기말과제 스켈레톤 코드 및 전체적인 알고리즘

```
// 센서 - 아마 순서 바뀌어야 함
#pragma config(Sensor, S1, cs_left, sensorEV3_Color, modeEV3Color_Color)
#pragma config(Sensor, S2, cs_middle, sensorEV3_Color, modeEV3Color_Color)
#pragma config(Sensor, S3, cs_right, sensorEV3_Color, modeEV3Color_Color)
#pragma config(Motor, motorB, lm, tmotorEV3_Large, PIDControl, encoder)
#pragma config(Motor, motorC, rm, tmotorEV3_Large, PIDControl, encoder)

#define TICKRATE 20             //매 연산이 실행되는 틱레이트 - 단위는 ms
#define SPEED_MAX 8             //로봇의 최대 이동 속도

//컬러 인덱스
#define BLACK 1
#define BLUE 2
#define GREEN 3
#define YELLOW 4
#define RED 5
#define WHITE 6
#define BROWN 7


//과제에 따라 변경해야 할 변수들
#define TASK 1                  // 수행 과제 - 1: 1번 과제 | 2: 2번 과제
#define MAP_SIZE_COL 4          // 맵 격자 가로 크기
#define MAP_SIZE_ROW 4          // 맵 격자 세로 크기
#define LOC_START 0             // 로봇의 시작 위치 정의 - (row,column)좌표값을 MAP_SIZE_ROW * row  + column 로 나타낸다
#define DIR_START 2             // 로봇의 시작 방향 정의 - 1 : 위쪽 | 2: 오른쪽 | 3 : 아래쪽 | 4 : 왼쪽
#define LOC_DEST 15             // 도착점 좌표
#define LOC_MOV 15              // 이동점 좌표 - 도착점과 이동점 좌표가 같을 수도 있다
#define GRID_TICK 4             // 가운데 컬러 센서가 좌우 컬러센서보다 뒤에 있을 경우 갱신을 위해 확인하는 틱

//상태 변수 관련 정보 - 상태 변수는 stat 배열에 저장되며 아래는 각 스텟의 인덱스이다 | get_stat(상태변수명)으로 접근 가능 | set_stat(상태변수명, 값)으로 셋 가능
#define STATUS_SIZE 4           // 상태 변수 배열의 크기
#define DETECT 0                // 현재 디텍팅 상태가 저장된 인덱스 - stat[DETECT] = 0 : 시작 전 | stat[DETECT] = 1 : 격자를 순서대로 탐색 중 | stat[DETECT] = 2 : 탐색 완료
#define MOVE 1                  // 현재 로봇의 이동 상태가 저장된 인덱스 stat[MOVE] = 0 : 현재 로봇은 정지 상태  | stat[MOVE] = 1 : 현재 로봇은 직진 상태  | stat[MOVE] = 2 :
#define ONGRID 2                // 현재 로봇이 격자를 발견하였는가 stat[ONGRID] = 0 : 로봇은 격자좌표 위에 있지 않음 | stat[ONGRID] = 1 : 왼쪽 혹은 오른쪽 컬러센서에 노란색이
#define OFFROAD 3               // 현재 로봇이 길을 벗어났는가 (가운데 컬러 센서 값이 흰색인가) - stat[OFFROAD] = 0 : 로봇은 길을 잘 따라가고 있음(가운데 컬러센서가 흰색이 아님

#define DEST_QUEUE_SIZE 20      // destination queue 의 크기를 정의 -  destination queue는 원형 큐로 dq_idx 변수가 현재 가리키고 있는 destination queue 의 인덱스이다 | 자세한 설명은

int stat[STATUS_SIZE];
int dq[DEST_QUEUE_SIZE];        // destination queue를 정의한다 -원형 큐이다 | 자세한 설명은 문서에
int map[MAP_SIZE_ROW][MAP_SIZE_COL]; // 0 : 빈 공간 | 1: 빨간 색 | 2 : 파란 색
int patch = 0;                  //패치를 저장하는 변수 - 가장 최근에 가운데 컬러 센서에 인식된 값이 [0 : 다른 색 | 1: 빨간 색 | 2 : 파란 색]의 형태로 저장되고, 빨간 색 혹은 파란
int grid_tick = GRID_TICK;

int score_q[DEST_QUEUE_SIZE];   // 점수 변화량을 저장하는 배열 -> 자세한 설명은 문서에

int dq_idx = 0;

int dir_cur = DIR_START;                    // 로봇의 현재 방향을 나타내는 변수  - 1 : 위쪽 | 2: 오른쪽 | 3 : 아래쪽 | 4 : 왼쪽
int dir_dest = DIR_START;                   // 로봇의 목표 방향을 나타내는 변수 - 회전 시 사용한다

int loc_cur = LOC_START;        // 로봇의 현재 위치를 나타내는 변수 - (row,column)좌표값을 MAP_SIZE_COL * row  + column 로 나타낸다  | ex - 5 x 4 맵에서 로봇이 1행 2열에 있다면 loc_

int score = 0;                  // 현재 점수
```

변경 가능

기본 스탯

```
33  //상태 변수 관련 정보 - 상태 변수는 stat 배열에 저장되며 아래는 각 스텟의 인덱스이다 | get_stat(상태변수
34  #define STATUS_SIZE 4           // 상태 변수 배열의 크기
35  #define DETECT 0                    // 현재 디텍팅 상태가 저장된 인덱스 - stat[DETECT] = 0 : 시작
36  #define MOVE 1                      // 현재 로봇의 이동 상태가 저장된 인덱스 stat[MOVE] = 0 : 현지
37  #define ONGRID 2                    // 현재 로봇이 격자를 발견하였는가 stat[ONGRID] = 0 : 로봇은
38  #define OFFROAD 3                   // 현재 로봇이 길을 벗어났는가 (가운데 컬러 센서 값이 흰색인가)
39
40  #define DEST_QUEUE_SIZE 20      // destination queue 의 크기를 정의 -  destination queue는 원형 큐로
41
42  int stat[STATUS_SIZE];
```

Stat[0] 에 저장되는 값의 의미 - 현재 디텍팅 상태

stat[DETECT] = 0 : 시작 전

stat[DETECT] = 1 : 격자를 순서대로 탐색 중

stat[DETECT] = 2 : 탐색 완료 후 도착점에서 이동점으로 이동 중

stat[DETECT] = 3 : 패치 경로를 탐색 중

stat[DETECT] = 4 : 패치 경로에 따라 되돌아오기 진행 중

stat[1] 에 저장되는 값의 의미 - 현재 로봇의 이동 상태

stat[MOVE] = 0 : 현재 로봇은 정지 상태

stat[MOVE] = 1 : 현재 로봇은 직진 상태

stat[MOVE] = 2 : 현재 로봇은 dir_dest을 향해 회전 중 상태

stat[2] 에 저장되는 값의 의미 - 현재 로봇이 격자를 발견하였는가

stat[ONGRID] = 0 : 로봇은 격자좌표 위에 있지 않음

stat[ONGRID] = 1 : 왼쪽 혹은 오른쪽 컬러센서에 노란색이 인식됨

stat[ONGRID] = 2 : 왼쪽 혹은 오른쪽 컬러센서가 막 노란색에서 흰색으로 변함 -> 격자 위에 진입함

stat[3] 에 저장되는 값의 의미 - 현재 로봇이 길을 벗어났는가 (가운데 컬러 센서 값이 흰색인가)

stat[OFFROAD] = 0 : 로봇은 길을 잘 따라가고 있음(가운데 컬러센서가 흰색이 아님)

stat[OFFROAD] = 1 : 로봇은 길을 벗어남(가운데 컬러센서가 흰색임)

stat 변수 접근 및 변경(사용)법

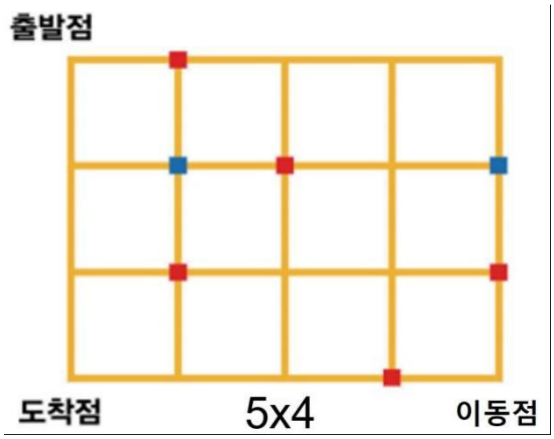
```
116
117  // input 인덱스에 해당하는 스텟 값을 가져오는 함수 - 사용 예시  : if(get_stat(DETECT) == 2) do something
118  int get_stat(int index){
119          return stat[index];
120  }
121
122  // index에 해당하는 스텟 값을 value로 수정하는 함수 - 사용 예시 : set_stat(DETECT, 2)
123  void set_stat(int index, int value){
124          if(index < STATUS_SIZE)
125                  stat[index] = value;
126          return;
127  }
128
```

해당 두 함수를 이용해 접근한다.

기본 변수들

map

```
44  int map[MAP_SIZE_ROW][MAP_SIZE_COL]; // 0 : 빈 공간 | 1: 빨간 색 | 2 : 파란 색
```



예를 들어 맵이 다음과 같다면 디텍팅이 끝난 뒤 map 배열에 저장되는 값은 다음과 같다

{

{0, 1, 0, 0, 0},

{0, 2, 1, 0, 2},

{0, 1, 0, 0, 1},

{0, 0, 0, 1, 0},

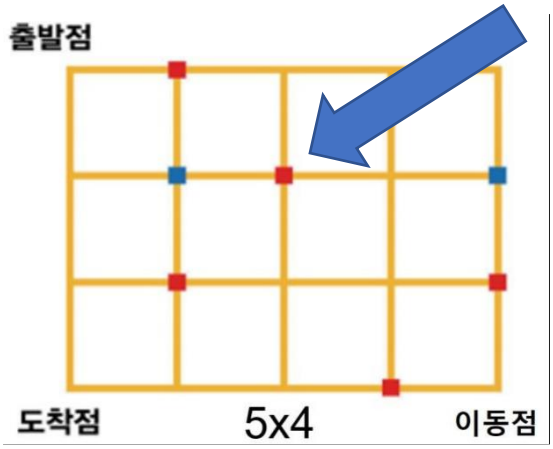
}

Location

```
55    int loc_cur = LOC_START;
```

```
// 로봇의 현재 위치를 나타내는 변수 - (row,column)좌표값을 MAP_SIZE_COL * row  + column 로 나타낸다
```

모든 위치 좌표를 x, y 로 연산하면 다중 배열을 너무 많이 사용하게 되므로 모든 저장되는 위치 좌표는 인덱스 포인터 방식으로 변환하여 저장한다.



ex - 5 x 4 맵에서 로봇이 1행 2열에 있다면 loc_cur = 7

5 x 4 맵에서 로봇이 0행 2열에 있다면 loc_cur = 2

5 x 4 맵에서 로봇이 3행 0열에 있다면 loc_cur = 10

Location 변환 법

```
183
184   // loc의 row좌표를 얻어온다
185   int get_loc_row(int loc){
186           return loc / MAP_SIZE_COL;
187   }
188   // loc의 column좌표를 얻어온다
189   int get_loc_col(int loc){
190           return loc % MAP_SIZE_COL;
191   }
192
193   // (row, column) 좌표를 loc 값으로 반환한다
194   int get_loc(int row, int column){
195           return MAP_SIZE_COL * row + column;
196   }
197
```

다음 함수들을 사용한다

Direction

```
51
52   int dir_cur = DIR_START;
53   int dir_dest = DIR_START;
54
```

```
// 로봇의 현재 방향을 나타내는 변수  - 1 : 위쪽 | 2: 오른쪽 | 3 : 아래쪽 | 4 : 왼쪽
// 로봇의 목표 방향을 나타내는 변수 - 회전 시 사용한다

1 : 위쪽 | 2: 오른쪽 | 3 : 아래쪽 | 4 : 왼쪽 을 나타낸다


Ex -  만약 로봇이 현재 오른쪽을 향하고 있고 아래쪽을 향하도록 회전해야 한다
```

- Dir_dest 를 3으로 변경하고 set_stat(MOVE,2)를 통해 현재 상태를 회전 중으로 바꾼다

전체적인 로봇의 이동

```
42    int stat[STATUS_SIZE];
43    int dq[DEST_QUEUE_SIZE];        // destination queue를 정의한다 -
```

로봇은 기본적으로 dq에 저장된 다음 목표치로 이동한다.

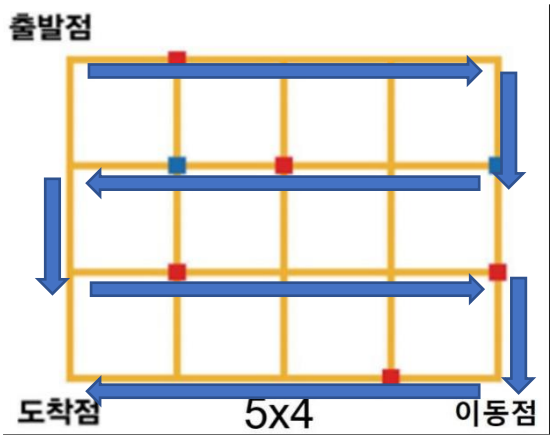
각 목표치는 변환된 위치 좌표 값이다.

각 인접한 목표치는 직진이어야 한다.

빈 자리는 -1이 저장된다. – 마지막 목표치 다음 값은 무조건 -1이어야 한다

```
49
50    int dq_idx = 0;
51
```

- 다음 목표치의 index값이다



예를 들어 해당 맵에서 로봇이 위과 같은 경로로 이동해야 한다면

Dq[]에는 다음과 같은 수가 저장된다.

Loc_cur = 0

Dq = {4, 9, 5, 10, 14, 19, 15, -1,-1,-1,.....}

로봇은 다음 목표치까지 각 그리드 좌표를 인식하며 직진하고,

목표치에 도달 시 dq_idx를 증가시키며 다음 목표치를 업데이트한다.

돌아올 때

처음에 패치들을 디텍팅할 때는 위의 예시처럼 이동한다.

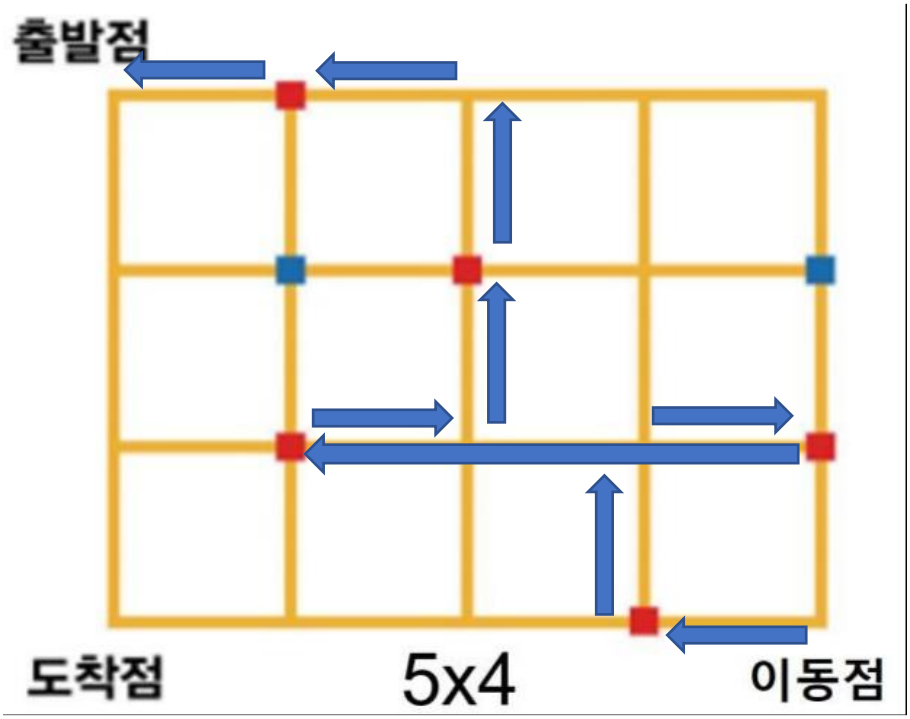
돌아올 때도 연산을 하여 계산된 경로를 dq에 업데이트하면 되지만

Score의 즉각적인 증가를 위해 약간 다른 점이 있다.

```
47
48   int score_q[DEST_QUEUE_SIZE];
49
```

첫째로 직진 도중이더라도 스코어가 변화되는 곳은 dq에 추가되어야 한다.

둘째로 dq와 같은 인덱스의 score_q[]에 스코어 변화가 명시되어 있어야 한다.



예를 들어 과제 2에서 가장 많은 스코어를 위해서는 다음과 같이 이동해야 하는데,

Dq = {18, 13, 14, 11, 12, 2, 0} 다음과 같이 저장하지 않고

Loc_cur = 19

Dq = {18, 13, 14, 11, 12, 7, 2, 1, 0}

Score_q = {5, 0, 5, 5, 0, 5, 0, 5, 0}

위와 같이 저장한다.

이후 각 이동에 맞추어 현재 도착한 dq의 인덱스와 같은 인덱스의 score_q값을 더한다

(매 격자 이동시마다 1점 감점은 내부 다른 코드에서 처리한다.

```
87  task main()
88  {
89          init_stat();
90          init_map();
91          init_dq();
92          init_score_q();
93          sleep(5000);  // 5초동안 대기 후 출발
94          update_dq_detect(); //destination queue를 ㄹ 자 모양의 탐색으로 초기화
95          set_stat(MOVE,1);
96
97          while(1){
98                  update_stat_by_color();
99                  update_status();
100                 update_action();
101                 sleep(TICKRATE);
102         }
103 }
104
```

내부 main 함수 – 전체적인 FLOW

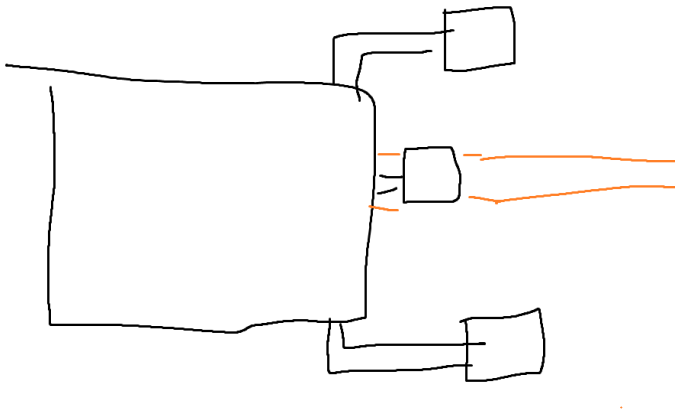
update_stat_by_color(); 현재 컬러에 따라 각 그리드 좌표 위를 지나는지 판단

update_status(); 컬러 센서로 받아온 정보에 따라서 부가적으로 stat을 업데이트

update_action(); 현재 스텟에 따라 다음 행동을 지정

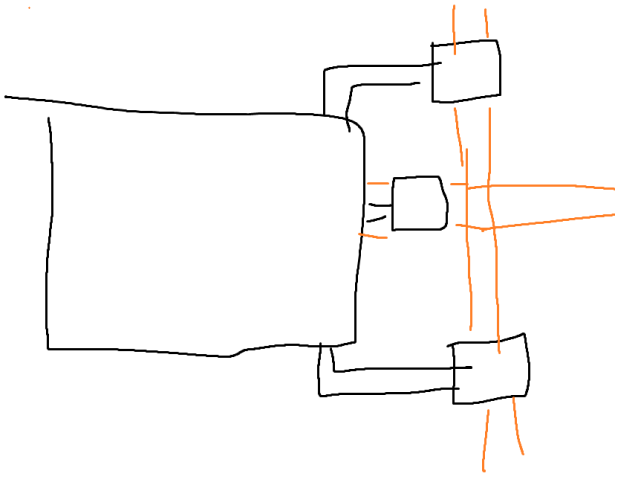
sleep(TICKRATE); TICKRATE만큼 SLEEP한다

각 격자 위에 올라감 판단



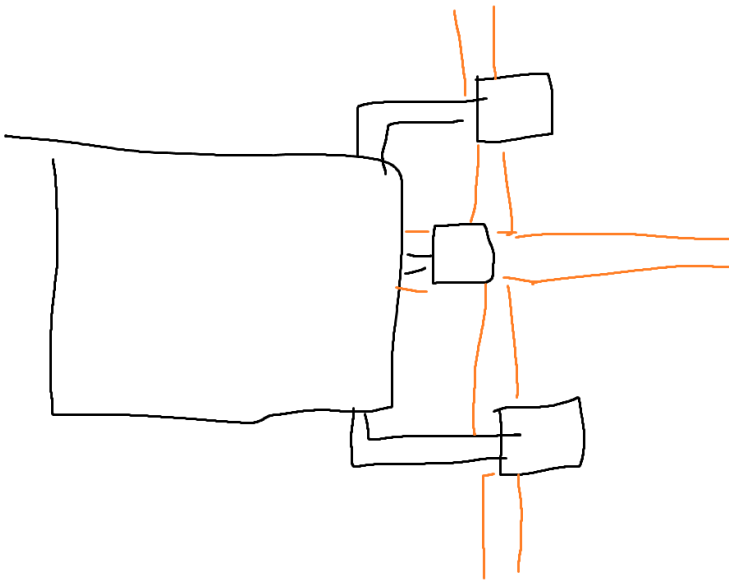
가운데 컬러센서는 색깔이 들어오고 이전에 좌우 컬러센서도 계속 흰색이었을 경우

`stat[ONGRID] = 0`



가운데 컬러센서 색깔이 들어오고 좌 혹은 우 컬러센서에 색깔이 들어왔을 경우

`stat[ONGRID] = 1 -> 좌우 컬러 센서가 격자를 지나고 있는 상태`

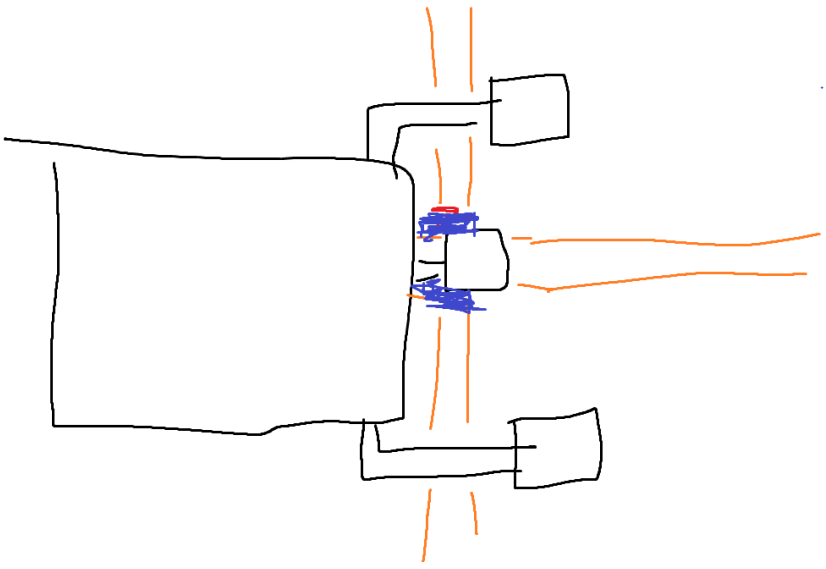


가운데 컬러센서 색깔이 들어오고 좌우 컬러센서 값이 모두 흰색인데

현재 stat[ONGRID] == 1일 경우 (바로 직전에) 좌 혹은 우에 색깔 있던 상태에서 방금 흰색으로 바뀌었을 때

➔ 격자 위에 올라온 것으로 판단

다만, 위 그림과 같이 가운데 컬러센서가 좌 우 컬러센서보다 뒤에 있어서 해당 순간에 격자의 색을 판단하지 못할 수도 있으므로



```
#define LOC_MOV 15
#define GRID_TICK 4

int grid_tick = GRID_TICK;
```

```
        if(col_middle == BLUE) patch = 2;
}
// 왼쪽 오른쪽 센서가 둘다 흰색일 경우
if(col_left == WHITE && col_right == WHITE){
        // 왼쪽 혹은 오른쪽 센서에 노란색 길이 인식되었다가 이제 막 흰색으로 변한 경우 -> 그리드 좌표 위에 올라옴 판단
        if(get_stat(ONGRID) == 1){
                // 가운데 컬러 센서가 좌우 컬러센서보다 뒤에 있을 경우를 대비해 지정된 틱 횟수만큼 추가 확인 후 ongrid 스탯을 2로 변경
                if(!grid_tick--){
                        set_stat(ONGRID,2);
                        grid_tick = GRID_TICK;
                        return;
                }

        }
}else{
```

다음과 같이 지정된 틱만큼 지난 후에 ONGRID를 2로 변경한다.

해당 틱횟수는 회전코드를 짜는 사람이 임의로 변경할 수 있다.

```
211
212  // 맵 정보를 출력하는 함수
213  void print_map(int startline){
214          for(int i = 0; i< MAP_SIZE_ROW ; i++){
215                  for(int j = 0; j< MAP_SIZE_COL ; j++){
216                          displayBigStringAt(startline+i, 2 * j, map[i][j] == 0 ? "+" :(map[i][j] == 1 ? "O" : "X"));
217                  }
218          }
219  }
220
221  // 현재 스텟을 출력하는 함수
222  void print_stat(){
223          displayBigTextLine(1, "det %d mv %d off %d",get_stat(DETECT), get_stat(MOVE),get_stat(OFFROAD));
224          displayBigTextLine(3, "cur [%d,%d] next [%d,%d]", get_loc_row(loc_cur), get_loc_col(loc_cur), get_loc_row(dq[dq_idx]), get_loc_col(dq[dq_idx]));
225          displayBigTextLine(5, "cur %s dest %s score %d",dir_to_text(dir_cur),dir_to_text(dir_dest),score);
226          print_map(7);
227  }
228
229
```

EV3의 스크린에는 언제나 다음과 같은 정보가 출력된다.

필요한 역할

1. 회전함수

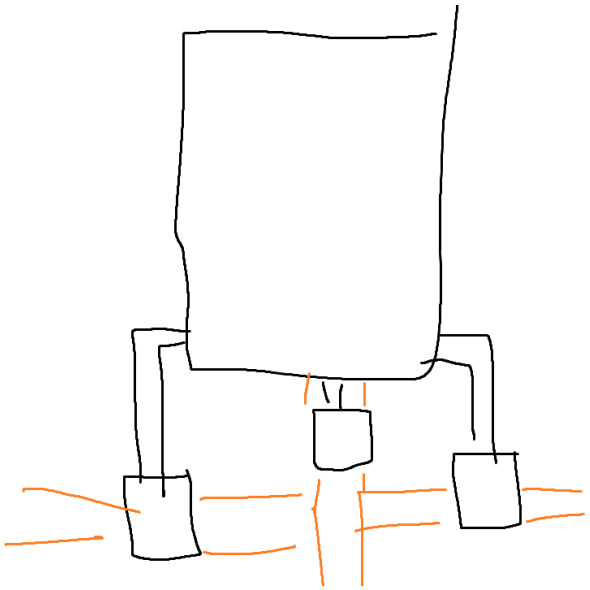
```
454
455         if(move == 2){
456                 // 현재 회전 상태일 경우 현재 방향인 dir_cur 가 dir_dest와 같아지도록 회전 - 단, 코드가 길어질 경우 함수로 따로 뺄 것
457                 // 단, 현재 방향과 목표 회전 방향이 같을 경우 아무것도 하지 않고 리턴
458                 /*
459                         code here!
460                 */
461                 set_stat(MOVE,1);               // 회전이 끝난 후 move를 직진 상태로 바꾸기
462                 move = get_stat(MOVE);
463         }
464
```

현재 MOVE stat이 2(회전 중)일 경우

로봇의 현재 direction인 dir_cur 가 dir_dest와 같아지도록 회전한다.

단, 회전각도는 90도가 아니라 180도나 0 (cur_dir가 처음부터 dir_dest와 같아서 회전하지 않음)

일 수 있고,



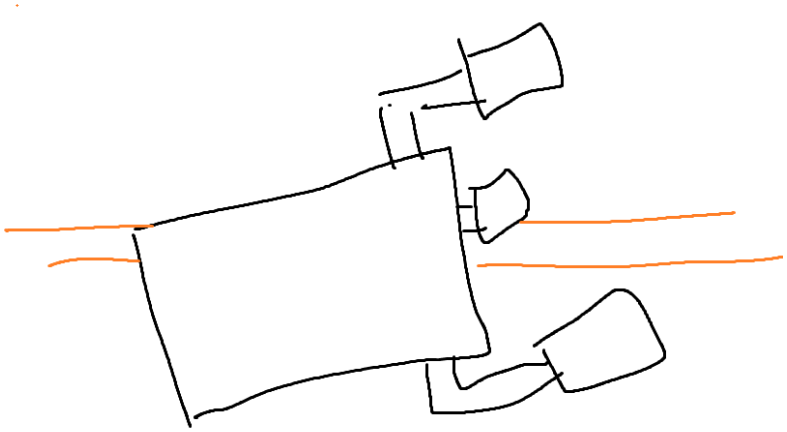
회전 이후는 calibration하는 사람을 위해 위와 같이 목표 방향과 수평이어야 한다(길을 어정쩡하게 벗어나는 상태여서는 안된다)

이 과정에서 로봇의 회전축이 정확이 GRID와 일치하도록 하는 GRID_TICK 수치를 찾아 조정하여도 되며, 원한다면 후진을 활용하여도 된다.

원한다면 해당 코드 내부에서 sleep(TICKRATE)를 사용하여도 된다.

2. 맵 벗어남 CALIBRATION

```
445         int move = get_stat(MOVE);
446
447         if(get_stat(OFFROAD)){
448                 // 길을 미세하게 벗어났을 경우 calibration 하는 코드 - 단, 코드가 길어질 경우 힐
449                 /*
450                         code here!
451                 */
452                 return;
453         }
454
```



위와 같이 로봇이 미묘하게 길을 벗어났을 때 현재 스텟들을 참고하여 조정하는 역할로, 원한다면 정지나 후진 후 수정하여도 된다. 다만 코드의 충돌과 인식오류를 방지하기 위해 앞쪽 그리드까지는 전진하지 않는 것을 추천한다.

코드가 충돌나지 않게 만들 자신이 있다면 앞으로 가면서 조정하는 것이 훨씬 더 속도면에서 좋으므로 그렇게 해도 좋다.

3. 1번과제 경로 알고리즘

```
384
385 // 1번 과제의 최다 획득 경로를 계산하고 해당 정보를 dq에 업데이트 하는 함수
386 void update_dq_1(void){
387         /*
388         code here!
389         */
390         /*
391         for ...
392                 dq[somthing] = something
393
394
395         dq[end+1] = -1;   //-> 마지막 경로 다음 값을 -1로 수정하는 것 잊지 말기
396         */
397 }
```

1번 과제의 최다 점수 획득 경로를 연산하고 해당 경로와 스코어 정보를 앞서 명시한 규칙대로 Dq[]와 score_q[]에 저장한다

4. 2번과제 경로 알고리즘

```
398
399  // 2번 과제의 최다 획득 경로를 계산하고 해당 정보를 dq에 업데이트 하는 함수
400  void update_dq_2(void){
401          /*
402          code here!
403          */
404          /*
405          for ...
406                  dq[somthing] = something
407
408
409          dq[end+1] = -1;   //-> 마지막 경로 다음 값을 -1로 수정하는 것 잊지 말기
410          */
411  }
412
```

2번 과제의 최다 점수 획득 경로를 연산하고 해당 경로와 스코어 정보를 앞서 명시한 규칙대로 Dq[]와 score_q[]에 저장한다