TEAM 2

Traditional vs Vibe

강병완 202211248
강현준 202211251
박　완 202211301
정민수 202211365

Table of Content

Summary for Changes

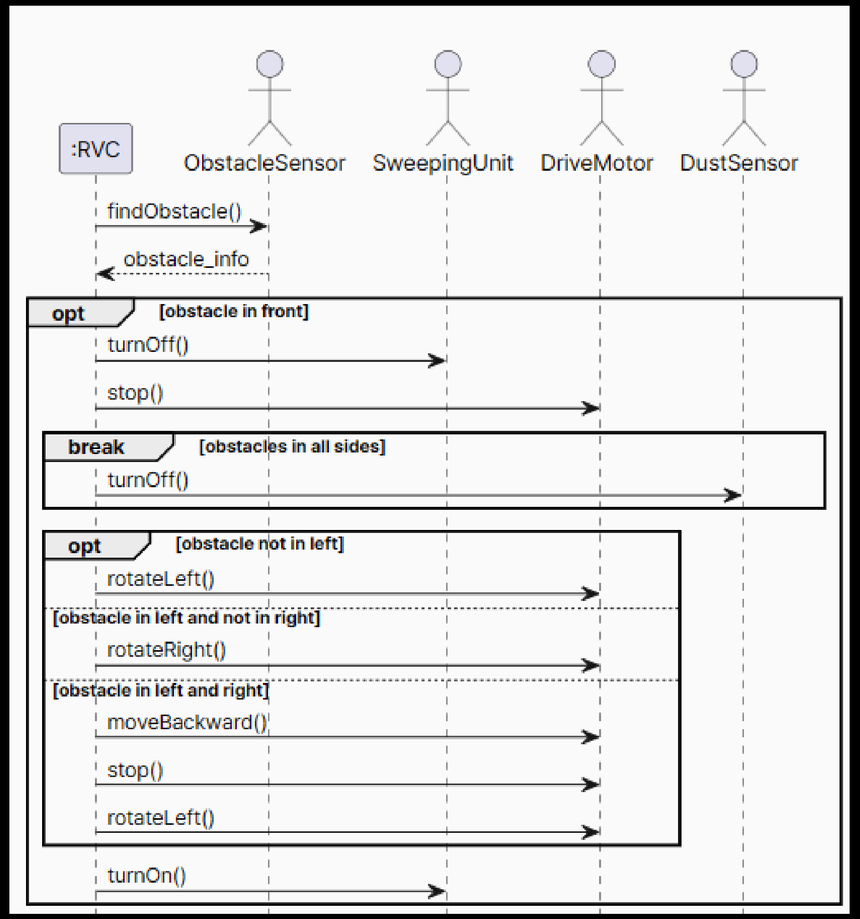
- UC-07에 대한 수정
- System Sequence Diagram – UC-07에 대한 수정
- Sequence Diagram – UC-07에 대한 수정
- Class Diagram – AbstractDriveController, AbstractObstacleSensor에 대한 수정
- Code – AbstractDriveController, AbstractObstacleSensor, DriveController, ObstacleSensor, CleaningController에 대한 수정
- Unit Test – 수정된 클래스에 대한 Test 수정
- System Test – Code, UT에 대한 수정 진행 후 일괄 진행

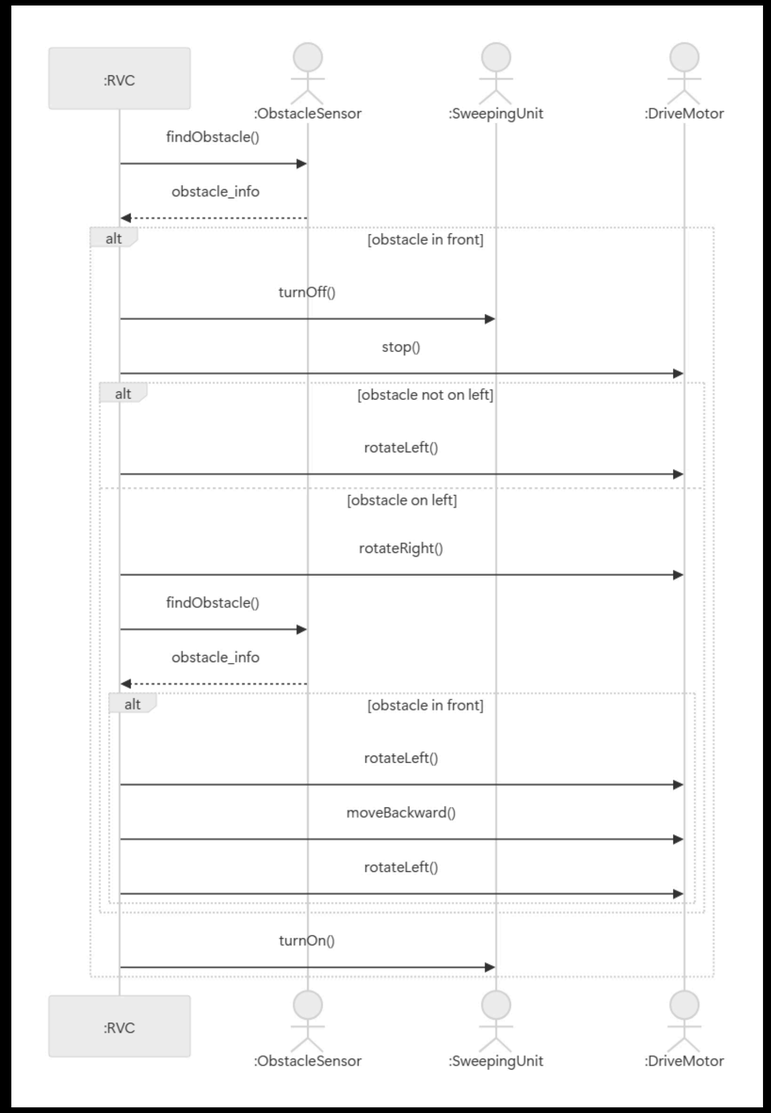
Use case

Name	7. 장애물 감지 및 회피
Actor	ObstacleSensor(Supporting), SweepingUnit(Supporting), DriveMotor(Supporting), ~~DustSensor(Supporting)~~[제거]
Pre-Requisites	RVC의 전원이 꺼져있는 상태이다.
Typical Courses of Events	(R): RVCSystem, (S): SweepingUnit, (M): DriveMotor, (O): ObstacleSensor, ~~(D): DustSensor~~[제거] 1. (R)이 (O)에게 장애물 감지를 요청한다. 2. (O)가 (R)에게 장애물 정보를 알려준다. 3. 정면에서 장애물이 감지되었다면, (R)이 (S)를 끈다. 4. (R)이 (M)의 이동을 중단한다. 5. 좌측에 장애물이 없다면, (R)은 (M)이 좌측으로 회전하도록 한다. 6. (R)이 (S)를 켠다.
Alternative Courses of Events	~~Line 5: 좌측에 장애물이 있고 우측에 장애물이 없다면,~~ ~~(R)은 (M)이 우측으로 회전하도록 한다. [제거]~~ 1. 좌측에 장애물이 있다면, (R)은 (M)이 우측으로 회전하도록 한다. 2. (R)이 (O)에게 장애물 감지를 요청한다. 3. 전방에 장애물이 있다면, (R)은 (M)이 좌측으로 회전 후 후진, 좌측으로 회전하도록 한다. [추가] ~~Line 5: 좌측과 우측에 모두 장애물이 있다면, (R)은 (M)이 후진하도록 한 뒤 정지시키고 좌측으로 회전하도록 한다.[제거]~~
Exceptional Courses of Events	~~Line 5~6: 사방에 장애물이 있다면, (R)은 (D)의 전원을 끄고 청소를 종료한다. [제거]~~

System Sequence Diagram

UC – 07

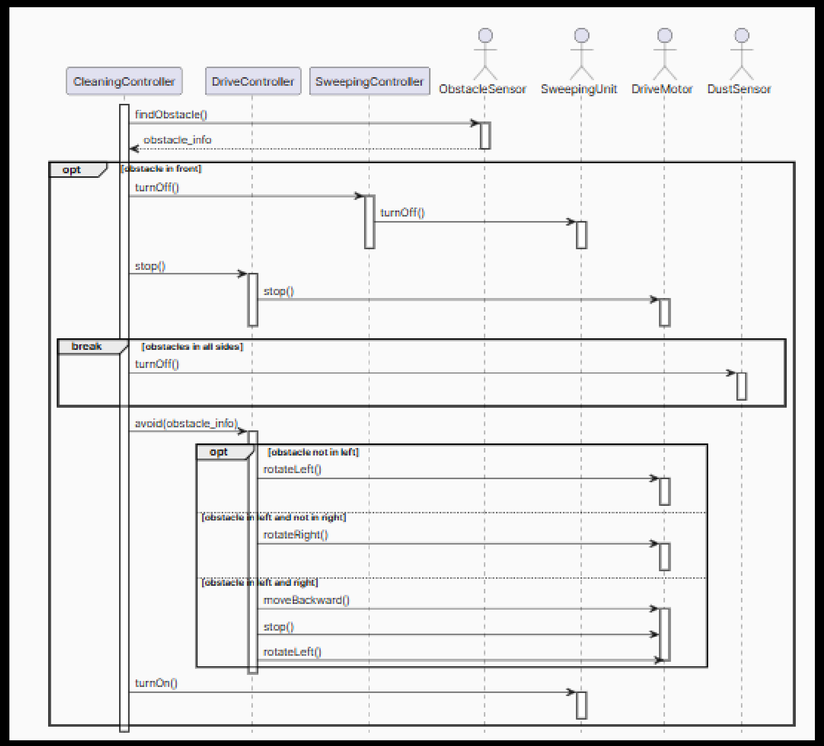


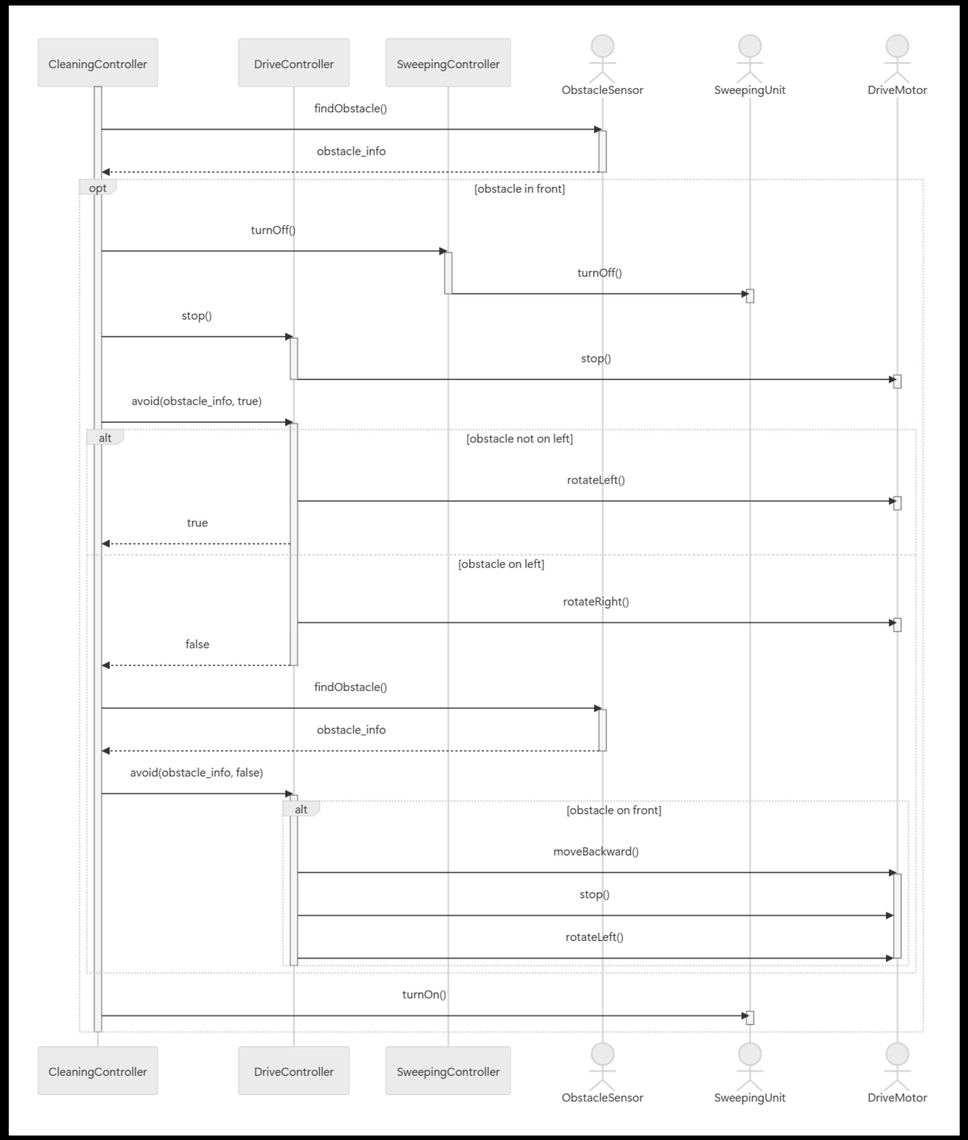


Traditional OOAD

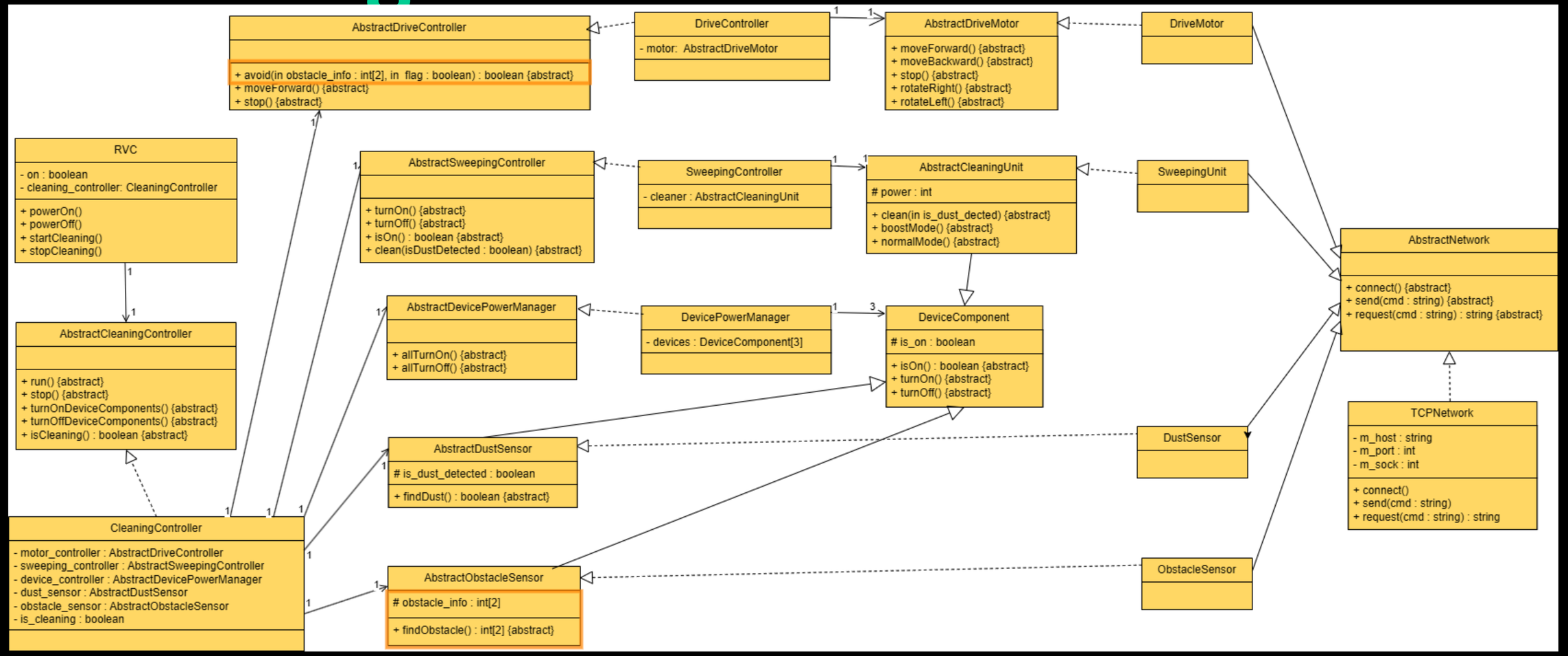
Sequence Diagram

UC – 07





Class Diagram



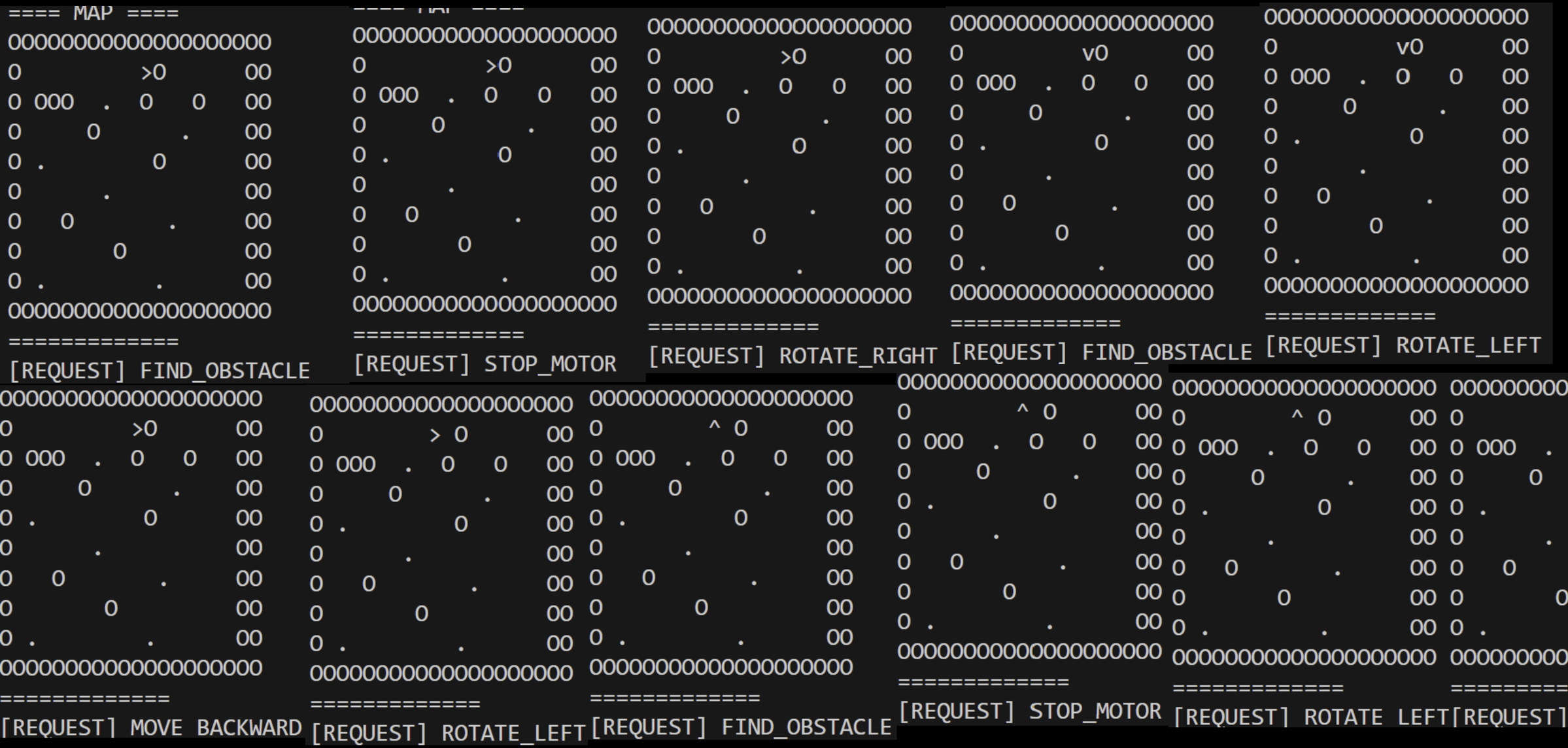
Unit Test

ObstacleSensor

#	Test Group	Test Name	Expected Result
1	ObstacleSensorTest	InitialStateIsOff	생성 직후 `isOn() == false`
2	ObstacleSensorTest	TurnOnMakesSensorOn	`turnOn()` 호출 시 `"OBSTACLE_SENSOR_ON"` 전송 및 `isOn() == true`
3	ObstacleSensorTest	TurnOffMakesSensorOff	`turnOn()` 후 `turnOff()` 호출 시 `"OBSTACLE_SENSOR_OFF"` 전송 및 `isOn() == false`
4	ObstacleSensorTest	FindObstacleReturnsZerosWhenPowerIsOff	전원 OFF 상태에서 `findObstacle()` 호출 시 `{0, 0}` 반환
5	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{0,0}`	`"OBSTACLE 0 0 0 0"` 응답 시 `{0, 0}` 반환
6	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{0,1}`	`"OBSTACLE 0 0 0 1"` 응답 시 `{0, 1}` (front=0, left=1) 반환
7	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{1,0}`	`"OBSTACLE 1 0 0 0"` 응답 시 `{1, 0}` (front=1, left=0) 반환
8	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{1,1}`	`"OBSTACLE 1 0 0 1"` 응답 시 `{1, 1}` (front=1, left=1) 반환
9	ObstacleSensorTest	FindObstacleReturnsZerosOnInvalidResponse	잘못된 응답(`"GARBAGE"`) 수신 시 `{0, 0}` 반환
10	ObstacleSensorTest	FindObstacleReturnsZerosOnEmptyResponse	빈 응답(`""`) 수신 시 `{0, 0}` 반환
11	ObstacleSensorTest	FindObstacleRequestsNetworkEveryCall	`findObstacle()` 2회 호출 시 매번 `"FIND_OBSTACLE"` 요청 전송 (각각 `{1,0}`, `{0,1}` 반환)
12	ObstacleSensorTest	TurnOffPreventsObstacleSensorFromQuerying	`turnOn()` 후 `turnOff()` 시 `findObstacle()` 호출해도 `"FIND_OBSTACLE"` 요청 없이 `{0, 0}` 반환

#	Test Group	Test Name	Expected Result
1	CleaningControllerMockTest	ConstructorThrowsWhenDriveControllerIsNull	DriveController가 null이면 `std::invalid_argument` 예외 발생
2	CleaningControllerMockTest	ConstructorThrowsWhenSweepingControllerIsNull	SweepingController가 null이면 `std::invalid_argument` 예외 발생
3	CleaningControllerMockTest	ConstructorThrowsWhenDevicePowerManagerIsNull	DevicePowerManager가 null이면 `std::invalid_argument` 예외 발생
4	CleaningControllerMockTest	ConstructorThrowsWhenDustSensorIsNull	DustSensor가 null이면 `std::invalid_argument` 예외 발생
5	CleaningControllerMockTest	ConstructorThrowsWhenObstacleSensorIsNull	ObstacleSensor가 null이면 `std::invalid_argument` 예외 발생
6	CleaningControllerMockTest	RunOnly	`run()` 호출 후 `isCleaning() == true`
7	CleaningControllerMockTest	StopOnly	청소 중이 아닐 때 `stop()` 호출 시 `isCleaning() == false` 유지
8	CleaningControllerMockTest	RunThenStop	`run()` 후 `stop()` 호출 시 `isCleaning() == false`
9	CleaningControllerMockTest	RunTwiceSecondCallReturnsEarly	`run()` 두 번 호출 시 두 번째는 `isCleaning() == true` 상태라 즉시 반환, `isCleaning() == true` 유지
10	CleaningControllerMockTest	TurnOnThenRun	`turnOnDeviceComponents()` 호출 시 `allTurnOn()` 1회 실행, 이후 `run()` 시 `isCleaning() == true`
11	CleaningControllerMockTest	TurnOnThenRunThenStop	`turnOn()` → `run()` 후 `stop()` 호출 시 `isCleaning() == false`
12	CleaningControllerMockTest	TurnOnThenRunThenTurnOff	`run()` 중 `turnOffDeviceComponents()` 호출 시 `allTurnOff()` 1회 실행, `isCleaning()` 유지
13	CleaningControllerMockTest	TurnOnThenRunThenStopAgain	`turnOn()` → `run()` → `stop()` 후 `isCleaning() == false`
14	CleaningControllerMockTest	TurnOnThenTurnOff	`allTurnOn()` → `allTurnOff()` 순서 보장, `isCleaning() == false`
15	CleaningControllerMockTest	TurnOffThenTurnOn	`allTurnOff()` → `allTurnOn()` 순서 보장, `isCleaning() == false`
16	CleaningControllerMockTest	TurnOnThreeTimes	`turnOnDeviceComponents()` 3회 호출 시 `allTurnOn()` 3회 실행
17	CleaningControllerMockTest	ObstacleDetectedThenAvoidAndMoveForward	장애물 감지 시 1차 `avoid(_, 0)` 성공 → 2차 `avoid(_, 1)` 수행, 먼지 감지 후 청소기 ON → `moveForward()` 호출
18	CleaningControllerMockTest	ObstacleDetectedButAvoidFails	장애물 감지 시 1차 `avoid(_, 0)` 실패 → 내부 분기 skip, 청소기 OFF, `moveForward()` 미호출
19	CleaningControllerTest (Real)	RealDeviceInitialStateIsNotCleaning	실제 장치 연결 후 초기 상태 `isCleaning() == false`
20	CleaningControllerTest (Real)	RealDeviceTurnOnAndTurnOffComponents	실제 장치에서 `turnOn()` → `turnoff()` 후 `isCleaning() == false`
21	CleaningControllerTest (Real)	RealDeviceStopWhenNotCleaningDoesNothing	실제 장치에서 청소 중이 아닐 때 `stop()` 호출해도 `isCleaning() == false` 유지

System Test


==== MAP ====
[REQUEST] FIND_OBSTACLE
[REQUEST] STOP_MOTOR
[REQUEST] ROTATE_RIGHT
[REQUEST] FIND_OBSTACLE
[REQUEST] ROTATE_LEFT
[REQUEST] MOVE BACKWARD
[REQUEST] ROTATE_LEFT
[REQUEST] FIND_OBSTACLE
[REQUEST] STOP_MOTOR
[REQUEST] ROTATE LEFT
[REQUEST] FIND DUS

System Test

Traditional OOAD

Test ID	Test Description	Preconditions	Test Steps	Expected Result
ST-01	초기 상태 전원 OFF 확인	서버 실행 중	RVC 객체 생성 후 `isOn()` 호출	`isOn() == false`
ST-02	전원 ON 시 상태 전환	서버 실행 중	`powerOn()` 호출	`isOn() == true`
ST-03	전원 ON 중복 호출 멱등성	서버 실행 중	`powerOn()` 2회 연속 호출	`isOn() == true` (변화 없음)
ST-04	전원 ON → OFF 상태 전환	서버 실행 중	`powerOn()` → `powerOff()` 호출	`isOn() == false`
ST-05	전원 ON 없이 OFF 호출	서버 실행 중	`powerOff()` 단독 호출	`isOn() == false` 유지
ST-06	전원 OFF → ON 역순 호출	서버 실행 중	`powerOff()` → `powerOn()` 호출	`isOn() == true`
ST-07	ON → OFF → ON 반복 사이클	서버 실행 중	`powerOn()` → `powerOff()` → `powerOn()`	최종 `isOn() == true`
ST-08	전원 OFF 상태에서 청소 시작 시도	서버 실행 중	`powerOn()` 없이 `startCleaning()` 호출	`isOn() == false`, `isCleaning() == false`
ST-09	청소 시작 후 중지	서버 실행 중	`powerOn()` → `startCleaning()` → `stopCleaning()` (1.2초 대기)	`isCleaning() == false`
ST-10	전체 ON-청소-중지-OFF 사이클	서버 실행 중	`powerOn()` → `startCleaning()` → `stopCleaning()` → `powerOff()`	`isOn() == false`, `isCleaning() == false`

Test ID	Test Description	Preconditions	Test Steps	Expected Result
ST-11	초기 위치 장애물 감지 — 전방 개방	`RESET` + `LOAD_MAP 1`, `(1,1)` facing RIGHT	`OBSTACLE_SENSOR_ON` + `FIND_OBSTACLE`	`"OBSTACLE 0 1 1 1"` (전방 free, 좌/우/후방 벽)
ST-12	초기 셀 먼지 없음	`RESET` + `LOAD_MAP 1`	`DUST_SENSOR_ON` + `FIND_DUST`	`"DUST 0"` (시작 셀 `' '`)
ST-13	16칸 전진 후 행 끝에서 전방 벽 감지	`RESET` + `LOAD_MAP 1`	`MOVE_FORWARD` × 16 → `FIND_OBSTACLE`	응답 `f=1` (`(1,18)='O'`)
ST-14	행 끝(1,17)에서 좌회전(UP) → 천장 벽 감지	`RESET` + `LOAD_MAP 1`	`MOVE_FORWARD` × 16 → `ROTATE_LEFT` → `FIND_OBSTACLE`	응답 `f=1` (`(0,17)='O'`)
ST-15	두 번 좌회전(LEFT) → 복도 방향 전방 개방	`RESET` + `LOAD_MAP 1`	`MOVE_FORWARD` × 16 → `ROTATE_LEFT` × 2 → `FIND_OBSTACLE`	응답 `f=0` (`(1,16)=' '`)
ST-16	후진 시 벽에 막혀 위치 유지	`RESET` + `LOAD_MAP 1`, `(1,1)` facing RIGHT	`MOVE_BACKWARD` (후방 `(1,0)='O'`) → `GET_POSITION`	`"POS 1 1 1"` (위치 변화 없음)
ST-17	복도형 맵에서 전체 RVC 청소 사이클	`RESET` + `LOAD_MAP 1`	`powerOn()` → `startCleaning()` → `stopCleaning()` → `powerOff()`	`isOn() == false`, `isCleaning() == false`

Test ID	Test Description	Preconditions	Test Steps	Expected Result
ST-18	시작 셀에 먼지 존재	`RESET` + `LOAD_MAP 2`, `(1,1)='.'`	`DUST_SENSOR_ON` + `FIND_DUST`	`"DUST 1"`
ST-19	전방 먼지 셀은 장애물 아님	`RESET` + `LOAD_MAP 2`	`OBSTACLE_SENSOR_ON` + `FIND_OBSTACLE`	`"OBSTACLE 0 1 1 1"`
ST-20	`CLEANER_ON` + `FIND_DUST` → 셀 청소 후 재확인	`RESET` + `LOAD_MAP 2`	`CLEANER_ON` → `FIND_DUST` (1회) → `FIND_DUST` (2회)	1회: `"DUST 1"`, 2회: `"DUST 0"`
ST-21	전진 이동 시 도착 셀 자동 청소	`RESET` + `LOAD_MAP 2`	`MOVE_FORWARD` → `FIND_DUST`	`"DUST 0"` (`(1,2)` 이동 중 자동 청소)
ST-22	2칸 전진 후 위치 확인	`RESET` + `LOAD_MAP 2`	`MOVE_FORWARD` × 2 → `GET_POSITION`	`"POS 1 3 1"`
ST-23	벽 앞에서 추가 전진 시 위치 유지 + 전방 장애물	`RESET` + `LOAD_MAP 2`, `(1,3)` 도달 후	`MOVE_FORWARD` × 3 → `FIND_OBSTACLE` + `GET_POSITION`	`f=1` (`(1,4)='O'`), 위치 `"POS 1 3 1"`
ST-24	대각선 맵에서 전체 RVC 청소 사이클	`RESET` + `LOAD_MAP 2`	`powerOn()` → `startCleaning()` → `stopCleaning()` → `powerOff()`	`isOn() == false`, `isCleaning() == false`

Test ID	Test Description	Preconditions	Test Steps	Expected Result
ST-25	개방 공간 초기 위치 전방 개방	`RESET` + `LOAD_MAP 3`	`OBSTACLE_SENSOR_ON` + `FIND_OBSTACLE`	`"OBSTACLE 0 1 1 1"`
ST-26	개방 공간 5칸 전진 후 위치 확인	`RESET` + `LOAD_MAP 3`	`MOVE_FORWARD` × 5 → `GET_POSITION`	`"POS 1 6 1"`
ST-27	개방 구역 내 먼지 없음	`RESET` + `LOAD_MAP 3`	`DUST_SENSOR_ON` + `FIND_DUST`	`"DUST 0"`
ST-28	16칸 전진 후 행 끝(1,17)에서 전방 벽 감지	`RESET` + `LOAD_MAP 3`	`MOVE_FORWARD` × 16 → `FIND_OBSTACLE`	응답 `f=1` (`(1,18)='O'`)
ST-29	우측 통로(col 17)로 남하 — (1,17) → 우회전 → 2칸 → (3,17)	`RESET` + `LOAD_MAP 3`, `(1,17)` 도달 후	`ROTATE_RIGHT` → `MOVE_FORWARD` × 2 → `GET_POSITION`	`"POS 3 17 2"`
ST-30	미로형 맵에서 전체 RVC 청소 사이클	`RESET` + `LOAD_MAP 3`	`powerOn()` → `startCleaning()` → `stopCleaning()` → `powerOff()`	`isOn() == false`, `isCleaning() == false`

Coverage

DriveMotor.cpp	22	22	0	0	100.00%
DustSensor.cpp	19	18	1	0	94.74%
ObstacleSensor.cpp	33	26	3	4	78.79%
RVC.cpp	25	23	1	1	92.00%

Name	↑ HEAD	Patch %	Change
> project/tests/CleangingControllerTest.cpp	96.34%	100.00%	+6.34%

Vibe Coding

Summary for Changes

- UC-03/UC-04에 대한 수정
- System Sequence Diagram – UC-03/UC-04에 대한 수정
- Sequence Diagram – UC-03/UC-04에 대한 수정
- Class Diagram 수정
- Unit Test – 수정된 클래스에 대한 Test 수정
- System Test – Code, UT에 대한 수정 진행 후 일괄 진행

Use case

Vibe Coding

UC-03: 전방 장애물 회피

항목	내용
ID	UC-03
이름	전방 장애물 회피
Primary Actor	Front Sensor
간략 설명	전방 장애물이 감지되고 RVC가 완전히 포위되지 않은 경우, RVC가 정지하고 개방된 측면으로 회전한 뒤 전진 navigation을 재개한다.
사전 조건	청소 세션이 활성 상태. Front Sensor가 True를 발생시킴. 좌측 또는 우측이 차단되지 않음.
사후 조건	RVC가 새로운 방향으로 전진 중이며, Cleaner는 On 상태를 유지한다.

→

UC-03 전방 장애물 회피

Name	전방 장애물 회피
Actor	FrontSensor(Primary), LeftSensor(Supporting), DriveMotor(Supporting), Cleaner(Supporting), RightScan(Supporting)
Pre-Requisites	RVC가 CLEANING 또는 INTENSIFYING 상태로 주행 중이며 전방 장애물이 감지되었다.
Typical Courses of Events	(R): RVCSystem, (F): FrontSensor, (L): LeftSensor, (M): DriveMotor, (C): Cleaner 1. (F)가 전방 장애물 interrupt를 발생시킨다. 2. (R)은 interrupt가 정상 주행 중인지 확인하고 (M)에 STOP을 명령한다. 3. 다음 tick에서 (R)은 (L)을 확인한다. 4. 왼쪽이 비어 있으면 (R)은 (M)에 LEFT를 명령하고 청소 상태로 복귀한다. 5. 왼쪽이 막혔으면 (R)은 (M)에 RIGHT를 명령하고 Right Scan을 준비한다. 6. 다음 tick에서 (R)은 FrontSensor로 기존 오른쪽 방향을 확인한다. 7. 오른쪽이 비어 있으면 청소 상태로 복귀한다.
Alternative Courses of Events	Line 6: 오른쪽도 막혔으면 UC-04로 확장한다.
Exceptional Courses of Events	회피 시퀀스 중 발생한 front interrupt는 Right Scan의 거짓 interrupt일 수 있으므로 무시하고 tick 진행을 계속한다.

기존 UC-03은 전방 장애물 감지 후 왼쪽/오른쪽 센서를 확인해
바로 회피 방향을 결정하는 흐름이었다.
변경 후 UC-03은 다음처럼 여러 tick에 나뉜다.

Use case

Vibe Coding

UC-04: 포위 상태 탈출

항목	내용
ID	UC-04
이름	포위 상태 탈출
Primary Actor	Front Sensor, Left Sensor, Right Sensor
간략 설명	세 방향(전방, 좌측, 우측) 모두에서 장애물이 감지되면, RVC가 후진하고 가능한 측면으로 회전한 뒤 재개한다.
사전 조건	청소 세션이 활성 상태. Front = True AND Left = True AND Right = True.
사후 조건	RVC가 새로운 방향으로 전진 중이며, Cleaner는 On 상태를 유지한다.

기존에는 “전방/왼쪽/오른쪽이 막힘”을 센서 3개로 판단했다면,
지금은 “전방/왼쪽/Right Scan 결과가 모두 Blocked”일 때 포위 상태로 본다.

UC-04 포위 상태 탈출

Name	포위 상태 탈출
Actor	FrontSensor(Supporting), LeftSensor(Supporting), RightScan(Supporting), DriveMotor(Supporting), Cleaner(Supporting)
Pre-Requisites	UC-03의 side evaluation에서 전방, 왼쪽, Right Scan 결과가 모두 blocked로 확인되었다.
Typical Courses of Events	(R): RVCSystem, (M): DriveMotor 1. (R)은 (M)에 LEFT를 명령해 Right Scan 이전의 원래 heading을 복구하고 ESCAPING 상태로 진입한다. 2. 다음 tick에서 (R)은 (M)에 BACKWARD를 한 번만 명령한다. 3. (R)은 AVOIDING_OBSTACLE 상태로 돌아가 side evaluation을 다시 수행한다.
Alternative Courses of Events	후진 후 왼쪽 또는 Right Scan 방향이 열리면 해당 방향으로 회피 후 청소를 계속한다.
Exceptional Courses of Events	전체 맵 커버리지 보장, 방문 상태, BFS, zigzag 주행은 본 Use Case 범위가 아니다.

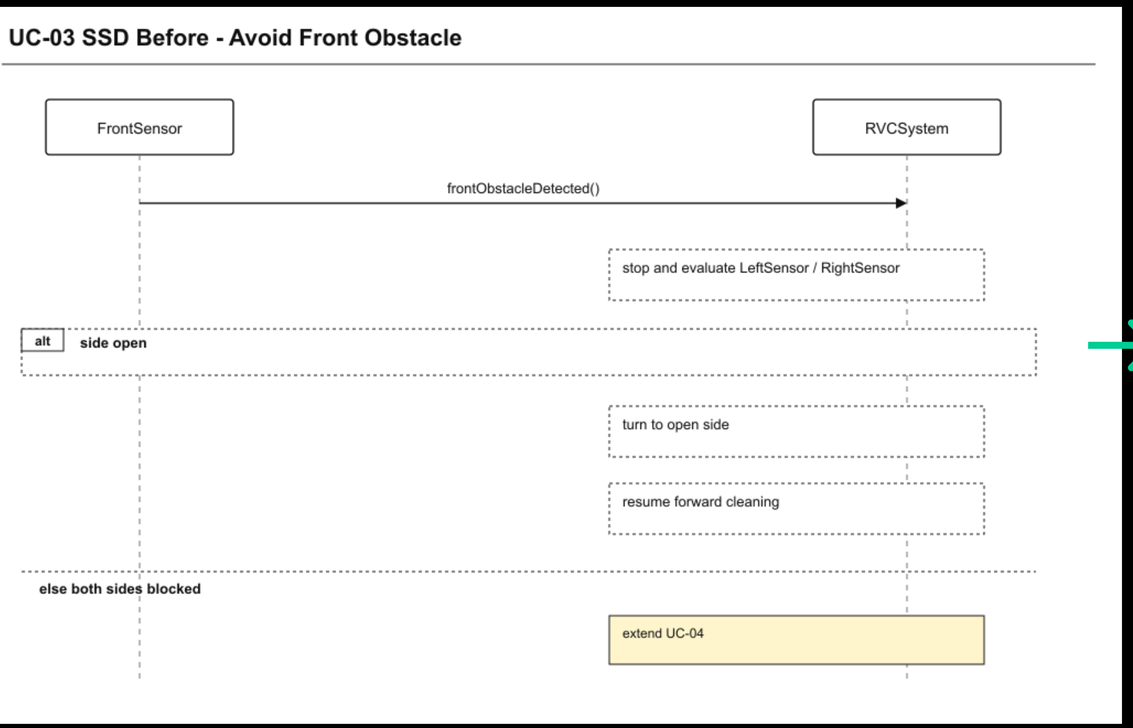
그리고 후진도 한 이벤트에서 몰아서 처리하지 않고:

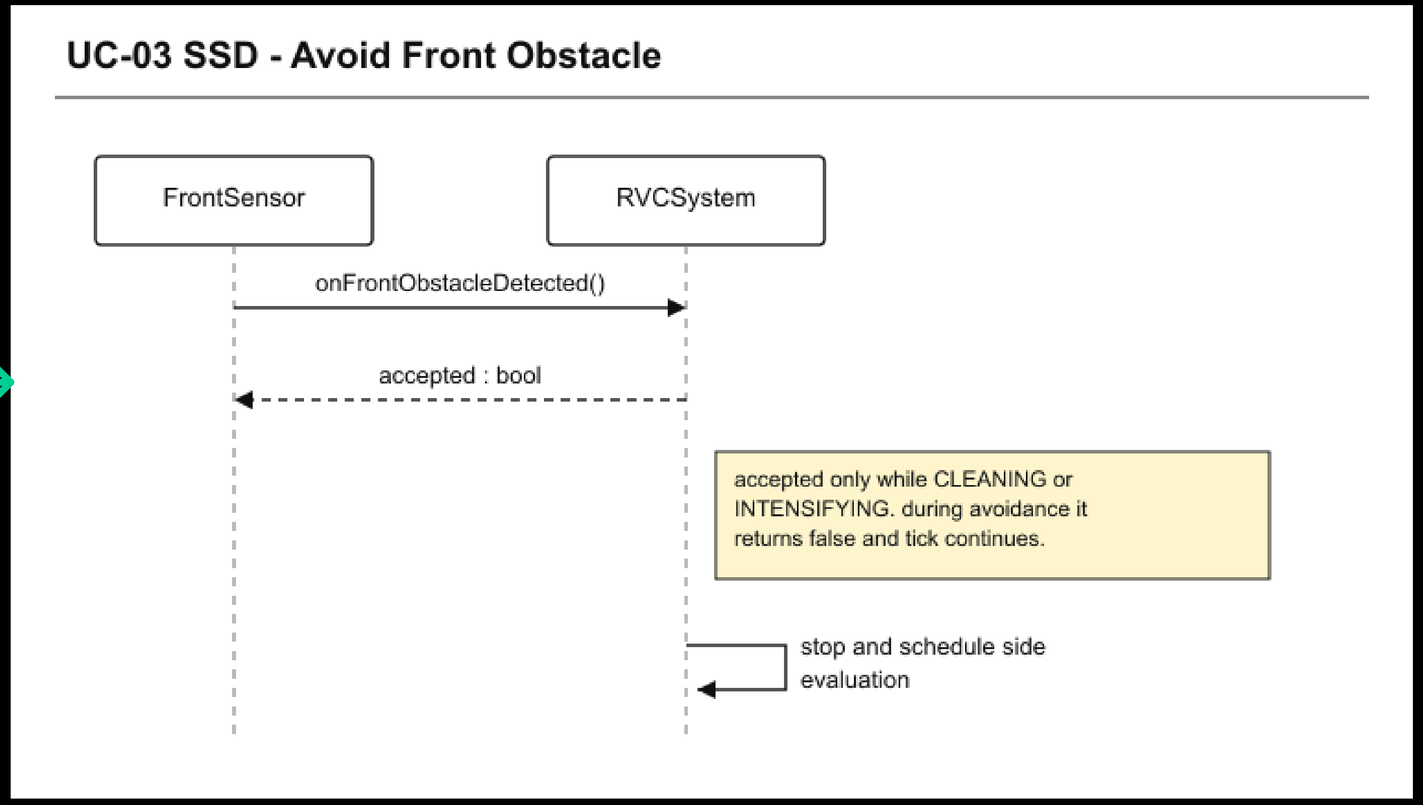
1. 원래 heading 복구
2. ESCAPING 진입
3. 다음 tick에서 BACKWARD 한 칸
4. 다시 side evaluation 수행

처럼 tick 단위로 분리. (tick 한 번에 여러 과정을 하는 버그 수정)

System Sequence Diagram

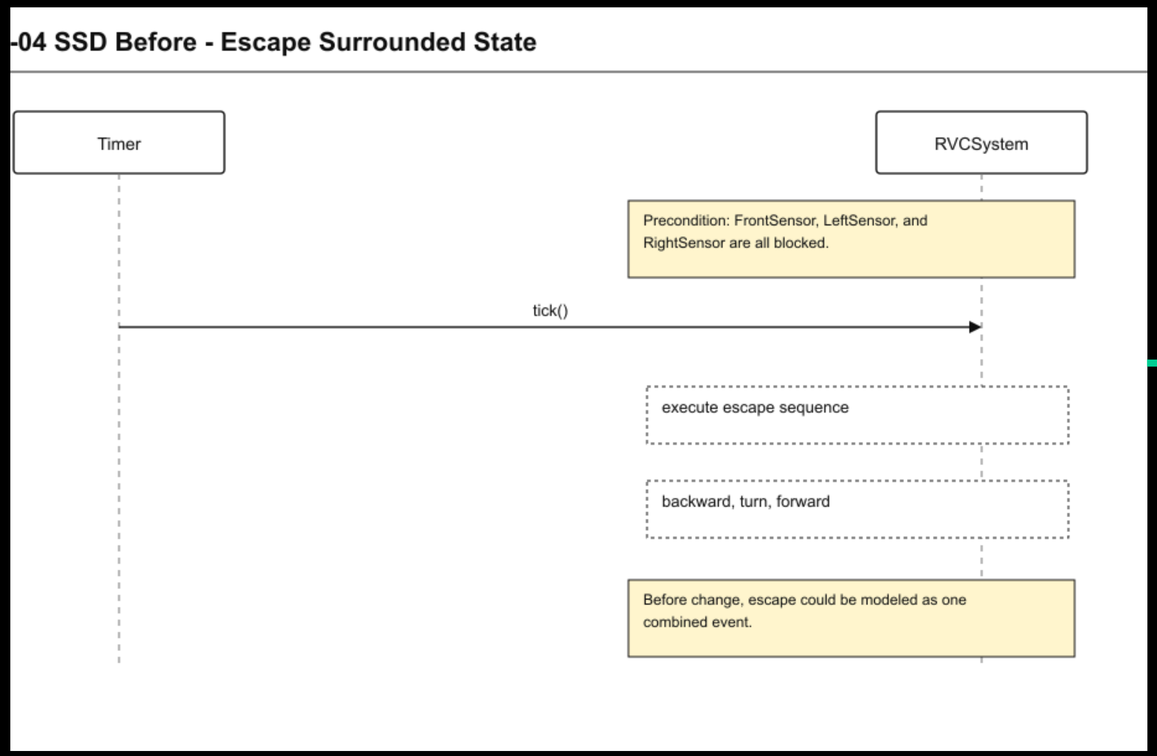
RVCSystem이 전방 장애물 이벤트를 받은 뒤
Side Evaluation을 예약하는 형태



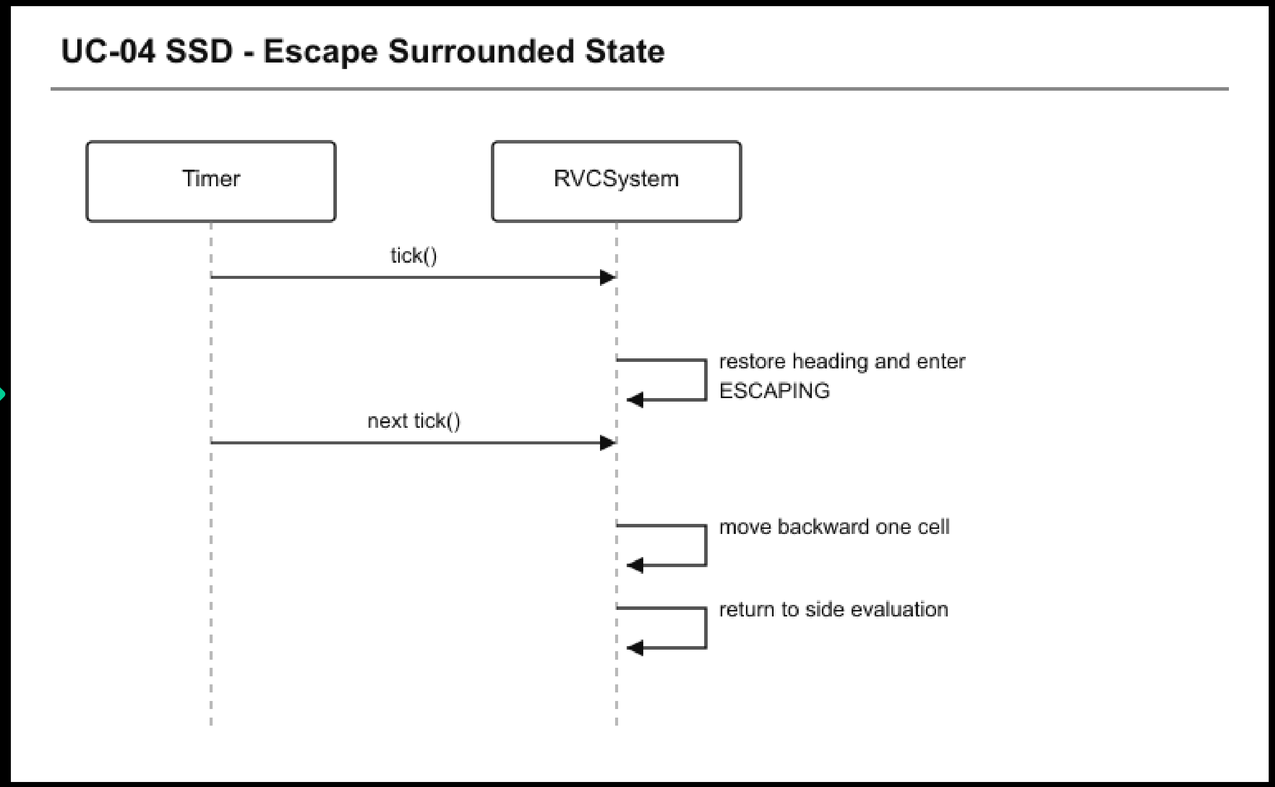


System Sequence Diagram

이거도 uc3와 uc4에 대한 Ssd

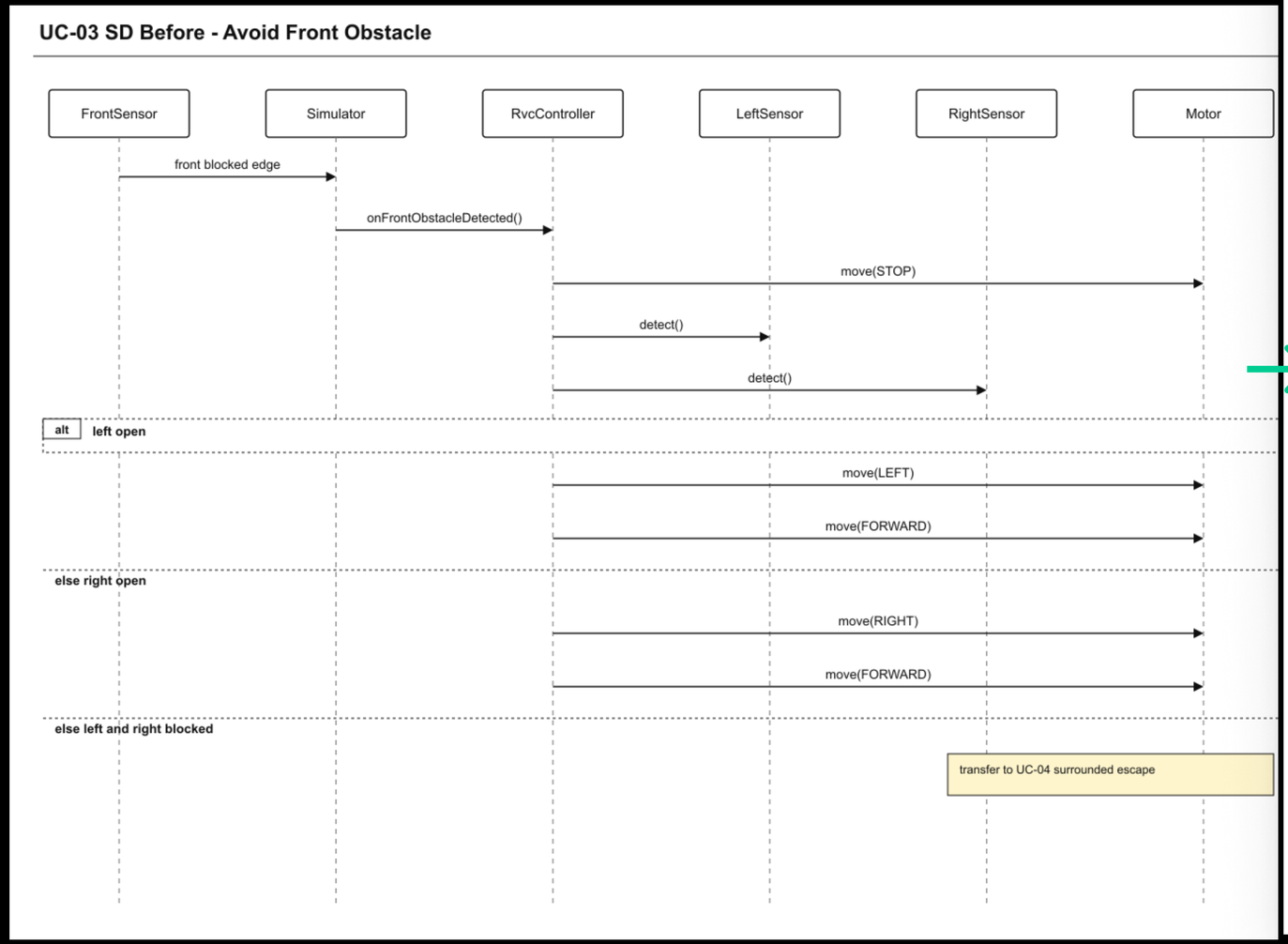


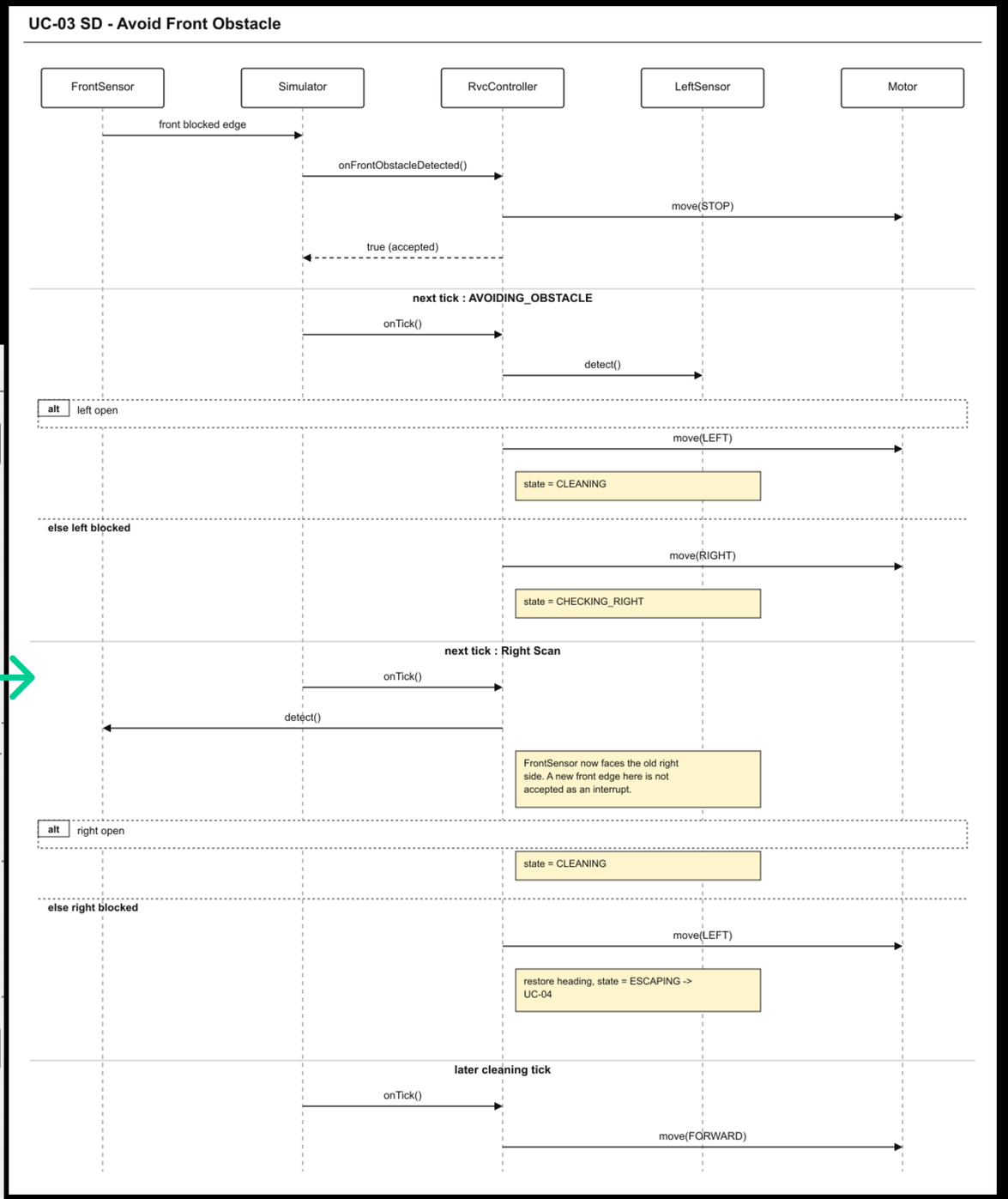
포위 상태의 조건이 RightSensor blocked에서 Right Scan blocked로 바뀜



Sequence Diagram

기존 RightSensor.detect() 대신
Motor.move(RIGHT) 후 FrontSensor.detect()가 추가

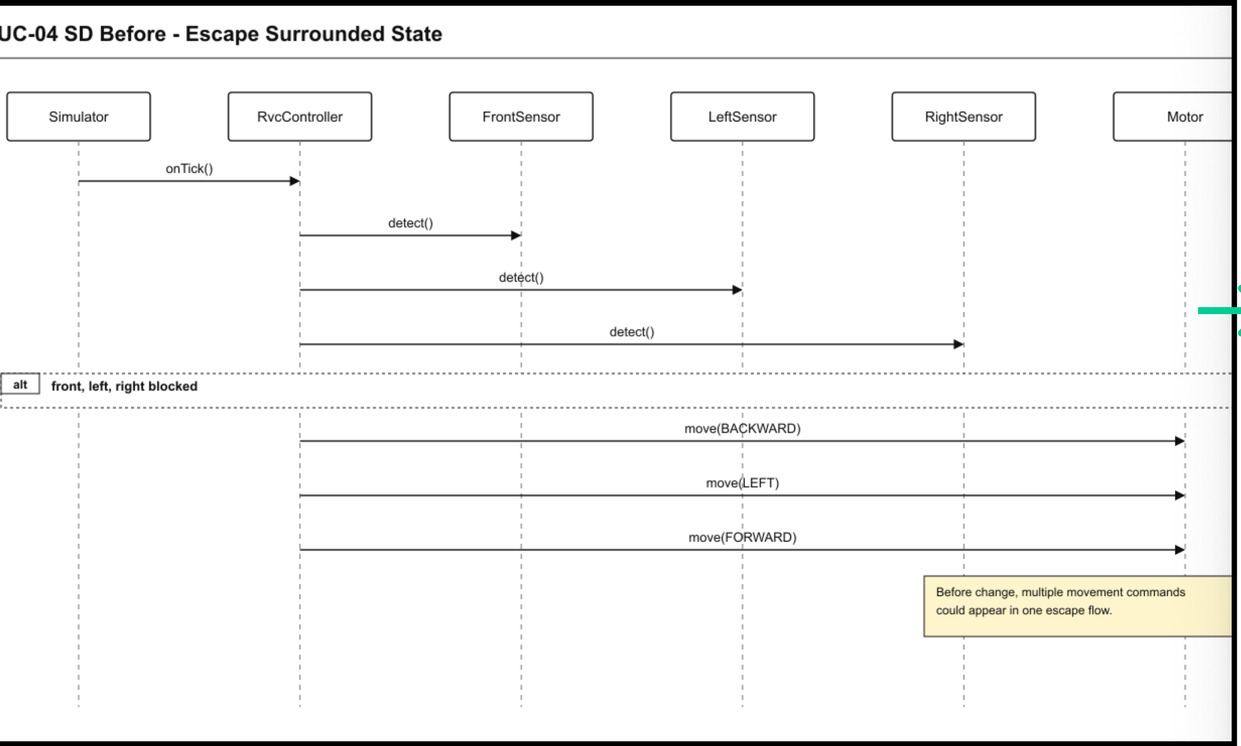


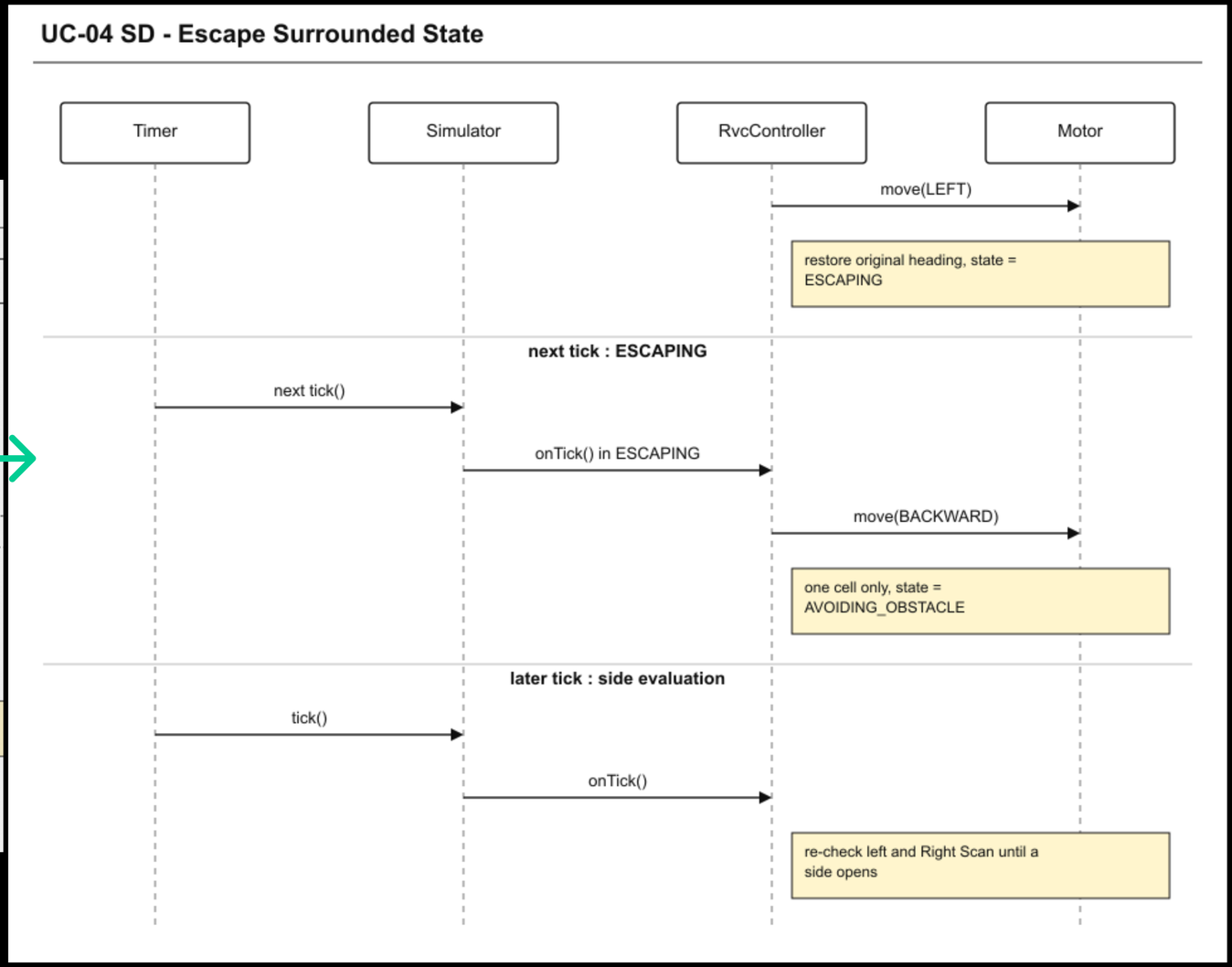


Sequence Diagram

Vibe Coding

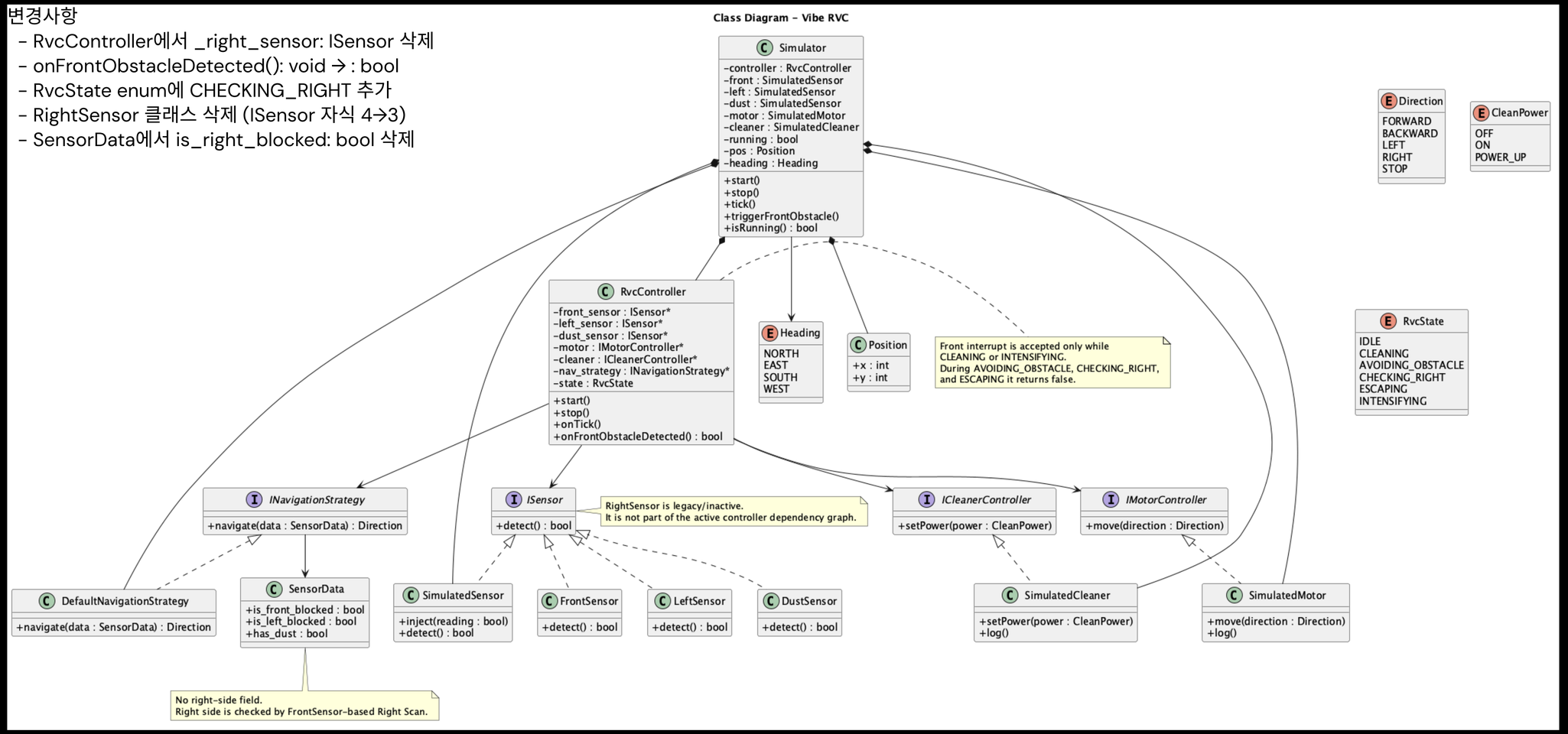
escape 동작이 한 번에 처리되는 흐름에서, tick 단위 흐름으로 분리
LEFT로 heading 복구 -> 다음 tick에서 BACKWARD 한 칸 -> 다시 side evaluation





Class Diagram

변경사항
- RvcController에서 _right_sensor: ISensor 삭제
- onFrontObstacleDetected(): void → : bool
- RvcState enum에 CHECKING_RIGHT 추가
- RightSensor 클래스 삭제 (ISensor 자식 4→3)
- SensorData에서 is_right_blocked: bool 삭제



Unit Test

Test	검증 내용
`ForwardWhenAllClear`	전방·좌측 모두 열림 → `FORWARD`
`LeftWhenFrontBlockedLeftOpen`	전방 막힘, 좌측 열림 → `LEFT`
`BackwardWhenFrontAndLeftBlocked`	전방·좌측 막힘 → `BACKWARD` (= "우측을 Right Scan으로 확인하라" 신호)
`DustAloneDoesNotAffectDirection`	먼지만 감지돼도 방향은 `FORWARD` 유지

Test	검증 내용
`StartMovesForwardAndCleansOn`	start → Motor `FORWARD`, Cleaner `ON`
`StopHaltsMotorAndTurnsOffCleaner`	stop → Motor `STOP`, Cleaner `OFF`
`TickDoesNothingWhenIdle`	IDLE 상태 tick → 아무 명령 없음

청소 강도 / 먼지 (2)

Test	검증 내용
`DustDetectionPowersUpCleaner`	먼지 감지 → Cleaner `POWER_UP`
`CleaningPowerRestoresAfterIntensifyDuration`	강화 지속시간 후 Cleaner `ON` 복구

Unit Test

Test	검증 내용
`FrontObstacleOnlyStopsThenWaits`	인터럽트는 **STOP만** 하고 다음 tick 대기
`LeftOpenTurnsLeftThenResumesForward`	좌측 열림 → `LEFT` 후 `FORWARD` 복귀
`LeftBlockedRightOpenProbesRightThenResumes`	좌측 막힘 → `RIGHT` probe(Right Scan) 후 복귀
`SurroundedProbesRightThenBacksUpOneCell`	Right Scan blocked 확인 → `LEFT` heading 복구 → 다음 tick `BACKWARD` 한 번

인터럽트 수용 정책 (F-10 핵심) (5)

Test	검증 내용
`FrontObstacleIgnoredWhenIdle`	IDLE 중 인터럽트 → `false`, 명령 없음
`IntensifyingInterruptedByFrontObstacle`	INTENSIFYING 중 인터럽트도 수용 → STOP
`FrontInterruptAcceptedWhileCruising`	정상 주행 중 인터럽트 → `true` 반환, STOP
`FrontInterruptIgnoredDuringAvoidance`	회피(AVOIDING) 중 인터럽트 → `false` (무시)
`FrontInterruptIgnoredWhileEscaping`	탈출(ESCAPING) 중 인터럽트 → `false`, 후진 연쇄 안 끊김

System Test

Vibe Coding

2. System / 통합 Test (16개) — SimulatorTest

실제 격자에 장애물/먼지를 놓고 tick을 돌려, 위치·방향이 의도대로 변하는지 통합 검증.

기본 / 생명주기 (6)

Test	검증 내용
`StartMovesForwardAndCleansOn`	start 시 FORWARD/ON으로 시작
`NormalTickReissuesForward`	일반 tick은 FORWARD 유지
`DustPowersUpCleanerThenRestores`	먼지 칸에서 POWER_UP 후 ON 복구
`StopHaltsMotorAndCleaner`	stop 시 STOP/OFF
`TickIgnoredWhenIdle`	start 전 tick 무시
`RunningStateChangesOnlyByStartAndStop`	running 상태는 start/stop으로만 변경

이동 정확성 (2)

Test	검증 내용
`NeverMovesMoreThanOneCellPerTick`	어떤 tick에서도 **한 칸 초과 이동 금지**
`FrontInterruptFiresOnceWhileStuck`	같은 막힘 상태에서 인터럽트 반복 발화 안 함 (edge-trigger)

System Test

Vibe Coding

장애물 회피 / 막다른 길 탈출 (8)

Test	검증 내용
`FrontObstacleAvoidanceResumesForward`	전방 장애물 회피 후 FORWARD 복귀
`SurroundedEscapeBacksUpThenResumesForward`	포위 시 후진 후 재평가
`DeadEndBacksUpMultipleTimes`	막다른 통로에서 여러 tick에 걸쳐 후진
`BacksUpAlongOriginalHeadingAfterProbe`	Right Scan 후 원래 heading 기준으로 후진
`EscapesDeadEndCorridorThroughLeftGap`	**왼쪽 출구** 있는 막다른 통로 탈출
`EscapesDeadEndCorridorThroughRightGap`	**오른쪽 출구** 있는 막다른 통로 탈출
`BacksUpFullCorridorThenEscapes`	긴 통로 끝까지 후진한 뒤 탈출
`AvoidanceInterruptDoesNotBreakBackupChain`	회피 중 거짓 인터럽트가 후진 연쇄를 끊지 않음 (F-10 회귀)

Coverage

interfaces/ICleanerController.hpp		100.0%	2 / 2
interfaces/IMotorController.hpp		100.0%	2 / 2
interfaces/INavigationStrategy.hpp		100.0%	2 / 2
interfaces/ISensor.hpp		100.0%	2 / 2
simulator/SimulatedCleaner.hpp		100.0%	6 / 6
simulator/SimulatedMotor.hpp		100.0%	6 / 6
simulator/SimulatedSensor.hpp		100.0%	2 / 2
simulator/Simulator.cpp		82.7%	86 / 104

System Test

Traditional OOAD

Test ID	Test Description	Preconditions	Test Steps	Expected Result
ST-01	초기 상태 전원 OFF 확인	서버 실행 중	RVC 객체 생성 후 `isOn()` 호출	`isOn() == false`
ST-02	전원 ON 시 상태 전환	서버 실행 중	`powerOn()` 호출	`isOn() == true`
ST-03	전원 ON 중복 호출 멱등성	서버 실행 중	`powerOn()` 2회 연속 호출	`isOn() == true` (변화 없음)
ST-04	전원 ON → OFF 상태 전환	서버 실행 중	`powerOn()` → `powerOff()` 호출	`isOn() == false`
ST-05	전원 ON 없이 OFF 호출	서버 실행 중	`powerOff()` 단독 호출	`isOn() == false` 유지
ST-06	전원 OFF → ON 역순 호출	서버 실행 중	`powerOff()` → `powerOn()` 호출	`isOn() == true`
ST-07	ON → OFF → ON 반복 사이클	서버 실행 중	`powerOn()` → `powerOff()` → `powerOn()`	최종 `isOn() == true`
ST-08	전원 OFF 상태에서 청소 시작 시도	서버 실행 중	`powerOn()` 없이 `startCleaning()` 호출	`isOn() == false`, `isCleaning() == false`
ST-09	청소 시작 후 중지	서버 실행 중	`powerOn()` → `startCleaning()` → `stopCleaning()` (1.2초 대기)	`isCleaning() == false`
ST-10	전체 ON-청소-중지-OFF 사이클	서버 실행 중	`powerOn()` → `startCleaning()` → `stopCleaning()` → `powerOff()`	`isOn() == false`, `isCleaning() == false`

Vibe Coding

장애물 회피 / 막다른 길 탈출 (8)

Test	검증 내용
`FrontObstacleAvoidanceResumesForward`	전방 장애물 회피 후 FORWARD 복귀
`SurroundedEscapeBacksUpThenResumesForward`	포위 시 후진 후 재평가
`DeadEndBacksUpMultipleTimes`	막다른 통로에서 여러 tick에 걸쳐 후진
`BacksUpAlongOriginalHeadingAfterProbe`	Right Scan 후 원래 heading 기준으로 후진
`EscapesDeadEndCorridorThroughLeftGap`	**왼쪽 출구** 있는 막다른 통로 탈출
`EscapesDeadEndCorridorThroughRightGap`	**오른쪽 출구** 있는 막다른 통로 탈출
`BacksUpFullCorridorThenEscapes`	긴 통로 끝까지 후진한 뒤 탈출
`AvoidanceInterruptDoesNotBreakBackupChain`	회피 중 거짓 인터럽트가 후진 연쇄를 끊지 않음 (F-10 회귀)

- Traditional OOAD 30개 vs Vibe Coding 20개
- Traditional OOAD는 특정 유형의 맵에 대해서 정상 작동 및 과정을 확인
- Vide Coding은 SW적으로 Simulator을 만들어서 돌림

Unit Test

Traditional OOAD

ObstacleSensor

#	Test Group	Test Name	Expected Result
1	ObstacleSensorTest	InitialStateIsOff	생성 직후 `isOn() == false`
2	ObstacleSensorTest	TurnOnMakesSensorOn	`turnOn()` 호출 시 `"OBSTACLE_SENSOR_ON"` 전송 및 `isOn() == true`
3	ObstacleSensorTest	TurnOffMakesSensorOff	`turnOn()` 후 `turnOff()` 호출 시 `"OBSTACLE_SENSOR_OFF"` 전송 및 `isOn() == false`
4	ObstacleSensorTest	FindObstacleReturnsZerosWhenPowerIsOff	전원 OFF 상태에서 `findObstacle()` 호출 시 `{0, 0}` 반환
5	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{0,0}`	`"OBSTACLE 0 0 0 0"` 응답 시 `{0, 0}` 반환
6	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{0,1}`	`"OBSTACLE 0 0 0 1"` 응답 시 `{0, 1}` (front=0, left=1) 반환
7	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{1,0}`	`"OBSTACLE 1 0 0 0"` 응답 시 `{1, 0}` (front=1, left=0) 반환
8	ObstacleSensorValidInputTest	ParsesValidObstacleResponseWhenPowerIsOn / `{1,1}`	`"OBSTACLE 1 0 0 1"` 응답 시 `{1, 1}` (front=1, left=1) 반환
9	ObstacleSensorTest	FindObstacleReturnsZerosOnInvalidResponse	잘못된 응답(`"GARBAGE"`) 수신 시 `{0, 0}` 반환
10	ObstacleSensorTest	FindObstacleReturnsZerosOnEmptyResponse	빈 응답(`""`) 수신 시 `{0, 0}` 반환
11	ObstacleSensorTest	FindObstacleRequestsNetworkEveryCall	`findObstacle()` 2회 호출 시 매번 `"FIND_OBSTACLE"` 요청 전송 (각각 `{1,0}`, `{0,1}` 반환)
12	ObstacleSensorTest	TurnOffPreventsObstacleSensorFromQuerying	`turnOn()` 후 `turnOff()` 시 `findObstacle()` 호출해도 `"FIND_OBSTACLE"` 요청 없이 `{0, 0}` 반환

Vibe Coding

Test	검증 내용
`StartMovesForwardAndCleansOn`	start → Motor `FORWARD`, Cleaner `ON`
`StopHaltsMotorAndTurnsOffCleaner`	stop → Motor `STOP`, Cleaner `OFF`
`TickDoesNothingWhenIdle`	IDLE 상태 tick → 아무 명령 없음

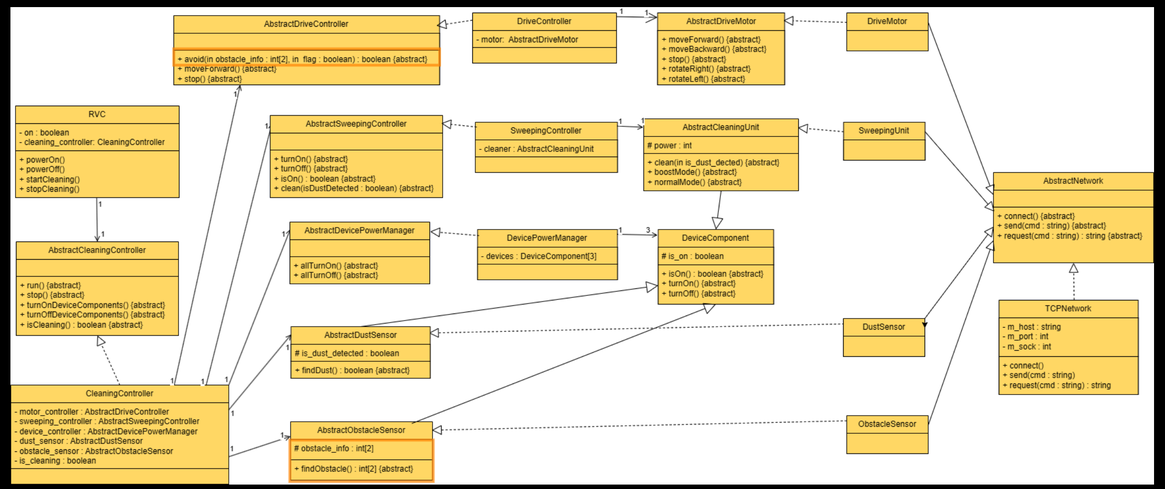
청소 강도 / 먼지 (2)

Test	검증 내용
`DustDetectionPowersUpCleaner`	먼지 감지 → Cleaner `POWER_UP`
`CleaningPowerRestoresAfterIntensifyDuration`	강화 지속시간 후 Cleaner `ON` 복구

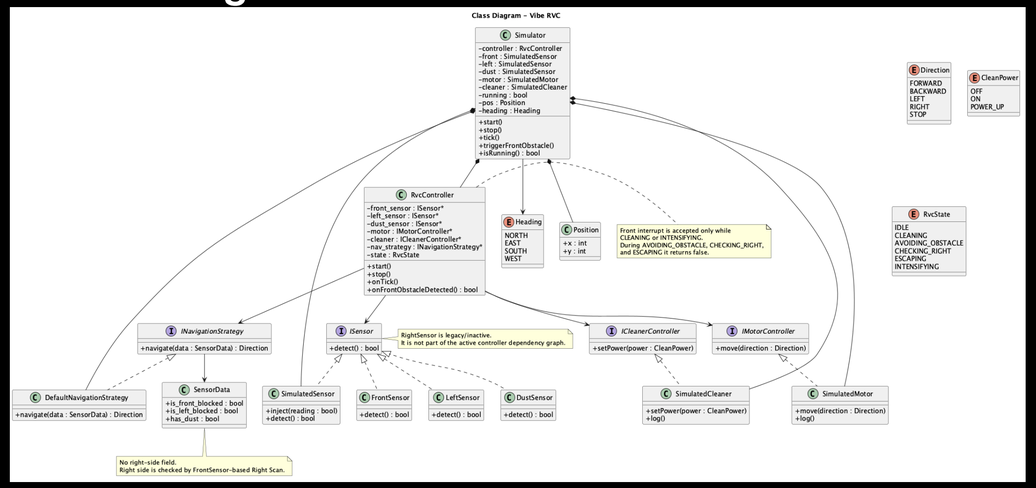
- Traditional OOAD 78개 vs Vibe Coding 34개
- Traditional OOAD는 구현한 주요 Class 별로 Unit Test를 진행
- Vibe Coding은 Navigation 로직과, RVC Controller의 동작 과정을 나누어 Unit Test를 진행

Class Diagram

Traditional OOAD



Vibe Coding

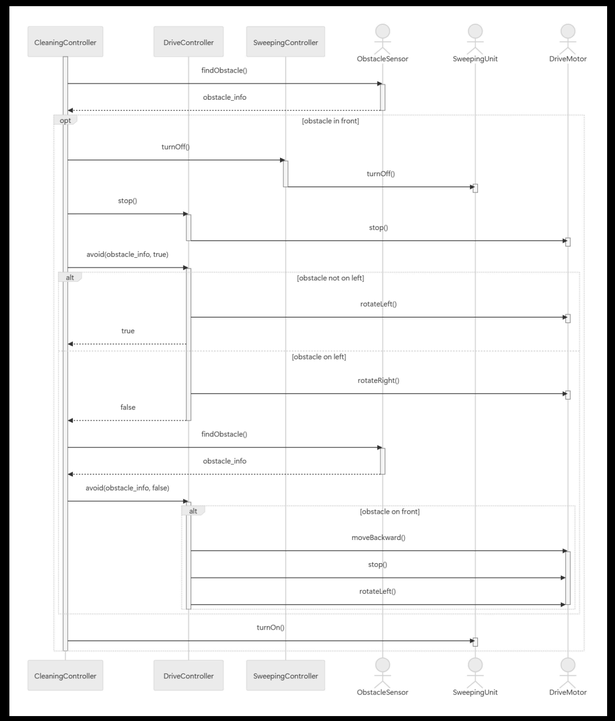


- Traditional OOAD는 총 20개의 Class를 구성
- Traditional OOAD는 필요한 모든 Class에 대해 Abstract Class를 상속받아 구현하는 방식으로 진행
  → 확장 가능한 인터페이스 구조
- Unit Test 시에는 Abstract Class를 상속받은 Mock Class를 통해 Test를 진행

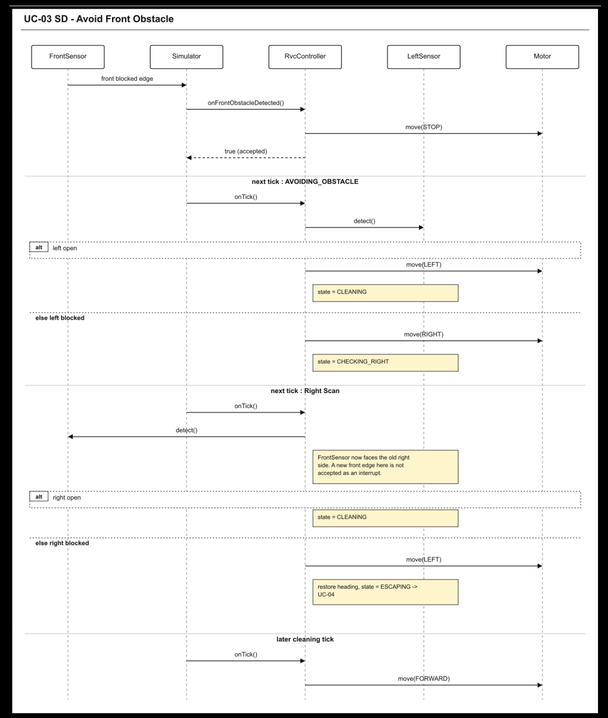
- Vibe Coding은 총 19개 Class 구성
- Vibe Coding은 경계에만 4개 Interface 적용
  → 확장 가능한 구조
- Unit Test는 Interface 구현 Mock으로 진행

Sequence Diagram

Traditional OOAD



Vibe Coding



- Traditional OOAD는 참여하는 객체의 역할을 세분화.
  전체 청소 과정 조율 → CleaningController
  이동에 관한 로직 → DriveController
  청소에 관한 로직 → SweepingController
  필요한 Actuator와 Controller가 소통하는 방식
- Vibe Coding은 RVCController에서 대부분의 로직을 담당
  → 역할의 세분화가 이뤄지지 않음
  → SOLID의 S 단일 책임 원칙을 위반

Use Case

Traditional OOAD

# Ref. #	Aa Use-Case Name	Actor
1	1. 전원 켜기	User (Primary) SweepingUnit(Supporting) ObstacleSensor(Supporting)
2	2. 전원 끄기	User (Primary) SweepingUnit(Supporting) ObstacleSensor(Supporting)
3	3. 청소 시작	User (Primary) SweepingUnit(Supporting) ObstacleSensor(Supporting)
4	4. 청소 종료	User (Primary)
5	5. 전진 이동	DriveMotor(Supporting) SweepingUnit(Supporting)
6	6. 먼지 감지 및 청소	SweepingUnit(Supporting)
7	7. 장애물 감지 및 회피	ObstacleSensor(Supporting) DriveMotor(Supporting) SweepingUnit(Supporting)

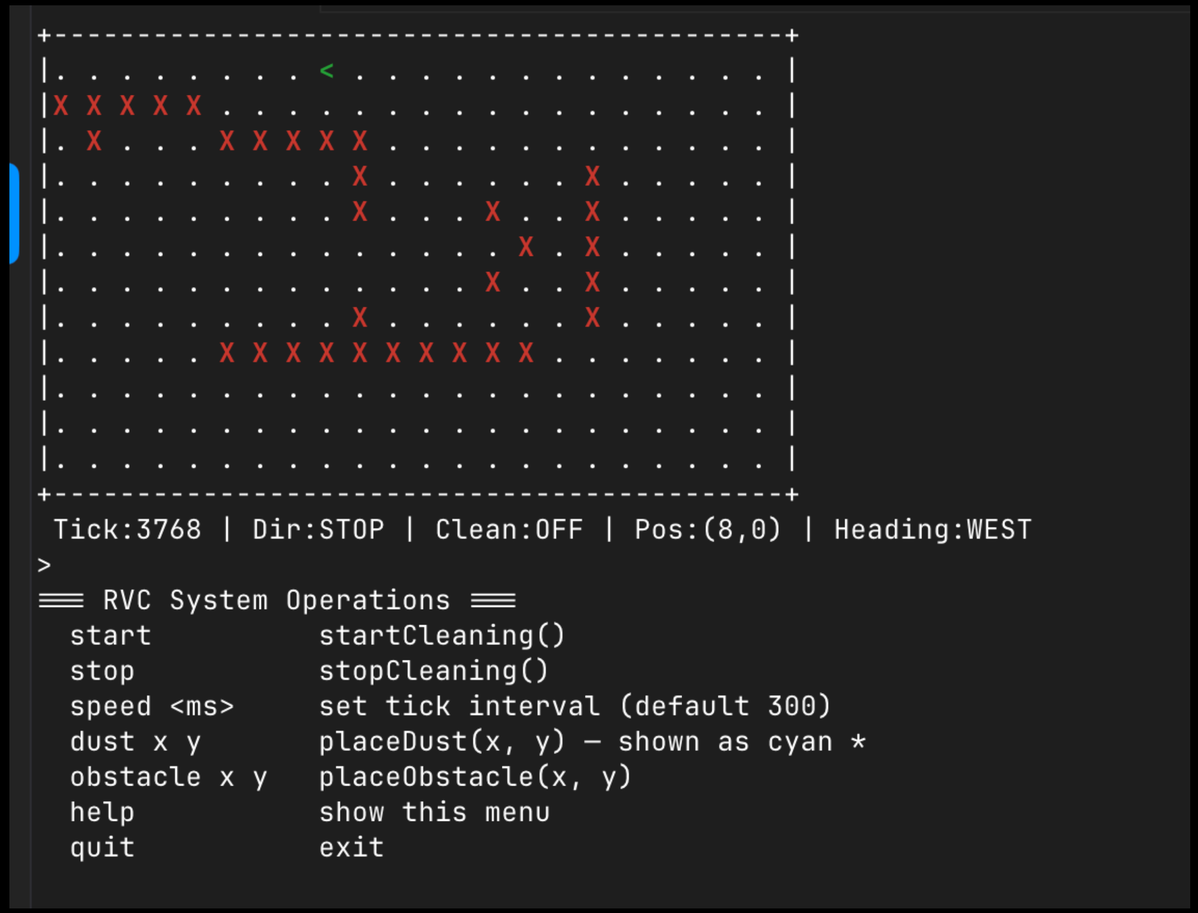
- Traditional OOAD는 총 7개의 Use Case를 제작
- 장애물 회피 로직을 “UC-07 장애물 감지 및 회피”에서 전담
- 시나리오 위주의 자연어 서술

Vibe Coding

Aa Name	ID
청소 세션 시작	UC-01
탐색 및 청소	UC-02
전방 장애물 회피	UC-03
포위 상태 탈출	UC-04
청소 강도 높이기	UC-05
청소 세션 종료	UC-06

- Vibe Coding은 총 6개의 Use Case를 제작
- 장애물 회피 로직을 “UC-03 전방 장애물 회피”, “UC-04 포위 상태 탈출”로 나눠서 구현
- Elaboration 단계임에도 불구하고 구현 방식이 Use case에 깊게 관여됨

Feedback from Traditional OOAD



전통적인 OOAD와 비교하며 Vibe Coding으로 만든 Code는 옆에 있는 사진과 같은 상황이면 제대로 탈출하지 못하고 무한히 빠지는 것을 알 수 있었다.

Vibe Modify 느낀점

pros

- 수정 속도가 압도적으로 빠르다
- 전체 구조를 함께 살펴볼 수 있다
- 문서화 부담을 줄여준다
- 테스트와 검증을 함께 진행할 수 있다

cons

- 요구사항을 과하게 해석할 수 있다

  작은 수정 요청이 더 큰 구조 변경이나
  새로운 기능 제안으로 확장

- 기존 설계 의도를 놓칠 수 있다

  코드만 보고 합리적으로 보이는 수정을 하더라도,
  원래 프로젝트의 의도와 다를 수 있음

- 엄밀한 최종 품질 검수가 필요하다

  테스트가 통과해도 실제 실행 화면을 재생하는 등
  사람의 손길이 닿아야 한다.
  AI는 한 번 놓친 것을 다시 인지하기 어렵다.

Thank you!