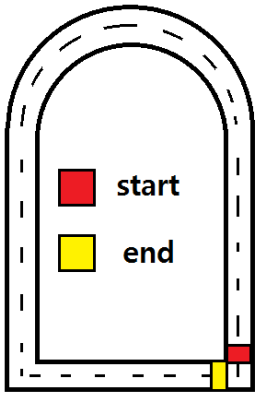


그냥 그림만 봐도 어떤 구조인지 쉽게 알아보실 수 있을 것 입니다.

그림 그대로 4비트씩 쪼개서 값을 사용합니다. 즉, 한 바퀴당 0~15 단계의 속도를 가집니다. 이런 방식으로 스마트카의 회전을 더욱 유동적으로 조절할 수 있도록 펌웨어를 약간 수정하였습니다.

트랙

트랙은 먼저 전체적으로 하드보드지로 만들고 트랙의 설치 장소는 국제관 복도 중앙에 설치되어 있는 쇼파를 주변으로 트랙을 설치하였습니다. 트랙은 전체적으로 실제 자동차 운행의 느낌을 살릴 수 있도록 곡선구조를 넣어주었고 제일 많이 꺾여 있는 부분은 90도까지 꺾이도록 하였습니다. 전체적인 트랙의 구조는 아래와 같습니다.

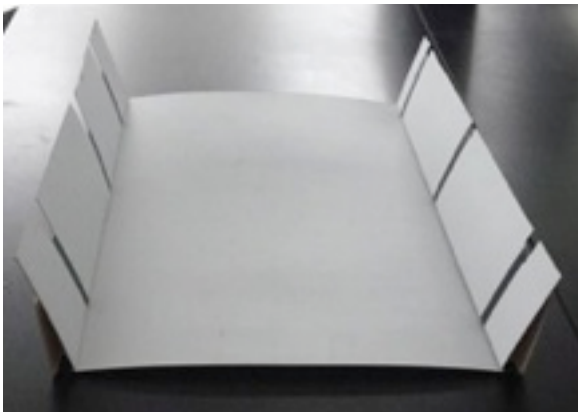


1. 직선 트랙

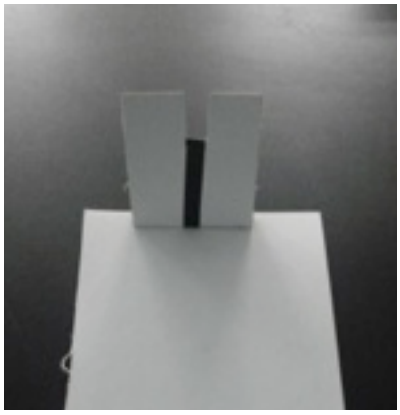
트랙의 직선구조는 2절 하드보드지 한 장을 밑바닥으로 사용하고 다른 한 장을 반으로 나누어 다음 사진과 같이 삼각형 구조로 벽을 세워 밑 바닥에 붙였습니다.

2. 곡선 트랙

트랙의 곡선구조는 곡선으로 벽을 세우기가 힘들어 벽은 직선구조에서 사용한 방법과 같은 방법을 사용하되 길이를 짧게 만들어 최대한 곡선의 느낌을 낼 수 있도록 배치하였습니다.

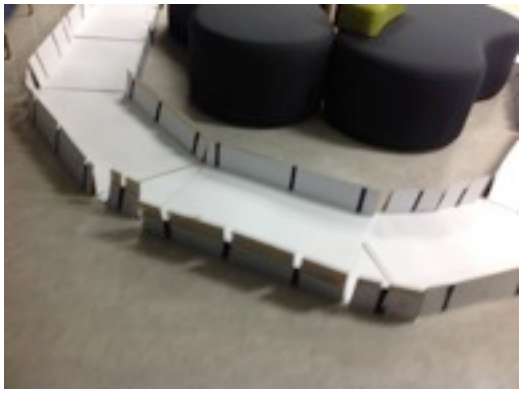


〈 직선 트랙 〉



〈 곡선 트랙 〉

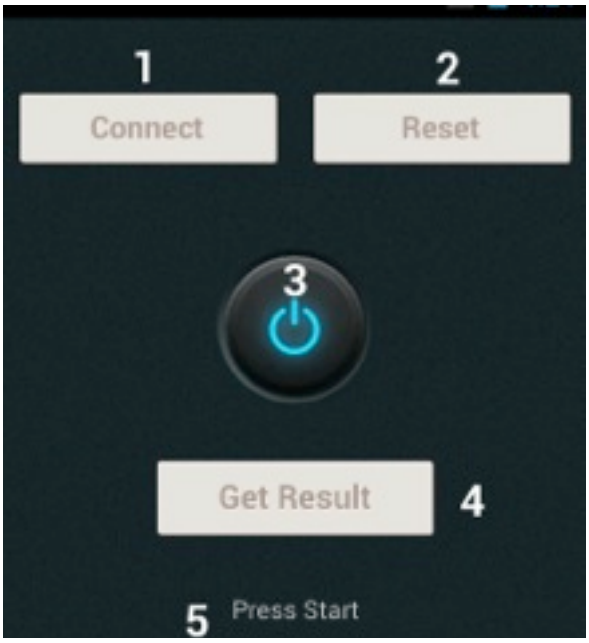
위의 트랙 구조를 조립하면 아래와 같은 형태의 트랙으로 만들 수 있게 되었습니다.



o 프로젝트 설계

어드민 UI

플레이어들의 출발을 담당하는 어드민 어플리케이션의 UI 입니다.



1. Connect 버튼: 서버로 부터 스마트카이 레디가 되어있는지 여부를 체크합니다.
2. Reset 버튼: 게임이 끝난 후 서버를 리셋합니다.
3. Start 버튼: 플레이어들이 동시에 시작할 수 있도록 서버에 명령을 내립니다.
4. Get Result 버튼: 누가 이겼는지를 서버로부터 받아옵니다.
5. Text 레이블: 앱 내부 함수의 결과를 표시하는 textview 입니다.

스마트카 - 플레이어 UI

스마트카을 조작하는 플레이어 어플리케이션의 UI 입니다.

1. 속도 스테이터스
   스마트카의 속도와 관련된 스테이터스 창 입니다. 속도에 따라 오른쪽과 원 주위의 게이지가 증감됩니다. 물론, 숫자도 변합니다.

2. 스마트카 연결
   이 아이콘을 누르면 스마트카과 연결할 수 있습니다.

3. 어드민 연결
   이 아이콘을 누르면 웹 서버와 통신하여 로그인을 시도합니다.

4. 아이템 창
   아이템을 휙득하게 되면 이 창에 아이템이 차곡차곡 쌓입니다. 사용은 아이템 창을 누르면 바로 사용하실 수 있습니다.

5. 브레이크 버튼
   브레이크 버튼 입니다. 이 버튼을 누르면 스마트카은 서서히 속도 0까지 속도가 감소 됩니다.

6. 부스터 버튼
   부스터 버튼 입니다. 이 버튼을 누르면 부스터를 사용하실 수 있습니다.

7. 부스터 게이지
   부스터의 게이지 입니다. 이 게이지에 따라 부스터를 사용하실 수 있습니다.

8. 레디 버튼
   레디 버튼 입니다. 스마트카과 어드민 연결 세팅이 모두 성공이라면 누르실 수 있습니다.

4. 구현 설명

1. 스마트카 - 플레이어 웹 서버 통신

```
CURL_DATA data;
const char *baseUrl = "http://dkumse02.appspot.com/user/ready?botid=%d"
char url[256];
sprintf(url, baseUrl, CommData::GetInstance()->botId);

if(network->connectHttp(url, &data) == CURLE_OK)
{
    string content = data.pContent;

    if(content == "ready fail")
        CCMessageBox("로그인 실패 하셨나 본데요?\n", "알림!");
```

이렇게 씁니다. 만들어둔 Network 라는 클래스를 사용하여 웹 서버와 통신하였습니다. 해당 클래스의 구조는 아래와 같으며, curl을 사용하여 구현하였습니다.

```
struct CURL_DATA {
    char    *pContent;
    int      size;

    CURL_DATA()
    {
        pContent    = (char*)malloc(1); //어짜피 reall
        size        = 0;
    }

    ~CURL_DATA()
    {
        free(pContent);
        pContent = NULL;
    }
};
```

위는 CURL 구조 입니다.

그냥 간단하게 메모리 얻어 두고 맨 위의 호출에 알아서 생성합니다.

```
class Network
{
private:
    CURL        *m_pCTX;

public:
    Network();
    ~Network();

private:
    static int _WriteCurlDataCallback(void *ptr, int size,
                                      int nmemb, void *pData);

public:
    CURLcode    connectHttp(const char *url, CURL_DATA *pData);

public:
    int             GetResponseCode();
    std::string     GetContentType();
    double          GetSize();
    double          GetSpeed();
};
```

위는 Network 클래스 구조입니다. connectHttp를 통하여 검색을 하는게 주가 됩니다. 그 외에는 리스폰 코드나 사이즈, 속도를 얻어오는 기능을 따로 얻어와서 구현하였습니다.

2. JNI 연결

시스템과 연결하는 부분은 JNI를 사용하여 작업하였습니다. 제가 맡은 파트가 UI였기 때문에, 필요한 연산처리는 자바에서 하도록 하였습니다.

원래 라면 UI를 자바에서 하고 연산은 Native에서 하는게 정상이긴 하지만, 학교에서 처음으로 제대로 하는 팀 프로젝트이였고, 팀원이 쓸 줄 아는 언어가 Java밖에 없었기에 이러한 방식으로 작업하게 되었습니다.

```
void ConnectWindow::botfunc(cocos2d::CCObject *pSender)
{
    if(connectBot)
        return;

#if (CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID )
    JniMethodInfo info;

    if(JniHelper::getStaticMethodInfo(info, "dku/mse/TestApp/Test", "ConnectBot",
        "()V"))
    {
        info.env->CallStaticVoidMethod(info.classID, info.methodID);
        info.env->DeleteLocalRef(info.classID);
    }
#endif
}
```

3. 웹 서버 구조

웹 서버는 구글 앱 엔진에서 파이썬을 사용하여 구현하였습니다. 일반적인 웹 페이지 띄우는 것 자체는 어려운 것이 아니니, 제외하고 db를 사용하여 데이터를 관리하는 방법에 대하여 자세히 기술하도록 하겠습니다.

```
class GameData(ndb.Model):
    bot_1_id = ndb.IntegerProperty()
    bot_1_ready = ndb.IntegerProperty()
    bot_1_time = ndb.IntegerProperty()

    bot_2_id = ndb.IntegerProperty()
    bot_2_ready = ndb.IntegerProperty()
    bot_2_time = ndb.IntegerProperty()

    game_start = ndb.IntegerProperty()
    findKey = ndb.StringProperty()
```

위의 코드는, db를 사용하기 위해 일단 데이터를 세팅하는 코드 입니다.

구글 앱 엔진에서는 db에 접근하거나, 새로 저장할때 미리 이러한 함수로 미리 세팅을 해 줘야 합니다. 아래는, 이 클래스를 이용하여 데이터를 관리하는 클래스에 대하여 소개하도록 하겠습니다.

```
class GameDataMgr:
  def dataClear(self):
      ndb.delete_multi(GameData.query().fetch(keys_only=True))

  def dataReset(self):
      self.dataClear()
      data = GameData(bot_1_id=0, bot_1_ready=0, bot_2_id=0, bot_2_ready=0, game_start=0,
      data.put()

  def loadDB(self):
      query = GameData.query()
      query = GameData.query(GameData.findKey == 'db')
      self.queryData = query.fetch(1)
      self.bot_1_id = self.parseValue('bot_1_id')
      self.bot_2_id = self.parseValue('bot_2_id')
      self.bot_1_rdy = self.parseValue('bot_1_ready')
      self.bot_2_rdy = self.parseValue('bot_2_ready')
      self.game_start = self.parseValue('game_start')
      self.bot_1_time = self.parseValue('bot_1_time')
      self.bot_2_time = self.parseValue('bot_2_time')
```

```
  def parseValue(self, findVariable):
      queryString = str(self.queryData)
      startPos = queryString.find(findVariable)+len(findVariable)+1
      endPos = queryString.find(',', startPos)

      if endPos == -1:
          endPos = queryString.find(')]', startPos)

      strlength = endPos - startPos
      strnum = ""

      for i in range(strlength):
          strnum += queryString[startPos+i]

      return int(strnum)
```

우선 구현 함에 있어서, db에서 가져온 값을 하나의 변수 자체로 뺴내는 방법을 찾질 못했습니다. 그래서 일단 급한 김에 쿼리문을 통하여 얻은 결과 자체를 문자열로 변환하였고, 얻은 문자열을 파싱하여 데이터를 얻는 방법을 택하였습니다.

일단, db에서 얻을 값들이 모두 정수형이였기에, 간단하게 해결할 수 있었습니다.

해당 위의 함수들은 문자열을 파싱하여 얻는 함수와 db를 리셋하여 청소시키는 함수로 이루어져 있습니다.

아래는, db를 업데이트하는 함수입니다.

```
  def updateDB(self):
      bot1id = self.bot_1_id
      bot2id = self.bot_2_id
      bot1rdy = self.bot_1_rdy
      bot2rdy = self.bot_2_rdy
      bot1time = self.bot_1_time
      bot2time = self.bot_2_time
      gs = self.game_start

      self.dataClear()

      data = GameData(bot_1_id=0, bot_1_ready=0, bot_2_id=0, bot_2_ready=0, game_start=0, bo
      data.bot_1_id = int(bot1id)
      data.bot_2_id = int(bot2id)
      data.bot_1_ready = int(bot1rdy)
      data.bot_2_ready = int(bot2rdy)
      data.game_start = int(gs)
      data.bot_1_time = int(bot1time)
      data.bot_2_time = int(bot2time)

      data.put()
```

우선 db에 데이터를 올리려면 ndb를 상속받은 객체에 put함수를 호출해야합니다.

그런데 저는 찾은 데이터를 수정하는 방법을 몰랐기에 일단 급한대로, db자체 내용을 전부 비우고 새로 작성하는 방법을 사용하였습니다.

어짜피 대규모 데이터도 아니고 하나 삭제 하나 추가이기에 속도는 그렇게 느리진 않았기에 그냥 이런 방식으로 구현하였습니다.

마지막으로, 아래는 웹 서버의 url 구조들입니다.

```
app = webapp2.WSGIApplication([
    ('/', MainHandler),
    ('/user/getbotid', GetBotID),
    ('/user/ready', Ready),
    ('/user/start', GetStart),
    ('/user/arrive', Arrive),
    ('/user/win', GetUserWin),
    ('/admin/reset', GameDBReset),
    ('/admin/start', AdminStart),
    ('/admin/getarrive', GetArrive),
    ('/admin/getarrivetime', GetArriveTime),
    ('/admin/check_all_ready', CheckAllReady),
    ('/admin/getwin', GetAdminWin),
], debug=True)
```

4. 어드민 앱 개발

```
package com.racing.admin;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;


public class MainActivity extends Activity{
    private ImageView start;
    private TextView text;
    private ImageView disconnect;
    private ImageView connect;
    private ImageView result;
```

먼저, 서버와 http로 통신하기 위해서 필요한 부분들과 액티비티 구성에 필수적인 요소들을 import 하였습니다. Admin 어플리케이션은 하나의 액티비티로 구성되어있으며 객체들을 함수간에 쉽게 접근할 수 있게 하기 위해 맨 윗부분에 선언하였습니다.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/re
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:orientation="vertical"
    android:background="@drawable/background" >

   <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="25dp"
        android:orientation="horizontal" >

        <ImageView
            android:id="@+id/button_connect"
            android:layout_width="140dip"
            android:layout_height="50dip"
            android:layout_margin="10dip"
```

이것이 admin 액티비티 레이아웃의 xml 구조 입니다. 레이아웃은 기본적으로 linearlayout을 통해구성했습니다. 특히, 버튼의 경우에는 안드로이드에서 기본적으로 제공하는 버튼으로는 디자인을 마음대로 바꿀 수 없어 ImageView를 사용하였습니다.

```
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


     text = (TextView)findViewById(R.id.textView);

    connect = (ImageView)findViewById(R.id.button_connect);
    connect.setOnClickListener(new Button.OnClickListener(){

        public void onClick(View v){
            text.setText("Connect");
            new Thread (new Runnable() {

                @Override
                public void run() {
                    // TODO Auto-generated method stub
                    try {
                        checkReady();
                        Log.d("CHECK", "checkReady");
                    } catch (IOException e) {
                        // TODO Auto-generated block
                        e.printStackTrace();
                    }
                }
            }).start();

        }
    });

    disconnect = (ImageView)findViewById(R.id.button_disconnect);
    disconnect.setEnabled(false);
    disconnect.setFocusable(false);


    disconnect.setOnClickListener(new Button.OnClickListener(){

        public void onClick(View v){
            text.setText("Disconnect");
                new Thread (new Runnable() {

                @Override
                public void run() {
                    // TODO Auto-generated method stub
```

액티비티가 실행될 때 호출되는 onCreate함수에서는 액티비티와 xml 레이아웃을 연결시키며 메모리를 할당해 객체를 화면에 나타냅니다. ImageView로 선언된 버튼들에는 클릭 리스너를 달았고 리스너 이 부분에서 http 통신이 쓰레드로 동작합니다.

```
private void gameStart () throws IOException{
    URL url = new URL("http://dkumse02.appspot.com/admin/start");
    HttpURLConnection httpUrl = (HttpURLConnection)url.openConnection();
    httpUrl.setDefaultUseCaches(false);
    httpUrl.setDoInput(true);
    httpUrl.setRequestMethod("GET");
    httpUrl.setRequestProperty("content-type", "application/x-www-form-urlencoded");

    BufferedReader bf = new BufferedReader(new InputStreamReader(httpUrl.getInputStream(), "UTF-8"));
    StringBuilder buff = new StringBuilder();
    String line;
    while((line = bf.readLine())!=null) {
        buff.append(line);
        if(line.equals("start")){
            runOnUiThread(new Runnable() {
              public void run() {
                  text.setText("Game Started!");
                  start.setImageResource(R.drawable.button_start_on);
                disconnect.setEnabled(true);
                disconnect.setFocusable(true);
                result.setEnabled(true);
                result.setFocusable(true);

              }
          });
        } else if (line.equals("fail")) {
            runOnUiThread(new Runnable() {

                @Override
                public void run() {
                      text.setText("Game cannot be started :(");
                }
            });
        }
    }
}
```

5. 스마트카 - 플레이어 시스템 구현

다음은 http 통신을 위해 사용되는 함수 중 하나입니다. 다른 함수들도 접속하는 URL만 다를 뿐 구조는 비슷합니다. 본 함수에서는 정해진 URL에 GET방식으로 접속하여 웹 페이지에 출력되는 문자열을 받아옵니다. 이후 불러온 문자열을 if-else 문에 따라서 UI를 어떻게 처리할 것인지가 결정됩니다. 오류를 방지하기 위해서 다음 단계에 꼭 필요한 버튼을 제외한 나머지는 비활성화 처리하였습니다.

```
case FORWARD:
    Motor_mode(0x09);
    setMaxSpeed(RX_buf[5]);
    break;
case LEFT:
    Motor_mode(0x0A);
    setMaxSpeed(RX_buf[5]);
    break;
case LIGHT:
    Motor_mode(0x05);
    setMaxSpeed(RX_buf[5]);
    break;
```

이 부분은 펌웨어에서 안드로이드 어플에서 속도와 관련된 패킷을 받아와서 switch()문 내에서 처리를 하는 부분입니다. case는 직진, 좌회전, 우회전으로 나뉘어 있는데 각도를 조금만 꺾는 회전은 Motor_mode는 직진이지만 양쪽 모터의 속도를 다르게 제어하여 회전하도록 구현했습니다. RX_buf[5]에는 속도를 들어있는데 위에서 설명한 바와 같이 왼쪽과 오른쪽 모터의 속도가 4bits씩 나누어 담겨있습니다. 그래서 setMaxSpeed()로 4bits씩 분할하여 왼쪽, 오른쪽 모터에 대한 최대속도를 셋팅하는 함수로 내부는 아래 사진과 같습니다.

```
void setMaxSpeed(char received) {
    right_max = received&15;
    left_max = received&240;
    left_max = left_max>>4;
}
```

8bits 중에 오른쪽의 4bit만 &연산으로 뽑아와서 right_max에 저장하고 왼쪽의 4bits를 뽑아와 오른쪽으로 4칸 쉬프트시켜서 left_max에 저장합니다.

이렇게 세팅된 왼쪽과 오른쪽의 최대 속도를 이용하여 loop()에서 계속 호출되는 Motor_Speed_Control()이라는 함수 내에서 왼쪽, 오른쪽 모터를 제어할 속도로 변환을 합니다. 함수 내부는 아래 사진과 같습니다.

```
void Motor_Speed_Control()
{
    if(left_max == right_max)
    {
        Motor_Speed_Left(left_max);
        Motor_Speed_Right(right_max);
    }
    else
    {
        left_speed = left_max*15;
        right_speed = right_max*15;
    }
}
```

left_max와 right_max가 같을 경우엔 Motor_Speed_Left(), Motor_Speed_Right() 함수를 호출하여 left_speed와 right_speed를 점점 증가시키고 left_max와 right_max가 다를 경우엔 바로 셋팅해줍니다. 두 함수의 내부는 아래와 같습니다.

```
void Motor_Speed_Left(char maxSpeed) {
    if (left_speed == maxSpeed * 15) {
        return;
    } else if (left_speed < maxSpeed * 15) {
        left_speed+=5;
    } else if (left_speed > maxSpeed * 15) {
        left_speed-=5;
    }
}

void Motor_Speed_Right(char maxSpeed) {
    if (right_speed == maxSpeed * 15) {
        return;
    } else if (right_speed < maxSpeed * 15) {
        right_speed+=5;
    } else if (right_speed > maxSpeed * 15) {
        right_speed-=5;
    }
}
```

loop()에서 아래사진과 같이 실행되므로 출발하거나 브레이크를 실행할 경우엔 속도가 가속되는 것처럼 보이는 효과를 낼 수 있었습니다.

```
void loop() {
    Motor_Speed_Control();
    Motor_Control('L', left_speed);
    Motor_Control('R', right_speed);
```

최종적으로 속도 컨트롤에 대해 간략하게 설명을 드리면 블루투스 통신으로 받아온 속도를 shift와 &연산을 이용하여 4bits로 나누어 왼쪽과 오른쪽의 최대속도를 셋팅하고 그 최대속도에 맞춰서 출발이나 브레이크 사용시엔 속도를 점점 증가하거나 감소하도록 하여 가속하는 효과를 보여주었고 회전시엔 좌,우 속도가 다른 값이 들어오므로 이 경우엔 바로 셋팅을 하여 반응속도를 최대로 올려주어 스마트카 조종에 현실감을 극대화 시켰습니다.

적외선 센서로 바닥을 인식하여 안드로이드 어플로 전송하는 부분에 대해서는 센서값을 읽을 때는 아두이노 자체함수인 analogRead를 사용합니다.

```
for(z=0;z<8;z++)
{
    Buff_A[z] = analogRead(SensorA[z]);
}
for(z=4;z<12;z++)
{
    if(Buff_A[z-4]>400)
    {
        TX_buf_infrared[z] = 1;
    }
    else
    {
        TX_buf_infrared[z] = 0;
    }
}
```

for 문을 8번 실행하도록 하여 8개의 센서에 대해서 analogRead() 함수를 이용하여 센서값을 읽어들인 후 자체 내에서 센서값이 400보다 클 경우 1을 아닐 경우엔 0을 배열에 저장하여 어플로 전송하도록 구현하였습니다. 400이라는 값은 센서값을 계속 출력해보는 많은 횟수로 반복해서 실험해본 결과 검은색을 인식하는 경우엔 300이하의 값이 출력되고 흰색을 인식하는 경우엔 600이상의 값이 출력됨을 확인하여 400이라는 저희만의 threshold 값을 지정해주었습니다. 어플에서는 전송받은 배열의 값이 모두 0일 경우 finish로 인식하고 모터의 속도를 0으로 제어하고 더 이상 패킷을 보내지 않도록 구현하였습니다.

먼저 시스템 구조에 대해서 전반적인 설명을 하자면 스마트카와 블루투스 통신을 이용하여 연결하고 안드로이드 장비(갤럭시 노트 10.1)에 탑재되어 있는 가속도 센서를 이용하여 장비를 기울이는 정도에 따라서 스마트카의 방향전환을 제어하도록 구현하였으며 레이싱 게임에 등장하는 게이지를 이용한 부스터, 아이템, 브레이크의 기능을 구현하였습니다. UI는 Cocos2dx 라는 엔진을 이용하여 NDK로 개발해야 하는 점을 고려하여 Java로 구현된 시스템 구조와 JNI를 이용하여 연결하였습니다.

블루투스 구현에 대해서 먼저 설명을 드리면 블루투스는 한백전자의 스마트카와 연결하기 위해 한백전자에서 jar파일로 배포한 라이브러리를 사용하려고 실험을 해보았는데 계속 안되어 jar파일을 코드로 뽑아내어 class로 만들어 log를 출력해보면서 라이브러리 내에서 액티비티의 뷰를 조정하는 부분을 모두 지우고 구현하였습니다. 블루투스를 연결할 때 디바이스 목록을 띄우고 선택을 하는 방법이 주로 사용되지만 저희는 고정된 스마트카 장비를 사용할 예정이라 주소를 강제세팅해두고 연결을 시도하면 자동으로 저희 팀이 사용하는 스마트카와 연결을 시도하도록 구현하였습니다.

이제부터는 소스를 보여드리며 설명드리겠습니다. 먼저 메인에는 아래 소스와 같은 함수들이 정의되어있습니다.

```
private native void UIReceive(int speed, int boost);
private native void BotConnect();
private native void Item(int itemId, int leftRight);
private native void Arrive(int time);
```

이 4개의 함수들은 JNI에서 시스템에서 UI로 Data를 넘겨줄 때 쓰이는 함수입니다.

다음 아래 함수들은 UI에서 Data로 이벤트가 일어났을 때 호출되는 함수들입니다.

```
public static void LeftItem()
{
    SystemData sData = SystemData.getIns();
    sData.leftItem = true;
}

public static void RightItem()
{
    SystemData sData = SystemData.getIns();
    sData.rightItem = true;
}

public static void Boost()
{
    SystemData sData = SystemData.getIns();
    sData.booster = true;
}

public static void Break()
{
    SystemData sData = SystemData.getIns();
    sData.brake = true;
}

public static void ConnectBot()
{
    SystemData sData = SystemData.getIns();
    sData.bluetooth = true;
}

public static void Start()
{
    SystemData sData = SystemData.getIns();
    sData.sendFlag = true;
}
```

이 함수들이 호출이 될 때는 싱글톤으로 구현되어있는 data를 관리하는 SystemData라는 클래스의 객체를 호출하여 각 상황의 flag를 true로 변환시켜줍니다.

이렇게 바꿔준 flag는 0.1초마다 호출되는 타이머에서 flag가 바뀐 것이 확인되면 해당하는 메소드를 호출하여 동작하고 flag는 다시 false로 초기화해주는 구조입니다.

타이머의 내부를 살펴보도록 하면 제일 먼저 아래의 루틴이 호출됩니다.

```
sData.x =acl.returnValue();
sData.gage = speed_cont.speedControl((int)sData.x);
storeMaxSpeed();
```

acl은 제가 구현한 클래스로 가속도센서에 관한 처리를 하는 클래스이며 returnValue() 함수는 안드로이드 장비의 가속도센서로 x축으로 기울어진 정도를 읽어들인 값을 반환하는 함수로 Data를 관리하는 싱글톤 객체에 저장합니다. 역시 제가 구현한 speed_cont 클래스 내에 speedControl 이란 함수는 회전을 위해 기울기 값을 이용하여 왼쪽과 오른쪽의 모터 속도를 다르게 설정하고 부스터 게이지값을 return 해주는 함수입니다. 그다음으로 호출되는 storeMaxSpeed() 함수의 내부는 아래 사진과 같습니다.

```
public void storeMaxSpeed()
{
        left = speed_cont.getLeftSpeed();
        right = speed_cont.getRightSpeed();
        if(left>right)
        {
                sData.maxSpeed = left;
        }
        else
        {
                sData.maxSpeed = right;
        }
}
```

왼쪽과 오른쪽 모터 속도를 비교하여 더 큰 값을 maxSpeed에 저장하는데 이유는 UI에 보내주어 속도를 표시해주기 위해서입니다.

속도는 0~15까지 16단계로 나뉘어져있는데 보통상태일때는 11이 최대속도이고 스피드 아이템을 사용할 경우 2가 올라가고 부스터를 사용할 경우 2가 올라가서 중복으로 사용할 경우 최대 15까지 올라갑니다. speedControl() 함수는 내부에서 부스터와 스피드 아이템 사용여부까지 고려하여 속도를 세팅해줍니다.

```
if(sData.bluetooth)
{
        mBluetooth.conntect();
        sData.bluetooth = false;
}
```

타이머 내에 있는 이 구문은 UI에서 블루투스를 연결하는 버튼을 눌렀을 때 블루투스를 연결하는 구문입니다.

```
if(sData.sendFlag)
{
        send();
        sData.time++;
        sData.item_count++;
        UIRecive(sData.maxSpeed, sData.gage);
}
```

이 구문은 sendFlag가 true일 때 실행되는데 sendFlag는 초기설정은 false이고 출발 신호가 들어오면 그때 sendFlag를 true로 세팅해줍니다.

send()함수는 패킷을 전송합니다. send() 함수에서는 sendCarPacket() 함수를 호출하여 속도와 모터 방향에 대한 패킷을 전송합니다.

처음에는 speed_control() 함수에서 셋팅한 속도를 그대로 보내주도록 구현했었는데, 회전반경이 너무 커서 send()함수에서 속도를 다시 셋팅하여 보내주도록 개발을 하였습니다.

또 이 구문에서는 랩타임을 지정해줄 time변수를 증가시키고 지정된 시간마다 아이템을 획득하기 위한 시간을 재어 줄 item_count 변수를 증가시키며 UI로 스피드와 게이지 값을 보내주어 실시간으로 UI에서 스피드값과 게이지값을 출력해줍니다.

```
if(infrared== 8)
{
        sData.sendFlag = false;
        byte cmd[];
        cmd = SmartMessage.CMD_FORWARD.clone();
        cmd[5] = (byte)0;
        cmd[6] = setCheckSum(cmd);
        mBluetooth.sendData(cmd);
        sData.init();
        Arrive(sData.time);
        Log.e("TEST", "arrive system");
}
infrared= 0;
```

이 구문에서 infrared 변수는 적외선 센서에서 0이 들어온 개수로 0이 8개가 들어올 경우 실행이 됩니다.

이 구문이 실행되면 sendFlag를 false로 세팅하여 더 이상 스마트카에 패킷을 보내지 않습니다.

```
if(sData.item_count == 30)
{
        sData.item_count = 0;
        sData.item = item_cont.obtainItem();
        Item(sData.item[1], sData.item[2]);
}
```

이 구문은 위에서 증가시킨 item_count값이 30이 되면 0으로 초기화시켜주고 아이템을 주는 구문입니다.