<규도기-Team>
<ADD-ON >
설계 산출물

문서버전	3.0
문서 ID	SE-2023-002
최종변경일	2023-12-03
문서상태	릴리즈

요약

SIM 및 AI 음성인식, ADD-ON 시스템의 설계 산출물을 기술.
서브 시스템의 구성과 각 서브 시스템의 구조를 기술.
주요 산출물
- ✓ 아키텍처도
- ✓ 클래스도
- ✓ 교류도
- ✓ 상태도

컴퓨터과학부 2019920006 김도현
컴퓨터과학부 2019920005 김기찬
컴퓨터과학부 2019920059 한규만

표 1 문서 변경 기록

문서이름	<SE-Team> <Mini-Shaker> 설계 산출물		
문서 ID	SE-2023-002		
버전		변경일	설명
1	1	2023-10-16	아키텍처도 작성했다.
	2	2023-10-18	클래스도, 교류도 초안 작성했다.
	3	2023-10-20	ADD-ON 서브시스템을 자세히 기술하고, 클래스도를 완성했다.
	4	2023-10-24	교류도, 상태도 초안 작성하고 클래스도에서 GUI를 분리했다.
	5	2023-10-27	교류도, 상태도를 완성했다.
2	0	2023-11-05	객체 상태도를 완성하였으며, 전체 클래스도의 흐름을 변경했다.
	1	2023-11-12	경로 탐색 과정에 오류가 있어 수정하였다.
	2	2023-11-19	로봇의 회전이 정상적으로 진행되지 않는 오류를 수정하였다.
	3	2023-11-25	spot의 입력 형식을 수정하였다.
3	0	2023-12-03	릴리즈

1. 개　요

1.1 목　적

본 문서는 “AI 기반 Mobile Robot Controller”의 분석 산출물을 기술한다.

- ✓ 전체 시스템이 어떤 서브 시스템으로 구성되어 있는지 "아키텍처도"를 통해 보여준다.
- ✓ 서브 시스템이 어떤 클래스로 구성되어 있고 클래스들간에 어떤 관계가 있는지를 "클래스도"를 통해서 보여준다.
- ✓ 객체간에 발생하는 동적인 행위를 "교류도"를 통해서 보여준다.
- ✓ 한 객체의 상태 변화를 "상태도"를 통해서 보여준다.

1.2 참고 문헌

없음.

2. 아키텍처도

RobotService 하위로 Controller, PathFinder, SIM, RobotController :

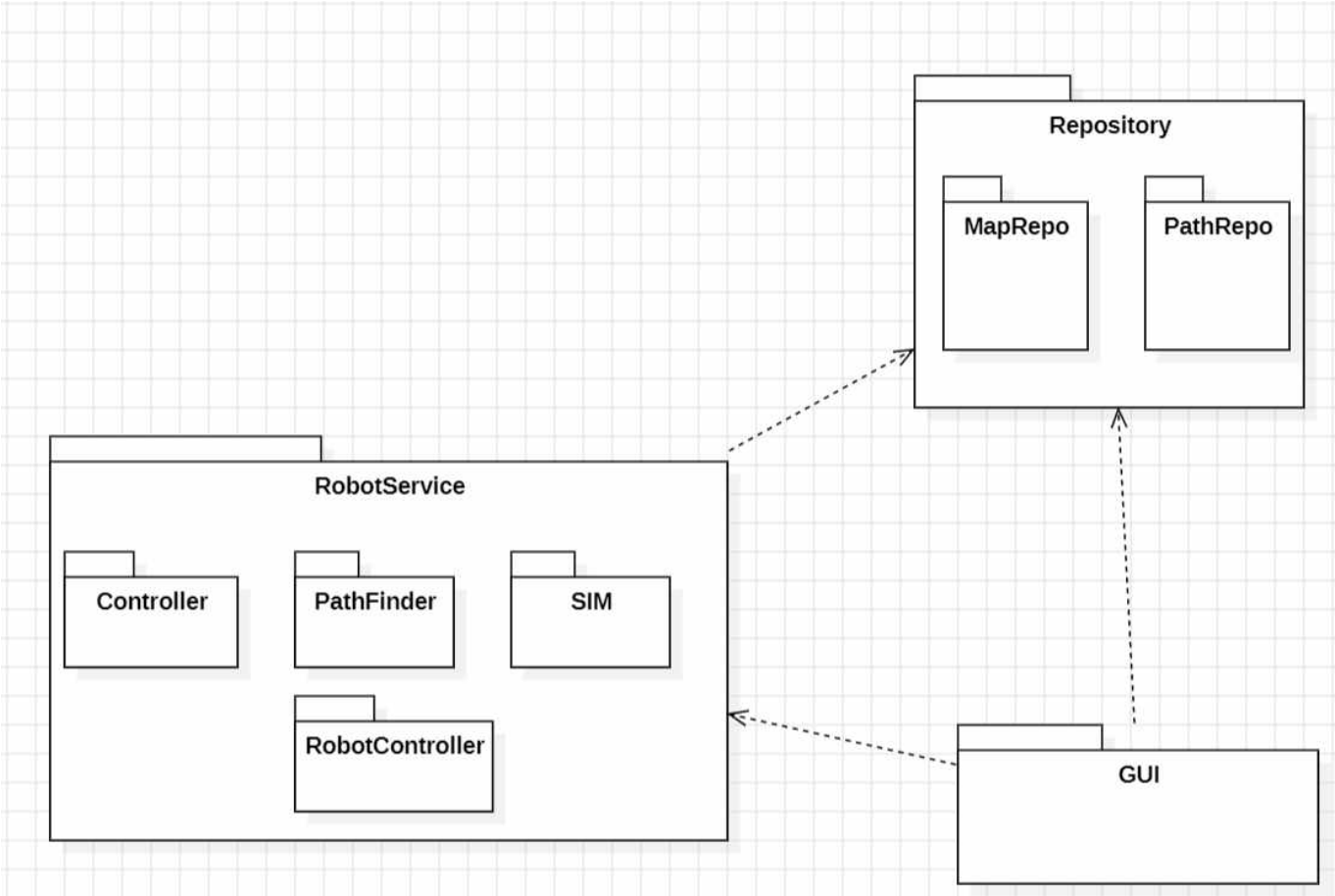
- Repo의 정보를 처리하는 역할을 한다.
- Java언어를 사용하여 작성된다.

Repository의 하위로 MapRepo, PathRepo:

- Map의 정보와 path의 정보를 저장하는 역할을 한다.
- Java를 사용하여 구현한다.

GUI:

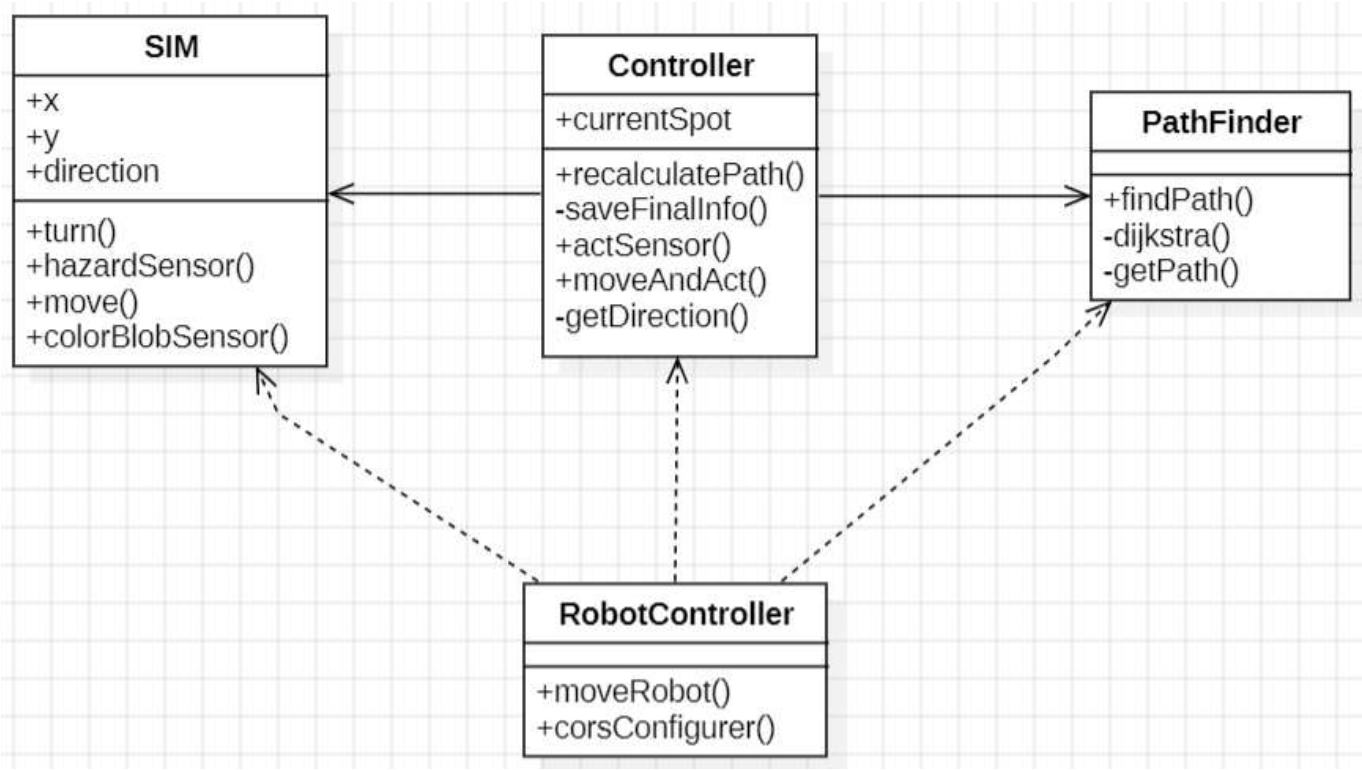
- 값을 입력받는 화면, 로봇의 이동 과정을 표시한다.



3. 서브시스템 세부 설계

- 전체 시스템은 아래의 서브시스템으로 구현되었다.

3.1 Robot Service System



SIM 클래스는 로봇의 기본 동작을 구현하는 클래스이다.
turn(): 로봇을 90도 시계방향으로 회전시키는 메서드이다. 현재 방향에 따라 다음 방향으로 변경한다.
colorBlobSensor(): 컬러 블롭을 탐색하는 센서 동작을 시뮬레이션하는 메서드이다.
현재 위치 주변의 컬러 블롭을 탐지하고 탐지된 블롭의 위치를 반환한다.
hazardSensor(): 해저드를 탐색하는 센서 동작을 시뮬레이션하는 메서드이다.
현재 방향으로 한 칸 전진하여 해저드를 탐지하고, 탐지된 위치를 반환한다.
move(): 로봇을 움직이는 메서드로, 0칸, 1칸, 또는 2칸 이동하는 시뮬레이션을 수행한다..
이동 결과에 따라 성공 여부와 이동한 칸 수를 반환한다.

Controller 클래스는 SIM의 기본 동작을 조합하여 로봇을 더 높은 수준에서 제어하는 클래스이다.
actSensor(): 센서를 가동하여 현재 로봇의 위치, 방향, 컬러 블롭, 해저드 정보를
문자열로 반환하는 메서드이다.
getDirection(int targetX, int targetY): 목적지로 향하는 방향을 설정하는 메서드로,
현재 위치와 목표 위치를 기반으로 시계 방향으로 90도 회전한 방향을 반환한다.
moveAndAct(): 로봇을 이동시키면서 센서 동작을 수행하는 메서드이다.
저장된 경로를 가져와 경로를 따라 이동하면서 센서 동작을 수행하고,
이동 중 오작동이 발생하거나 해저드를 탐지하면 경로를 재계산하여 이동을 제어한다.

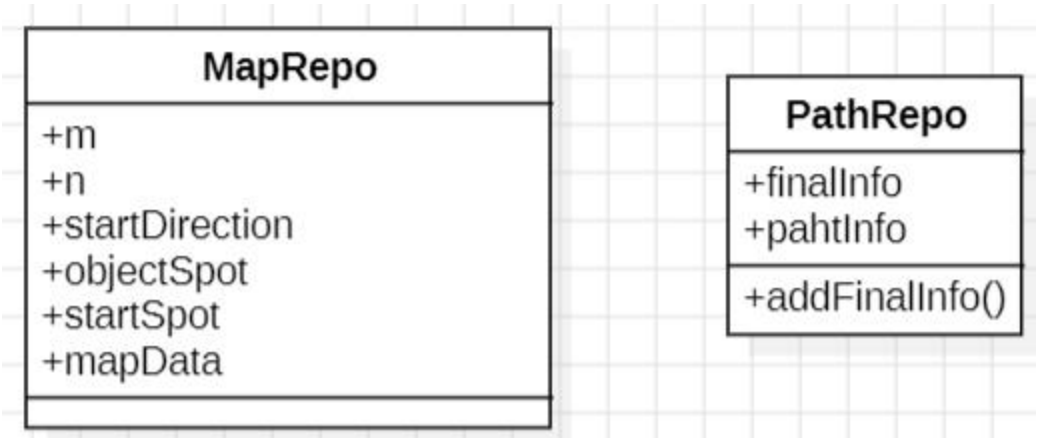
recalculatePath(): 경로를 재계산하는 메서드로, 현재 로봇의 위치와 목적지를 기반으로

새로운 경로를 계산하고 PathRepo에 저장한다.
saveFinalInfo(): 최종 경로 정보를 저장하는 메서드로, 로봇의 현재 위치, 방향, 컬러 블롭
및 해저드 정보를 저장하고, 목적지에 도착한 경우 목적지 도착 여부를 표시한다.

PathFinder 클래스는 로봇과 지도의 정보를 기반으로 로봇의 이동 경로를 찾는다.
findPath(): 출발지에서 여러 목적지 까지의 최단 경로를 찾아내고, 결과를 path Result에 추가한다.
dijkstra(): 다익스트라 알고리즘을 사용하여 출발지에서 각 지점까지의 최단 거리를 계산한다.
getpath(): 출발지와 목적지 사이의 최단 경로를 반환한다.

RobotController 클래스는 웹 애플리케이션에서 로봇 서비스와 프론트엔드간의
통신을 담당하는 클래스입니다.
moveRobot(): 클라이언트에서 전송된 요청을 받아와 처리한다.
요청 데이터에는 초기 상태, 장애물, 컬러 블롭등이 있다.
corsConfigurer(): API 보안 설정을 해제해준다.

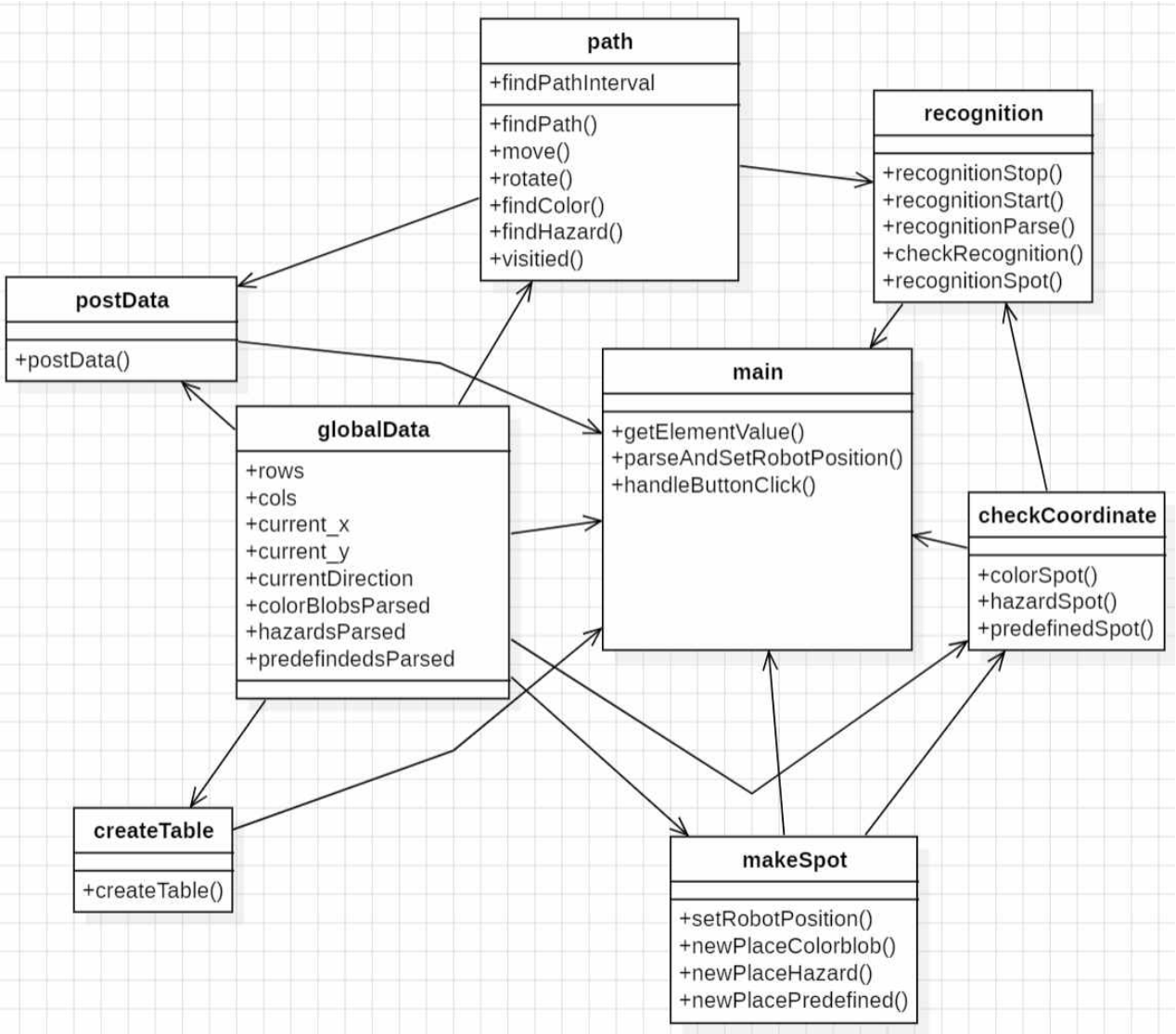
3.2 Repository System



MapRepo 클래스는 지도에 관한 정보를 저장하는 클래스이다.

PathRepo 클래스는 경로에 대한 정보를 저장하는 클래스이다.
addFinalInfo(): info리스트를 finalInfo 리스트에 추가하는 메서드

3.3 GUI System



main: 가장 최초의 화면에서 입력을 받고 동작을 시키는 역할을 한다.
getElmentValue(): HTML 요소의 ID를 인자로 받아 해당 요소의 값을 반환한다.
parseAndSetRobotPosition(): 로봇의 위치를 파싱, 저장하고 지도 상에 표시한다.
handleButtonClick(): 버튼 클릭 이벤트를 처리한다.

globalData: 데이터를 관리하기 위한 데이터 저장소 역할을 한다.

postData(): 클라이언트에서 서버로 맵, 로봇 위치 및 주변 환경 데이터를 전송하고
서버에서 반환된 경로 정보를 처리하여 로봇의 경로 탐색을 시작하는 메서드

path: 로봇의 제어를 수행하는 역할을 한다.
findPath(): 로봇의 이동 및 상황 정보를 바탕으로 로봇을 이동시키고 환경을 업데이트한다.
move(): 화면상 로봇의 위치를 변경하고 이동시킨다.
rotate(): 로봇의 방향을 변경하여 회전을 시뮬레이션한다.
findColor(): 컬러 블롭 센서의 정보를 받아와 화면에 표시한다.
findHazard(): 위험 지점 센서의 정보를 받아와 화면에 표시한다.
visited(): 목적지를 방문했을 때의 동작을 처리한다.

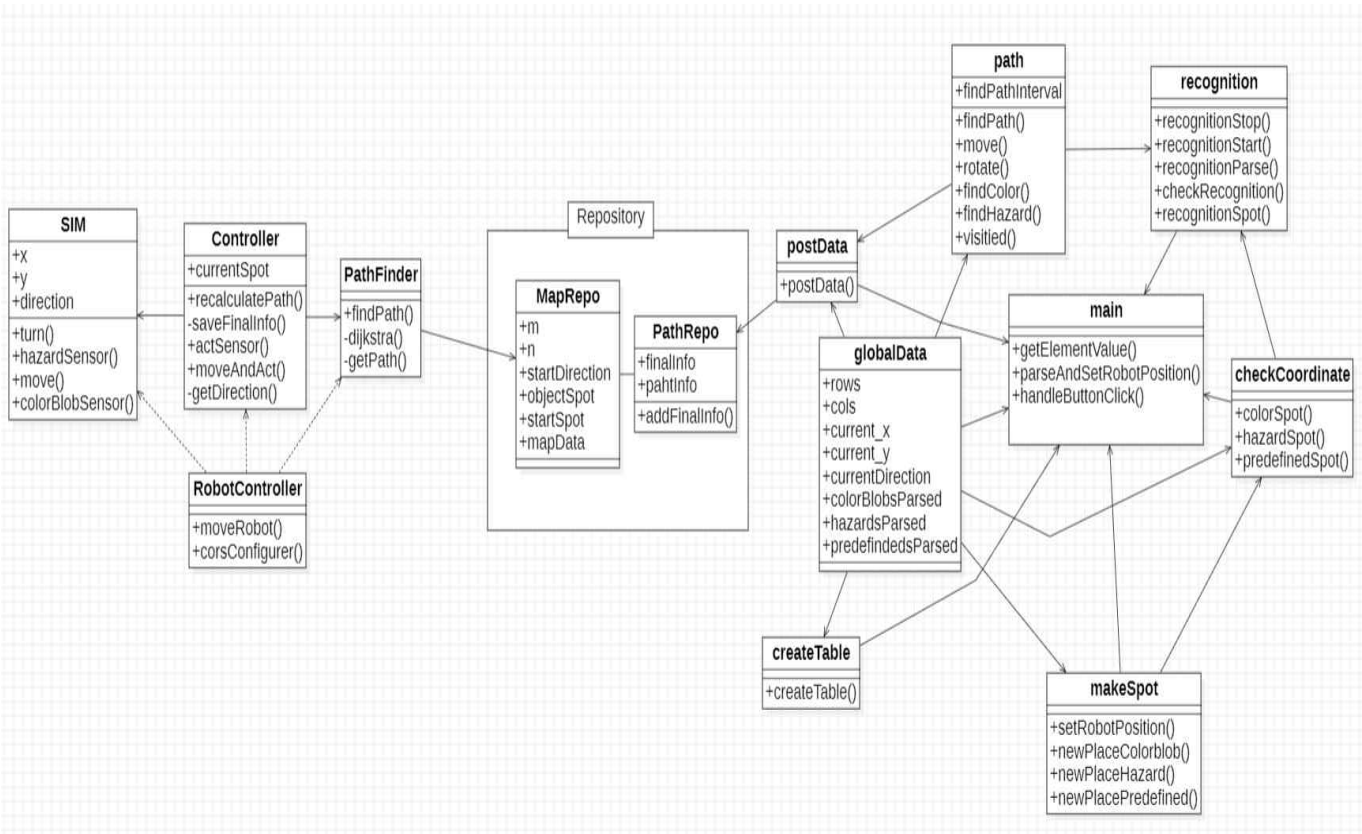
recognition: 음성 인식을 통해 사용자의 명령을 받아 처리하는 역할을 한다.
recognitionStop(): 음성 인식을 중지시킨다.
recognitionStart(): 음성 인식을 시작하고 사용자의 음성을 입력받아 처리하기 위한 설정을 수행한다.
recognitionParse(): 음성 인식 결과 문자열을 파싱하여 필요한 정보를 추출하고, 잘못된 입력을 필터링.
checkRecognition(): 파싱된 음성 인식 결과를 검사하여 유효한 명령인지 확인하고, 잘못된 입력 처리.
recognitionSpot(): 유효한 음성 인식 결과에 따라 해당 위치에 컬러 블롭 또는 위험 지점을 표시.

checkCoordinate: 주변 지점에 컬럽 블롭, 위험 지점, 목적지를 표시하는 역할을 한다.
colorSpot(): 주어진 위치에 컬러 블롭을 표시하고 해당 위치 정보를 데이터에 추가한다.
hazardSpot(): 주어진 위치에 위험 지점을 표시하고 해당 위치 정보를 데이터에 추가한다.
predefinedSpot(): 주어진 위치에 미리 정의된 목적지를 표시하고 해당 위치 정보를 데이터에 추가한다.

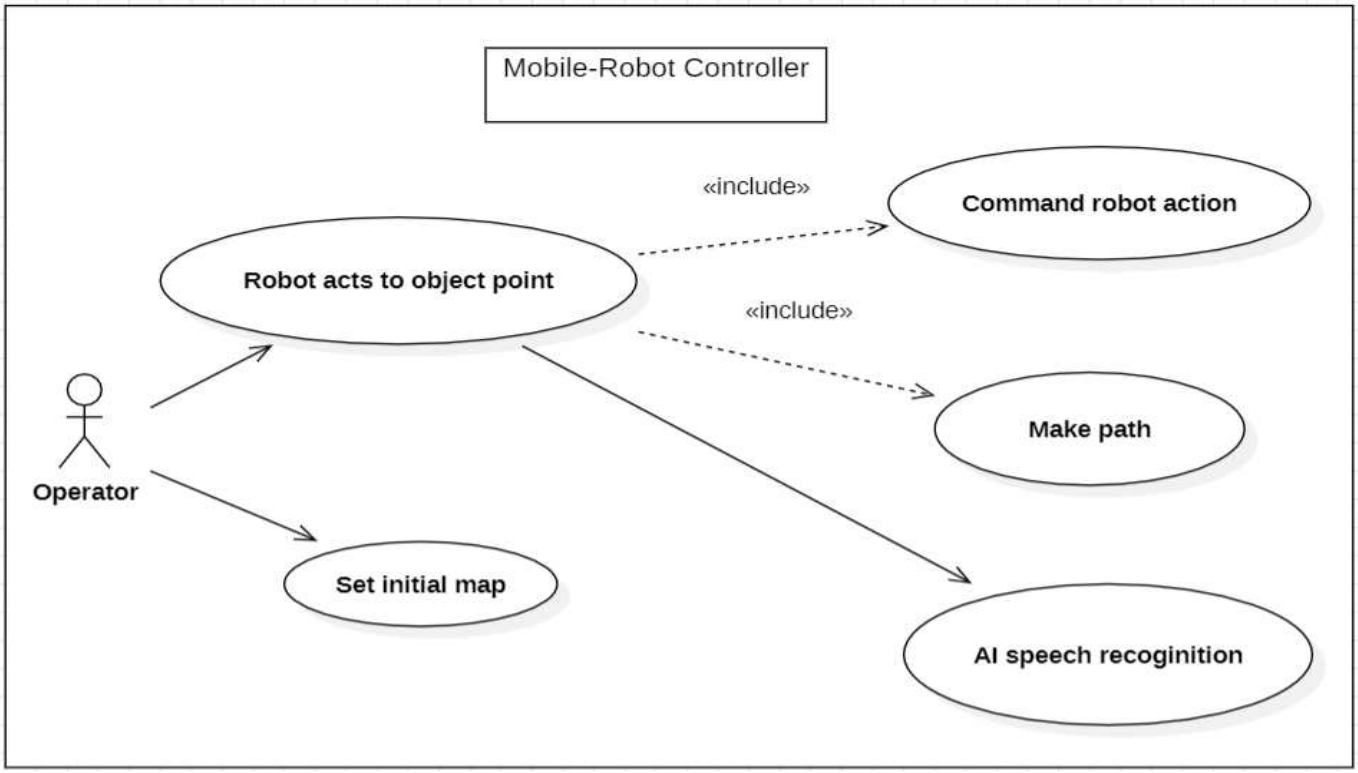
makeSpot: 로봇의 위치 및 환경을 시각적으로 표시하는 역할을 한다.
setRobotPosition(): 로봇의 현재 위치를 주어진 좌표로 설정하고 시각적으로 표시한다.
newPlaceColorBlob(): 주어진 위치에 회색 컬러 블롭을 표시한다.
newPlaceHazard(): 주어진 위치에 회색 위험 지점 이미지를 표시한다.
newPlacePredefined(): 주어진 위치에 회색 목적 지점 이미지를 표시한다.

createTable(): 사용자가 입력한 행과 열 수에 기반하여 HTML 테이블을 생성하고, 이를 화면에 표시한다.

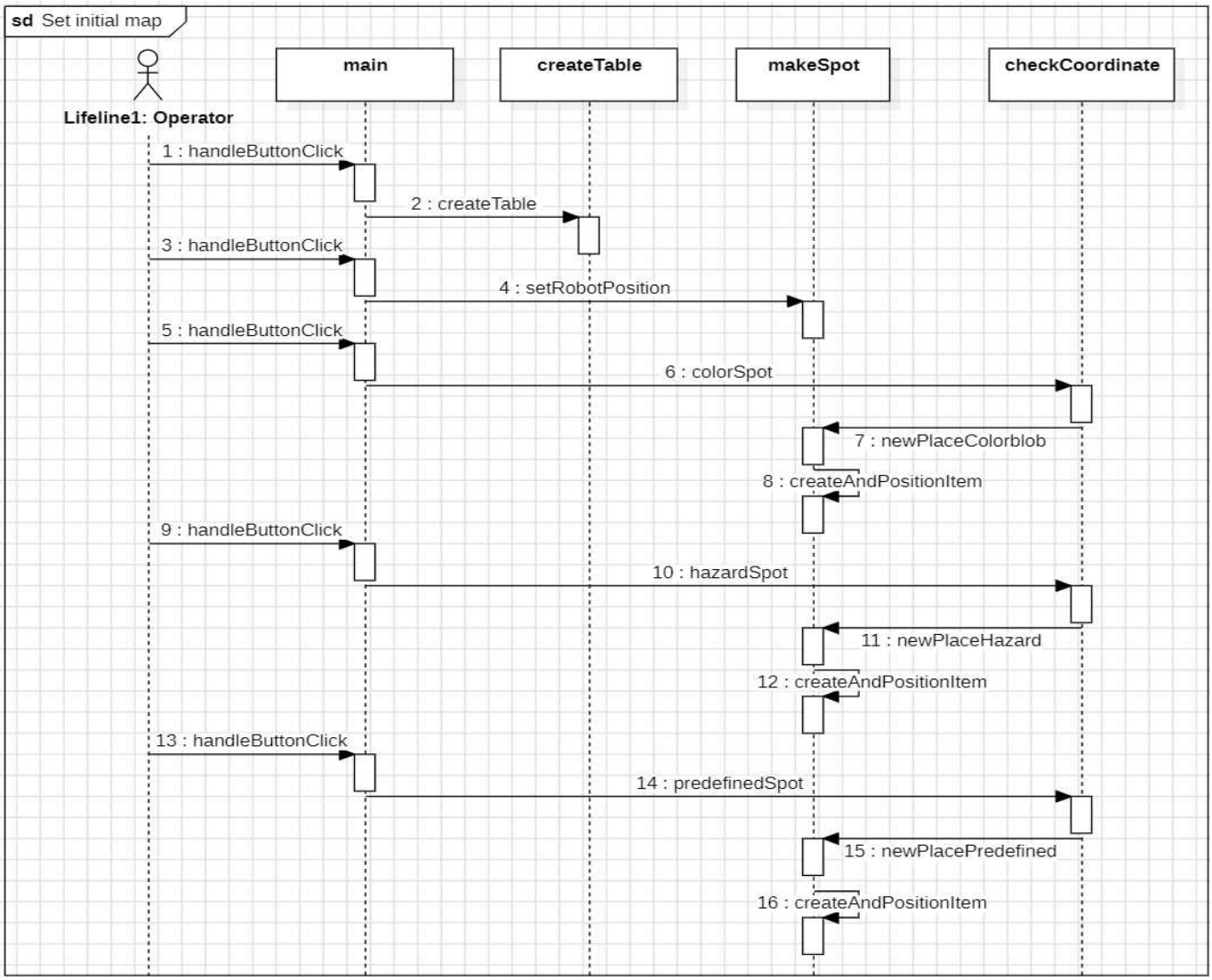
3.4 시스템 구조



4. 교류도

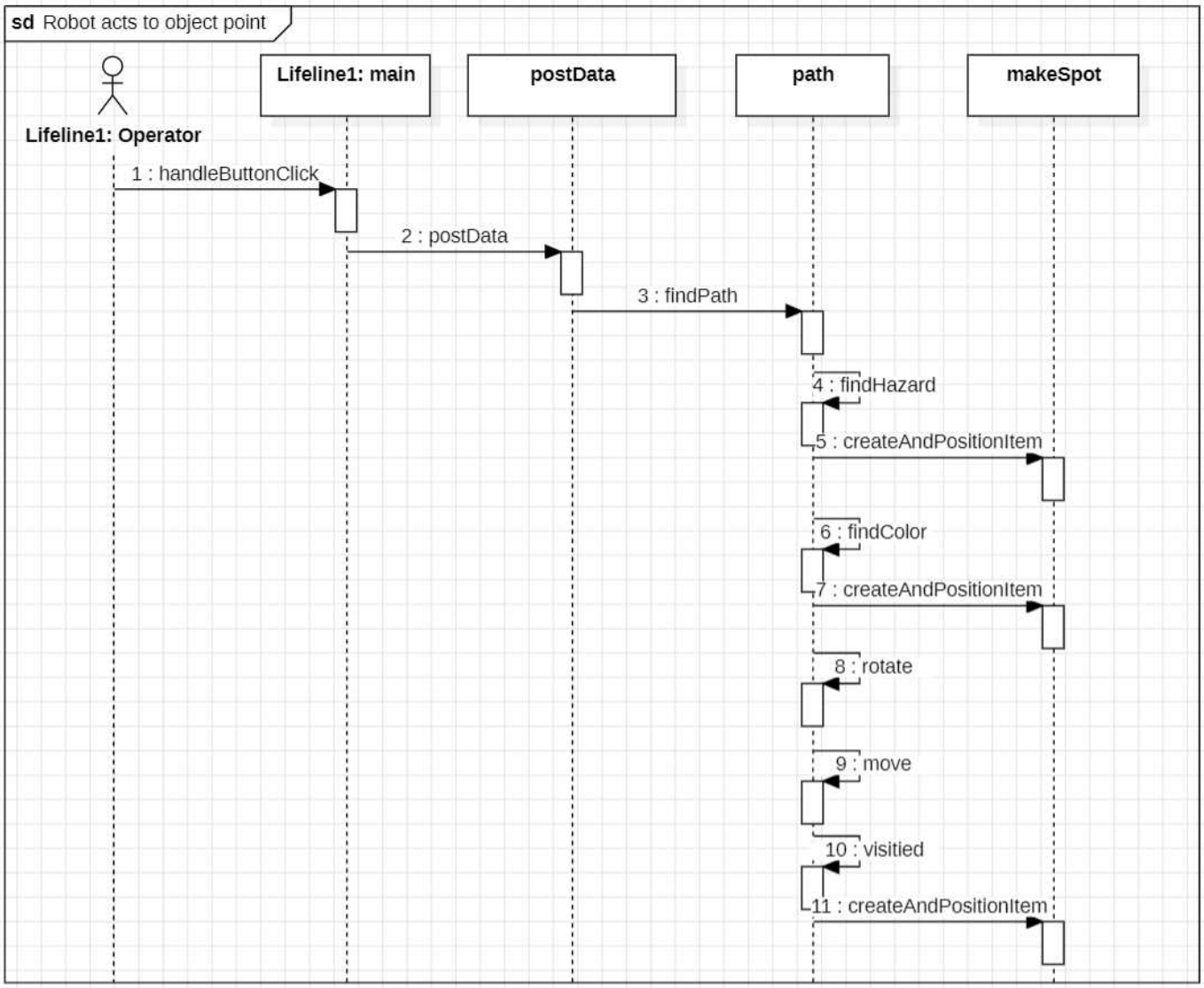


4.1 초기 맵 정보 입력



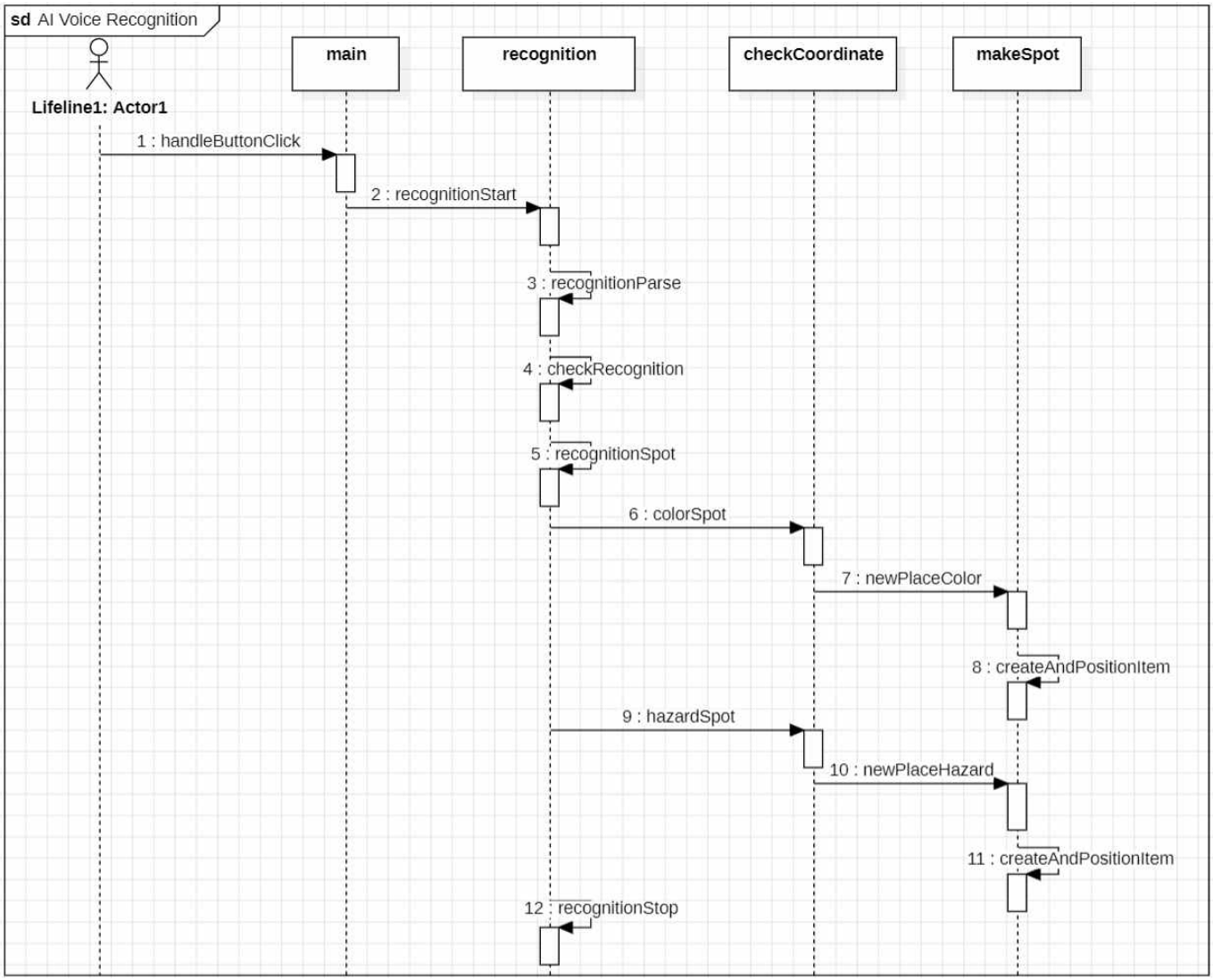
1. handleButtonClick 메소드를 통해 버튼을 관리한다.
2. 지도 크기 입력 칸에 (x y) 형식으로 입력하면 지도가 그려진다.
3. 로봇의 시작 위치를 (x y) 형식으로 입력하면 로봇의 초기 위치가 설정된다.
4. color blob의 위치를 ((x1 y1)(x2 y2)) 형식으로 입력하면 color blob이 설정된다.
5. hazard spot의 위치를 ((x1 y1)(x2 y2)) 형식으로 입력하면 hazard spot이 설정된다.
6. predefined spot의 위치를 ((x1 y1)(x2 y2)) 형식으로 입력하면 predefined가 설정된다.

4.2 로봇의 길 탐색 과정 표시(Floating Navigator)



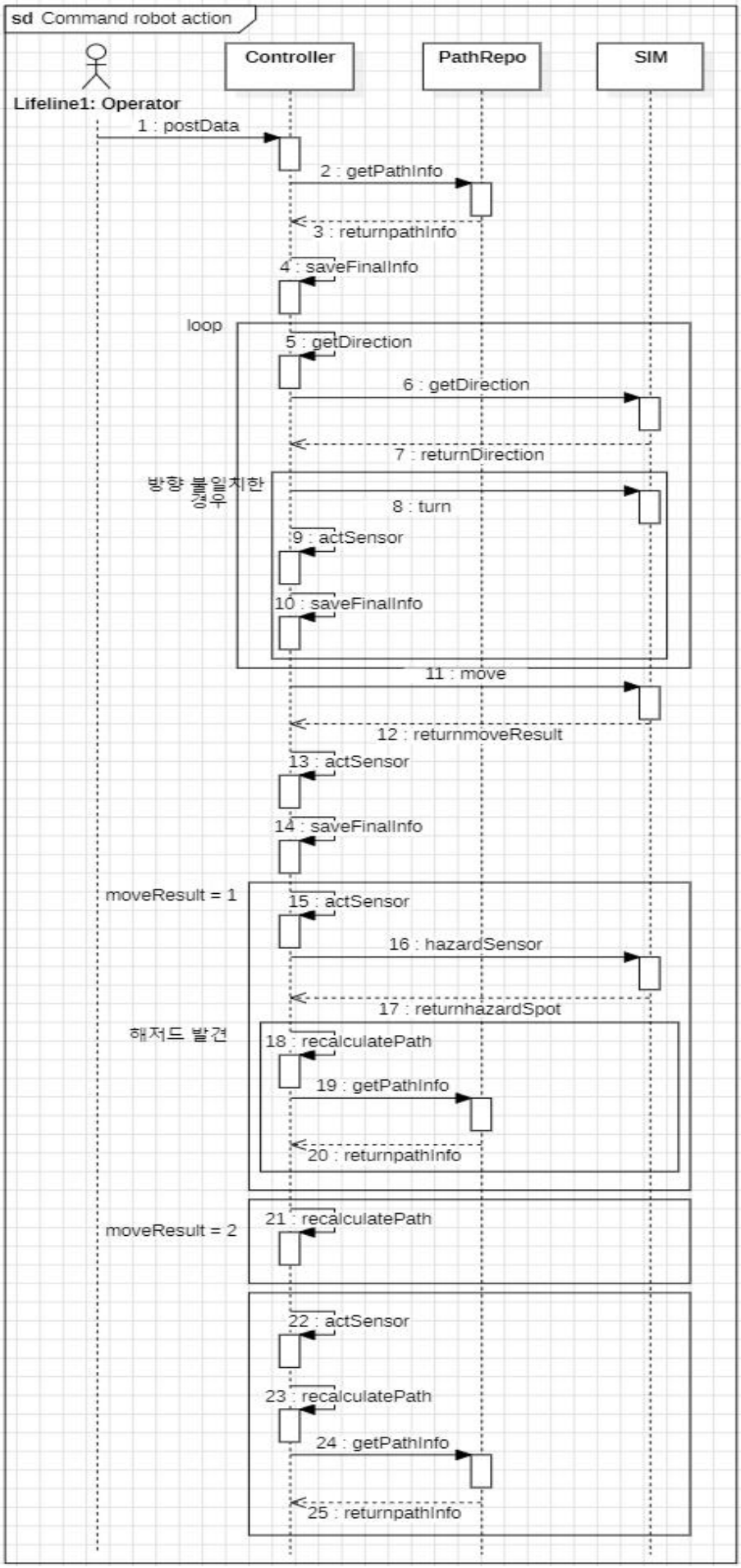
1. 로봇 이동 버튼을 누른 후 경로를 탐색한다.
2. postData()를 요청하여 경로 정보를 불러온다.
3. 요청한 정보를 바탕으로 findPath()가 실행되어 경로를 바탕으로 로봇이 움직인다.
4. findHazard(), findColor()를 통해 hazard, color blob 센서가 가동되어 탐색 됐다면 신호를 보낸다.
5. 센서 가동 후 rotate()를 통해 회전, move()를 통해 움직인다.
6. visitied()가 실행되어 현재 위치가 predefined spot인지 체크하고, 맞다면 신호를 보낸다.

4.3 AI 음성 인식



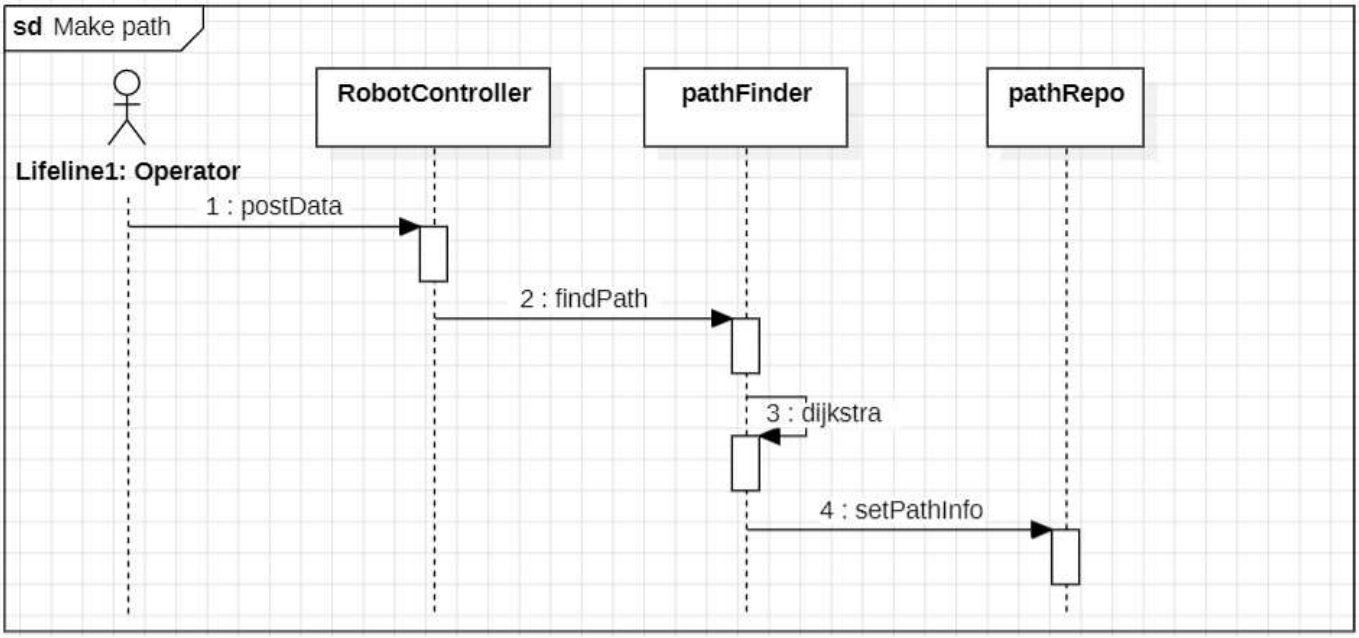
1. 음성 인식 시작 버튼 클릭을 통해 recognitionStart()가 실행되며 음성 인식이 시작된다.
2. 입력 받은 결과값을 recognitionParse(), checkRecognition()을 통해 올바른 결괏값인지 검사하고 변환한다.
3. recognitionSpot()을 통해 color blob인지 hazard spot인지 판별 후 해당 지점을 추가한다.
4. color blob이라면 newPlaceColor()메소드를, hazard spot이라면 newPlaceHazard()를 실행시킨다.
4. 음석 인식 정지 버튼 클릭을 통해 recognitionStop()이 실행 되며 음성 인식이 정지된다.

4.4 로봇 동작 지시


sd Command robot action
Controller
PathRepo
SIM
Lifeline1: Operator
1 : postData
2 : getPathInfo
3 : returnpathInfo
4 : saveFinalInfo
loop
5 : getDirection
6 : getDirection
7 : returnDirection
방향 불일치한 경우
8 : turn
9 : actSensor
10 : saveFinalInfo
11 : move
12 : returnmoveResult
13 : actSensor
14 : saveFinalInfo
moveResult = 1
15 : actSensor
16 : hazardSensor
17 : returnhazardSpot
해저드 발견
18 : recalculatePath
19 : getPathInfo
20 : returnpathInfo
moveResult = 2
21 : recalculatePath
22 : actSensor
23 : recalculatePath
24 : getPathInfo
25 : returnpathInfo

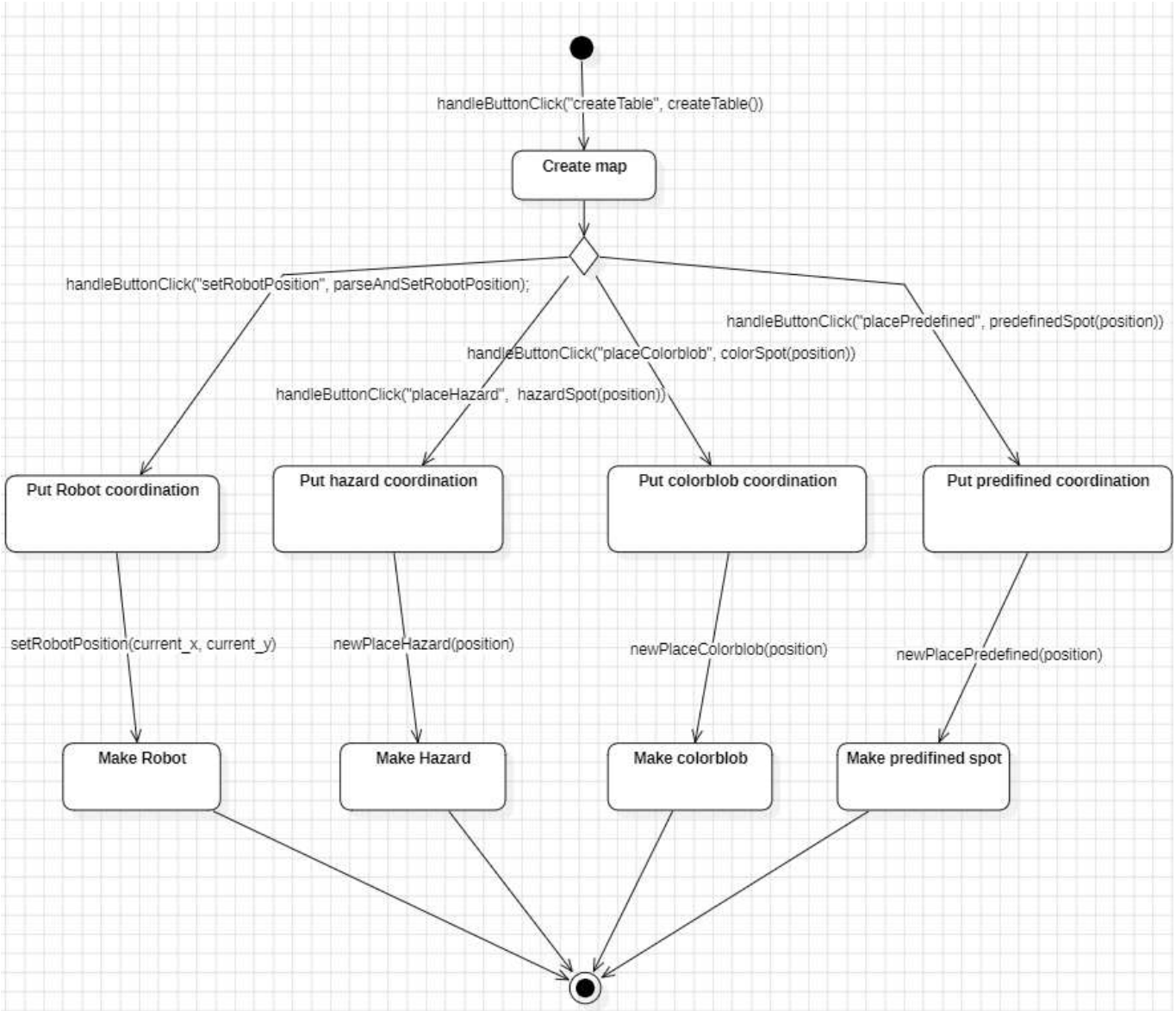
1. pathRepo.getPathInfo()를 호출하여 경로 정보를 얻습니다.
2. saveFinalInfo()를 호출하여 최종 정보를 저장합니다.
3. 경로 정보를 기반으로 반복문을 시작합니다. pathInfo 리스트의 각 항목을 순회합니다.
4. 현재 위치가 목표 위치와 다를 경우, 다음 동작을 수행합니다.
5. 목표 위치를 문자열로부터 파싱하여 targetX와 targetY를 얻습니다.
6. 현재 방향을 목표 방향과 일치할 때까지 루프를 실행합니다.
- getDirection(targetX, targetY)를 호출하여 현재 위치에서 목표 위치로 가기 위한 방향을 얻습니다.
- sim.getDirection()를 호출하여 현재 시뮬레이터의 방향을 가져옵니다.
- 현재 방향과 목표 방향이 일치하면 루프를 종료합니다.
- 그렇지 않으면, sim.turn()을 호출하여 방향을 변경하고, actSensor()와 saveFinalInfo()를 호출하여 센서 동작을 수행하고 정보를 저장합니다.
7. 목표 위치까지 이동합니다.
- sim.move()를 호출하여 이동을 시도하고 이동 결과를 moveResult에 저장합니다.
- 이동 결과에 따라 다음 동작을 수행합니다.
    - moveResult가 1인 경우, 해저드가 감지되었을 때:
        - actSensor()를 호출하여 센서 동작을 수행합니다.
        - sim.hazardSensor()를 호출하여 해저드 위치를 확인합니다.
        - 해저드가 발견되면, recalculatePath()를 호출하여 경로를 재계산하고, 경로 정보를 다시 얻어옵니다.
        - i를 -1로 설정하여 경로 정보를 처음부터 다시 순회합니다.
    - moveResult가 2인 경우, 이동에 성공했지만 다른 동작은 필요하지 않을 때:
        - recalculatePath()를 호출하여 경로를 재계산합니다.
    - 그 외의 경우:
        - 이동에 실패했거나 다른 상황이 발생한 경우:
            - actSensor()를 호출하여 센서 동작을 수행합니다.
            - recalculatePath()를 호출하여 경로를 재계산하고, 경로 정보를 다시 얻어옵니다.
            - i를 -1로 설정하여 경로 정보를 처음부터 다시 순회합니다.
8. 경로의 모든 항목을 순회하고 종료됩니다.

4.5 경로 생성



1. 사용자가 엔드포인트를 호출하여 로봇 이동을 요청한다.
2. 지도를 초기화하고 경로 계산을 준비한 이후 findPath 메서드를 호출한다.
3. dijkstra 메서드를 호출하여 최단 경로를 계산한다.
4. 계산 결과는 pathRepo에 저장된다.

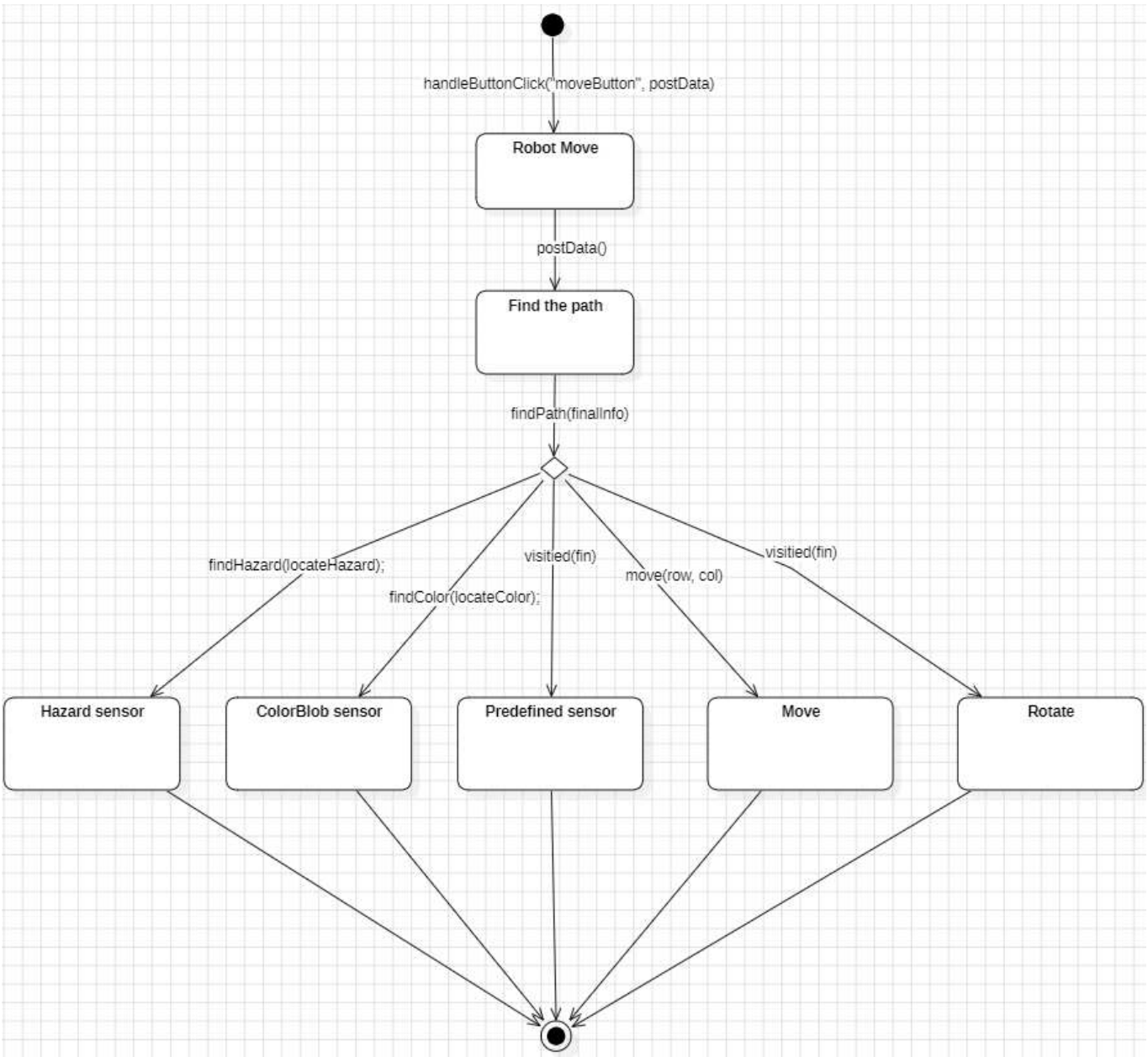
5.1 Set initial map 객체 상태도



메인 화면에서의 상태를 보여준다.

1) Create map: 지도 크기 버튼이 클릭되어 지도 정보 설정 후, 지도가 화면에 표시된다.
2) Put robot coordination: 로봇 위치 설정 버튼이 클릭되어 로봇 위치 설정을 한다.
3) Put hazard coordination: hazard 추가 버튼이 클릭되어 hazard spot 위치 설정을 한다.
4) Put colorblob coordination: colorblob 추가 버튼이 클릭되어 color blob 위치 설정을 한다.
5) Put predefined coordination: predefined 추가 버튼이 클릭되어 predefined 위치 설정을 한다.
6) Make Robot: 설정된 로봇 위치를 바탕으로 로봇이 화면에 표시된다.
7) Make Hazard: 설정된 hazard spot의 위치를 바탕으로 hazard spot이 화면에 표시된다.
8) Make colorblob: 설정된 color blob의 위치를 바탕으로 color blob이 화면에 표시된다.
9) Make predefined spot: 설정된 predefined spot의 위치를 바탕으로 predefined spot이 화면에 표시된다.

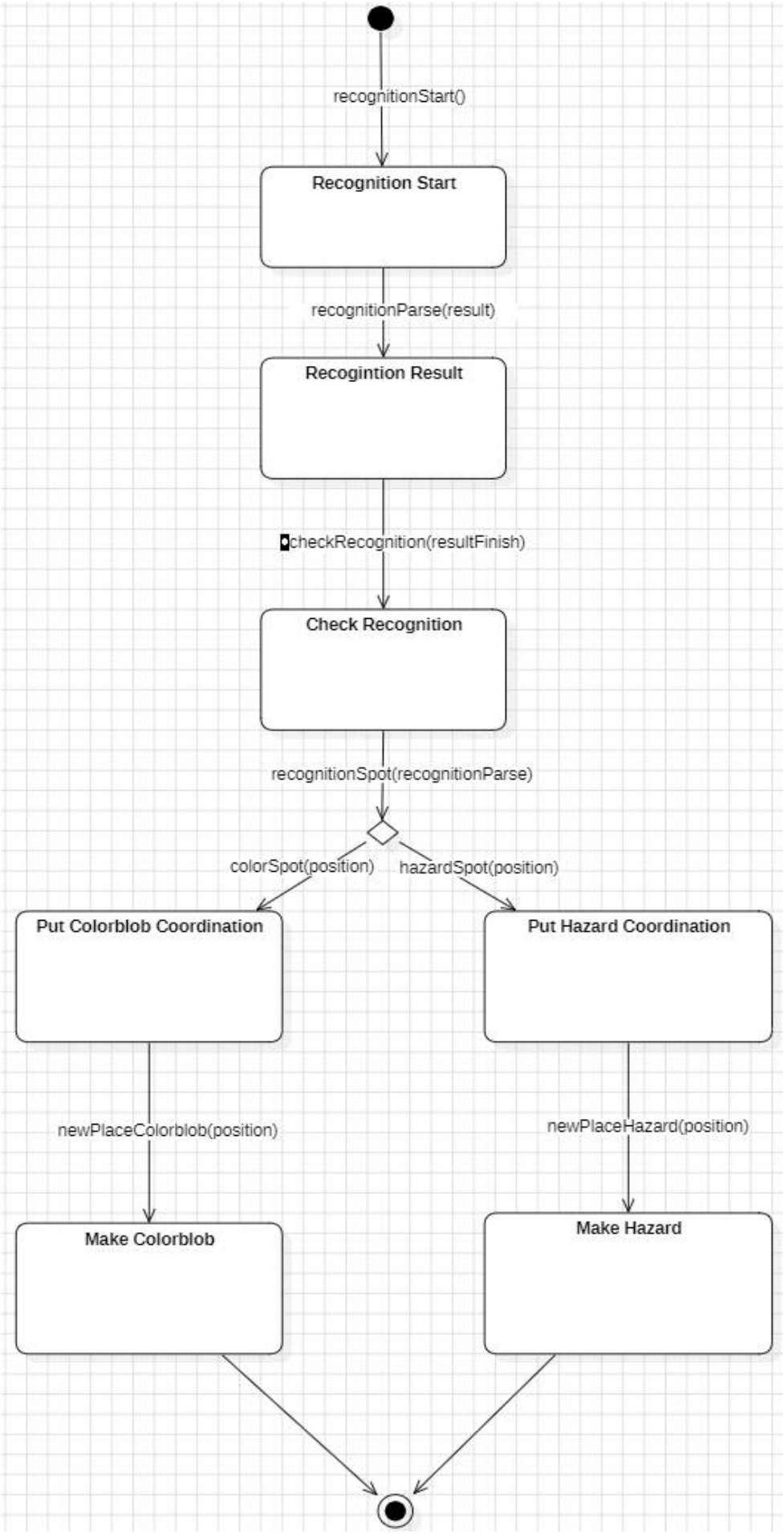
5.2 Robot acts to object 객체 상태도



로봇의 움직임의 상태를 보여준다.

1) Robot Move: 로봇 이동 버튼을 클릭하면 로봇의 이동을 시작하는 신호를 보낸다.
2) Find the path: postData()를 통해 서버에 경로, spot 데이터를 요청한 후 json 데이터를 알맞게 파싱한다.
3) findPath(finalInfo) 함수를 실행시켜 센서 가동, 회전, 이동 함수를 실행시킨다.
4) Hazard sensor: findHazard(locateHazard)를 통해 hazard spot이라면 화면에 표시한다.
5) ColorBlob sensor: findColor(locateColor)를 통해 color blob이라면 화면에 표시한다.
6) Predefined sensor: visitied(fin)을 통해 predefined spot이라면 화면에 표시한다.
7) Move: move(row, col)을 통해 경로 정보를 바탕으로 다음 좌표로 로봇을 움직인다.
8) Rotate: rotate(direction)을 통해 방향 정보를 바탕으로 다음 방향으로 로봇을 회전한다.

5.3 AI Voice Recognition 객체 상태도



음성인식 상태를 나타낸다.

1) Recognition Start: 음성 인식 시작 버튼이 클릭되면 recognitionStart() 함수가 실행되어 음성 인식이 시작된다.

2) Recognition Result: recognitionParse(result) 함수를 통해 음성 인식 결과를 파싱한다.

3) Check Recognition: checkRecognition(recognitionParse) 함수를 통해 음성 인식 데이터가 알맞은지 확인한다.

4) recognitionSpot(recognitionParse) 함수를 통해 color blob인지, hazard spot인지 판별한다.

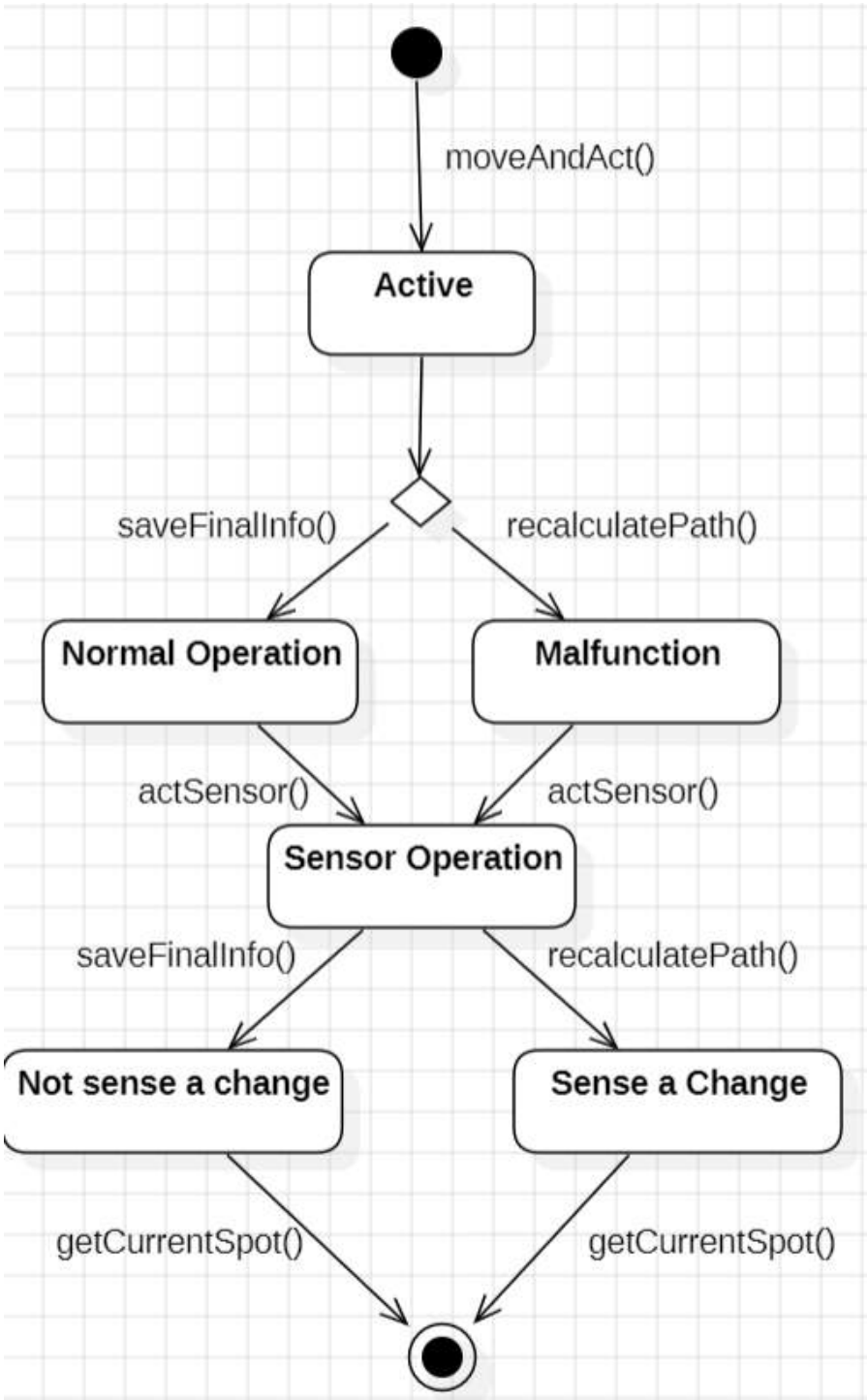
5) Put Colorblob Coordination: colorSpot(position)을 실행시켜 음성 인식으로 새로 추가된 color blob 좌표를 설정한다.

6) Make Colorblob: newPlaceColorblob(position)을 실행시켜 화면에 새로운 color blob을 표시한다.

7) Put Hazard Coordination: hazardSpot(position)을 실행시켜 인식으로 새로 추가된 hazard spot 좌표를 설정한다.

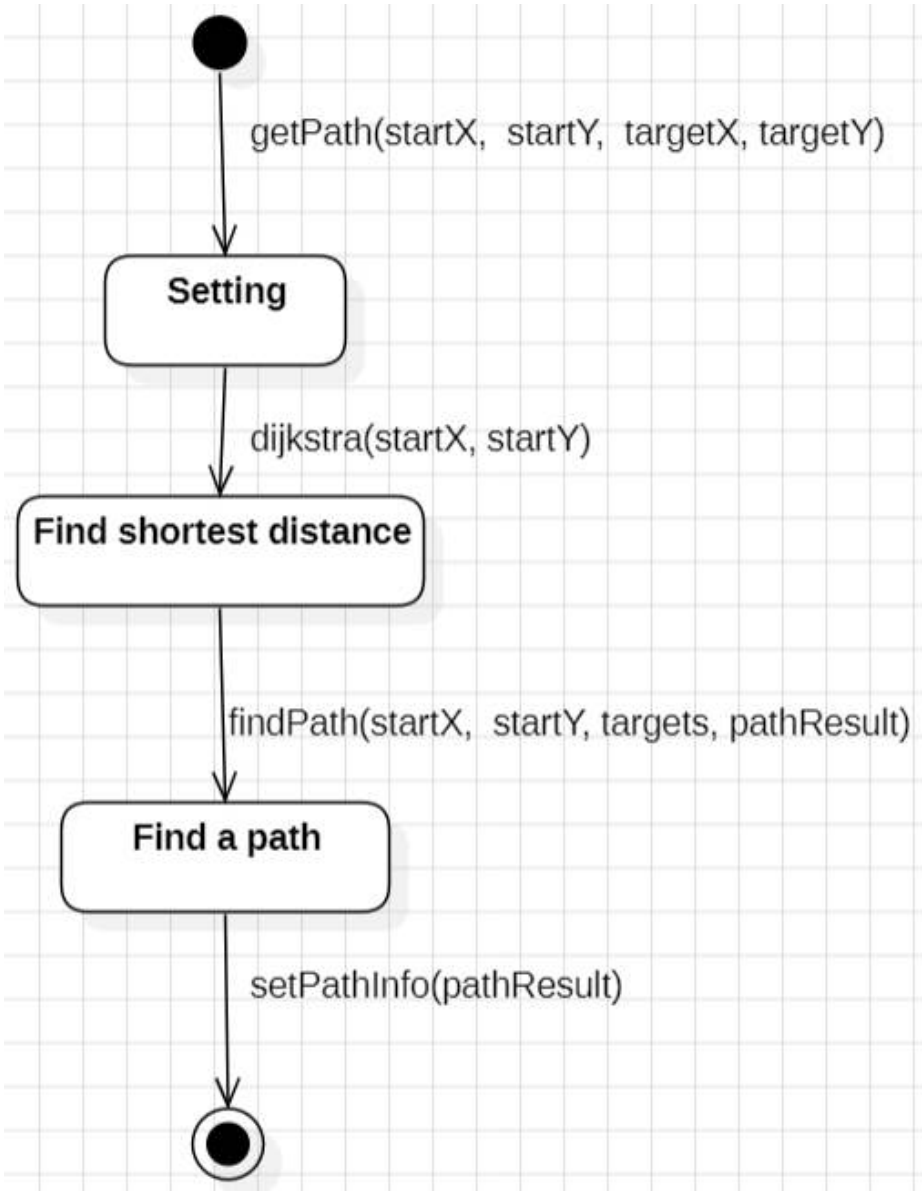
8) Make Hazard: newPlazceHazard(postion)을 실행시켜 화면에 새로운 hazard spot을 표시한다.

5.4 Command robot action 객체 상태도



1. moveAndAct()가 호출되면 로봇이 동작 과정에 들어간다.
2. 로봇의 오작동 가능성에 따라 90%확률로 정상 작동하며 이를 최종 경로에 저장한다. 5%확률로 0칸 이동하거나 5%확률로 2칸 이동하는 경우 경로가 변경될 수 있음에 따라 recalculatePath()로 경로 재설정에 들어간다.
3. 정상 작동 혹은 오작동이 일어나면 로봇의 동작이 발생함에 따라 센서가 가동된다. 센서가 가동되어 해저드가 탐지 된 경우 경로를 재탐색하고, 해저드가 탐지되지 않은 경우 경로를 저장한다.
4. 현재 위치에 따라 목적지의 탐색 여부를 파악하여 동작을 마무리한다.

5.5 Make path 객체 상태도



1. getPath()를 통해 시작점과 목표지점에 대한 정보를 받아와 기본적인 경로에 대한 초기 정보를 설정한다.
2. 다익스트라 알고리즘을 통하여 각 목적지까지의 최단 경로 탐색을 진행한다.
3. findPath()메서드는 여러 개의 목적지를 먼저 출발지점과의 거리 순으로 재배열 하고, 가장 가까운 목적지부터 다익스트라 알고리즘을 통해 해저드를 피할 수 있는 최적 경로를 탐색한다.
4. 최종적인 경로가 모두 생성되면 setPathInfo를 통해 경로를 결정 및 저장하게 된다.