

ROS2 주행-1

서비스(음식 배달) 로봇 및 관제 시스템 구축

C-1 느림이조



CONTENTS

ROS2 주행-1 서비스(음식 배달) 로봇 및 관제 시스템 구축



요구사항 문서

- 1 시스템 구성도
- 2 기능적 요구사항
- 3 비기능적 요구사항



시스템 설계 문서

- 1 목표 시스템 개념도
- 2 Flow Chart
- 3 컴포넌트 설명



UI/UX 설계 문서

- 1 와이어프레임
- 2 UI 디자인



프로젝트 계획서

- 1 작업 분해구조
- 2 Gantt Chart
- 3 리소스 배치도



프로젝트 완료

- 1 결과



1. 테이블 오더를 통한 주문
2. 주방 디스플레이를 통한 주문 접수(접수 소리, 알람, 접수, 취소)
3. 주방 디스플레이를 통한 서빙 로봇 제어
4. 서빙 로봇 작동
5. 주문 내역을 데이터 베이스에 저장
6. 데이터 베이스의 데이터를 이용한 통계
7. nav2 param 조정을 통한 로봇 작동 최적화
8. Logging
9. QoS (Qualit of Service)
10. 토픽, 서비스, 액션 각각 한 개 이상 사용



기능적 요구사항

No.	요구사항	상세 내용
1	테이블 오더를 통한 주문	고객이 테이블에서 주문을 입력할 수 있는 기능
2	주방 디스플레이를 통한 주문 접수(접수 소리, 알람, 접수, 취소)	주방에서 주문을 확인하고 처리할 수 있는 디스플레이 기능
3	주방 디스플레이를 통한 서빙 로봇 제어	주방에서 서빙 로봇을 제어하여 음식을 배달할 수 있는 기능
4	서빙 로봇 작동	로봇이 경로를 따라 이동하며 음식을 전달하는 기능
5	주문 내역을 데이터베이스에 저장	모든 주문 내역을 데이터베이스에 기록 및 저장하는 기능
6	데이터베이스의 데이터를 이용한 통계	저장된 데이터를 기반으로 통계 정보를 생성하는 기능
7	토픽, 서비스, 액션 각각 한 개 이상 사용	시스템 내에서 토픽, 서비스, 액션을 활용해 컴포넌트 간의 데이터 전송 및 제어 기능

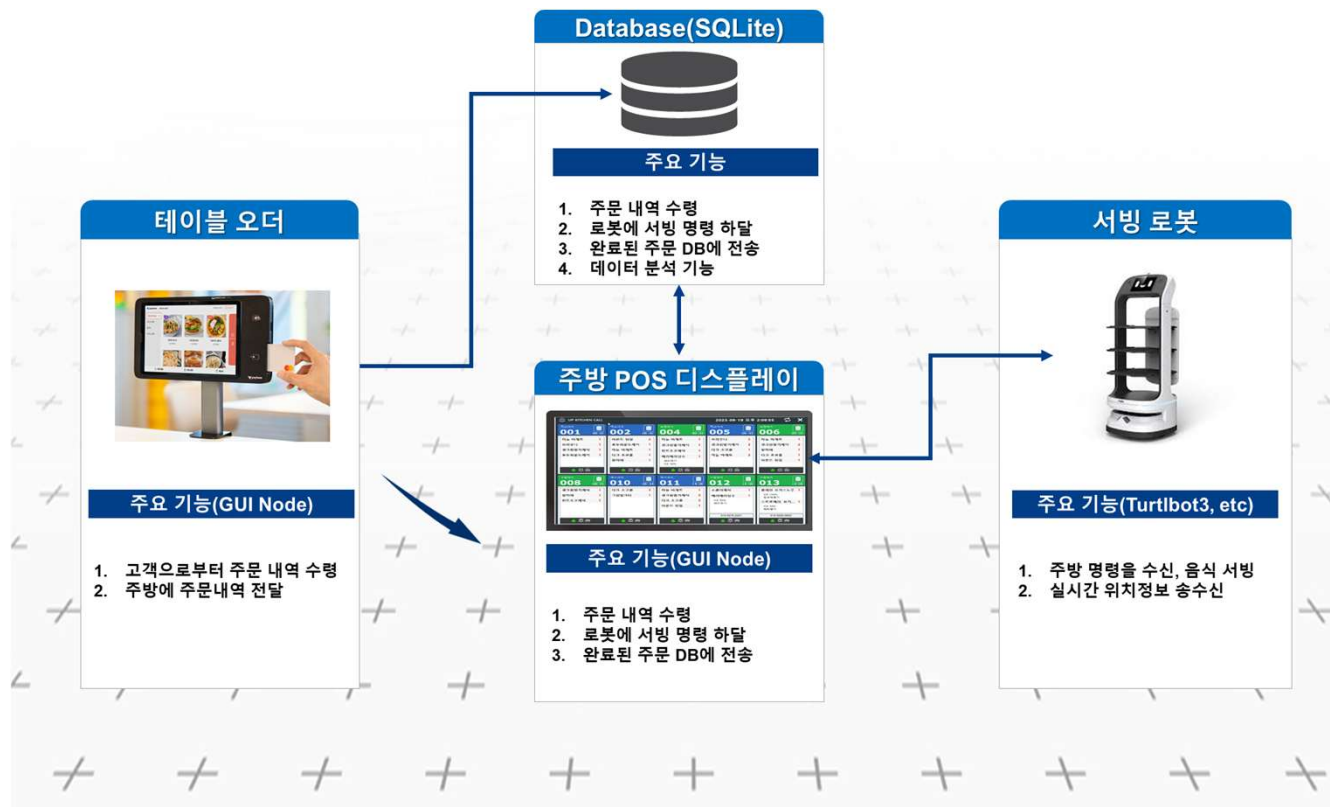


비기능적 요구사항

No.	요구사항	상세 내용
1	nav2 param 조정을 통한 로봇 작동 최적화	고객이 테이블에서 주문을 입력할 수 있는 기능
2	Logging	주방에서 주문을 확인하고 처리할 수 있는 디스플레이 기능
3	QoS (Quality of Service)	주방에서 서빙 로봇을 제어하여 음식을 배달할 수 있는 기능.

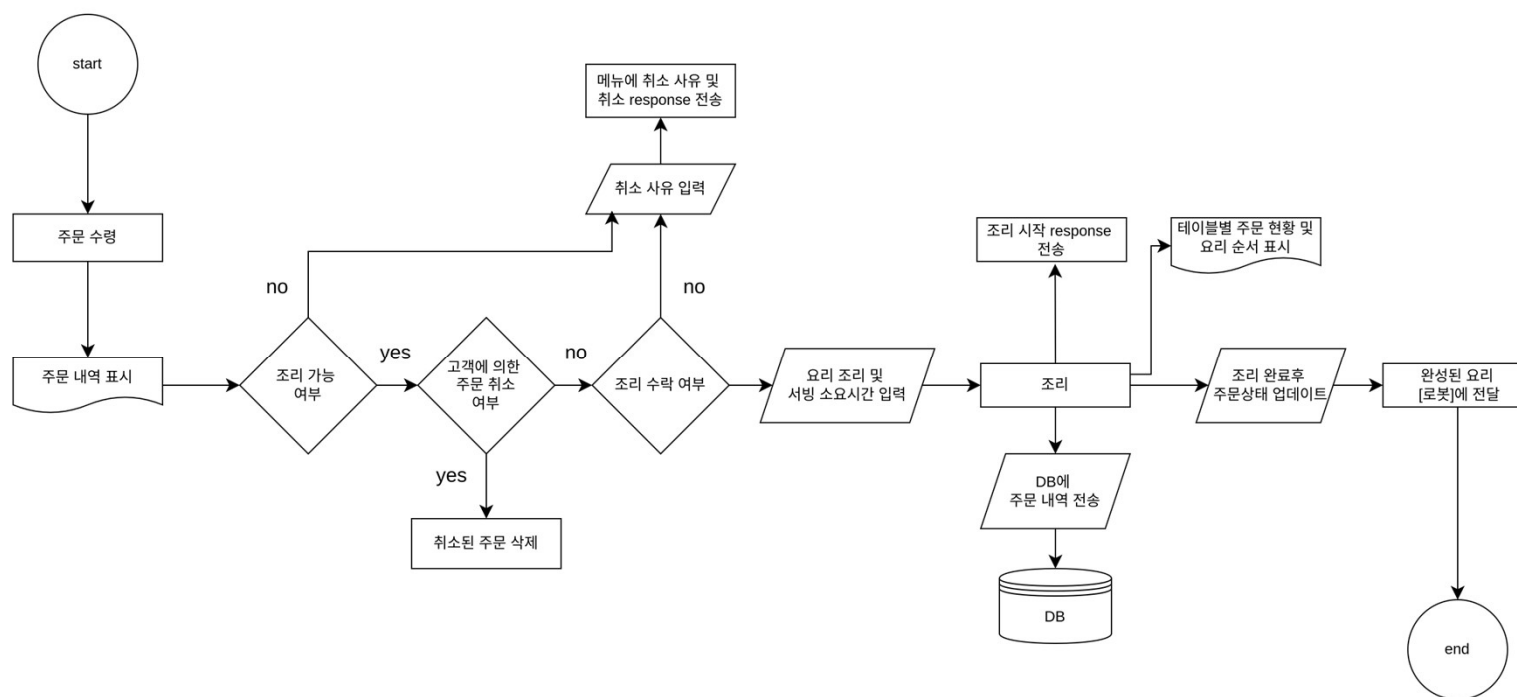


▶ 아키텍처 설계(1) 목표 시스템 구성도



아키텍처 설계(1) Flow Chart

프로세스 흐름 다이어그램(Flowchart)



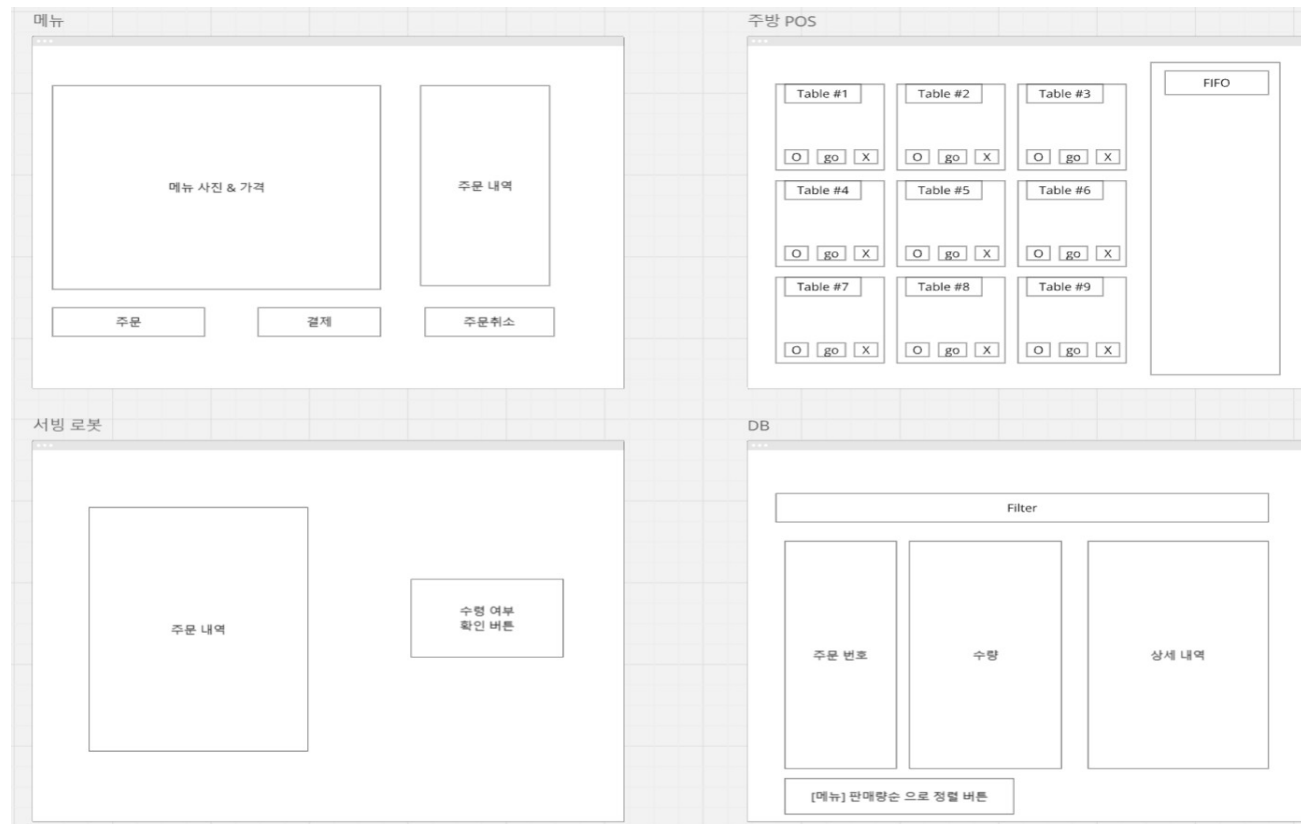


컴포넌트 설명

No.	컴포넌트	역할	주요 기능
1	테이블 오더 시스템	고객 주문 입력 시스템	주문 입력 및 취소
2	주방 디스플레이 시스템	주문 확인 및 알림 제공	주문 알림 및 로봇 제어
3	서빙 로봇	음식 서빙 및 경로 탐색	경로 탐색, 충돌 회피
4	데이터베이스 시스템	주문 및 상태 정보 저장	주문 기록, 통계 제공
5	통계 및 모니터링 시스템	통계 생성 및 모니터링	데이터 분석, 성능 모니터링
6	nav2 Parameter 조정 모듈	로봇 경로 최적화	경로 탐색 조정
7	Logging 시스템	이벤트 기록 및 디버깅	이벤트 기록, 로그 저장
8	QoS (Quality of Service) 모듈	메시지 전송 안정성 보장	메시지 안정화, 품질 보장
9	통신 인터페이스 (토픽, 서비스, 액션)	데이터 전송 및 제어	데이터 전송, 명령 전송

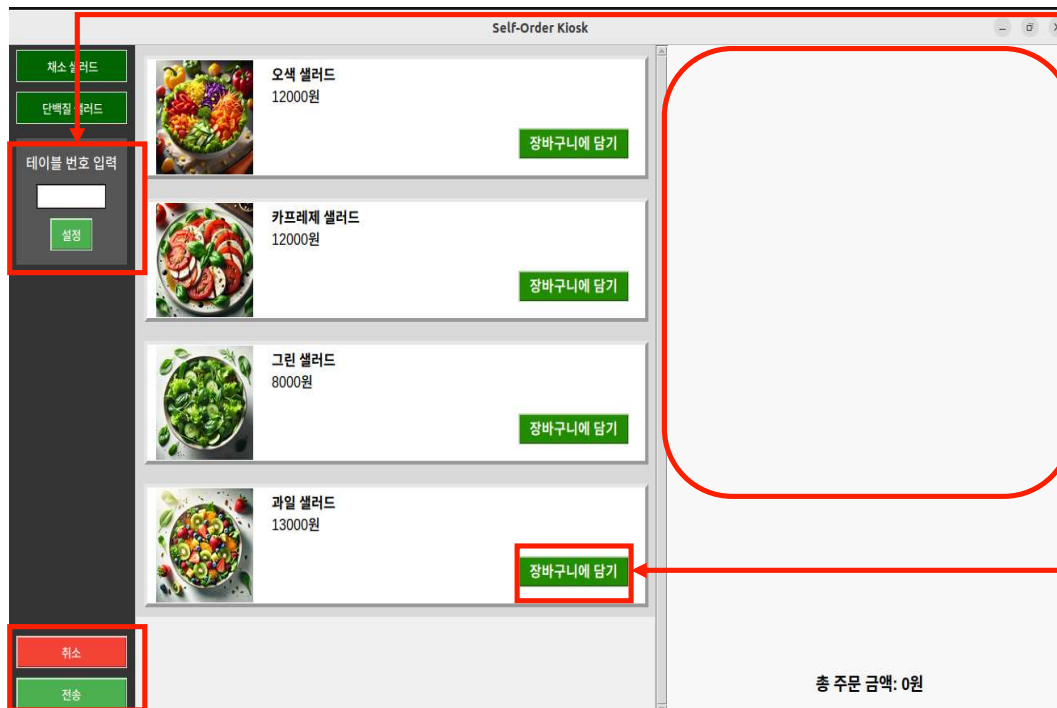


와이어프레임



▶ UI 디자인 – 주문 GUI

■ 주문 GUI



The image shows a UI design for a 'Self-Order Kiosk'. The interface is divided into several sections:

- Left Sidebar:** Contains buttons for '채소 샐러드' (Vegetable Salad), '단백질 샐러드' (Protein Salad), '테이블 번호 입력' (Table Number Input), '설정' (Settings), '취소' (Cancel), and '전송' (Send).
- Main Content Area:** Displays four salad items with images, names, and prices:
 - 오색 샐러드 (Osaek Salad) 12000원
 - 카프레제 샐러드 (Caprese Salad) 12000원
 - 그린 샐러드 (Green Salad) 8000원
 - 과일 샐러드 (Fruit Salad) 13000원
 Each item has a green button labeled '장바구니에 담기' (Add to Cart).
- Bottom Right:** Displays '총 주문 금액: 0원' (Total Order Amount: 0원).

Red arrows and boxes highlight specific UI elements and their corresponding functions in the adjacent text blocks.

테이블 번호 입력

해당 주문에 대한 테이블 번호를 부여함

주문 내역

주문한 메뉴와 수량을 출력함

장바구니에 담기

해당 메뉴의 수량을 입력된 테이블 번호의 주문에 입력함

취소 및 전송

해당 테이블의 주문을 취소하거나 전송하여 주방에 요청할 수 있음

▶ UI 디자인 – 주방 POS GUI

■ 주방 POS GUI



테이블 그리드

테이블에 들어온 주문내역과 고객에게 안내된 조리완료시간을 표출함

테이블 숫자 그리드

해당 테이블로 로봇을 보내는 명령을 하기 위해, 좌표를 전송

주문 큐 창

FIFO 기준으로 테이블별로 입력된 주문을 시간순으로 나열함

오늘의 매출

오늘 입력된 총 주문의 내역과 합산된 매출을 표출하는 창으로 이동



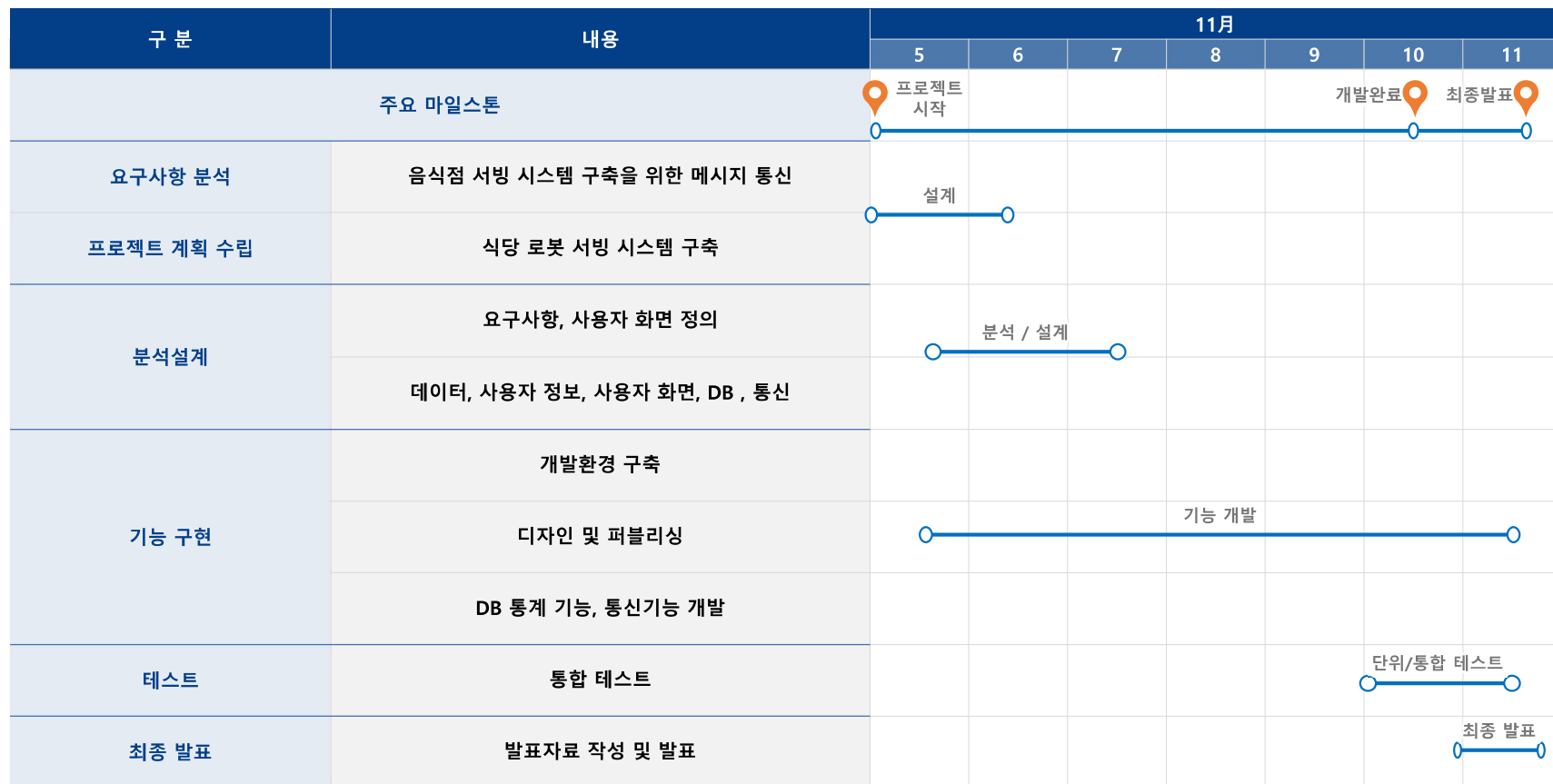
작업 분해 구조(WBS)

진행업무	담당자	결과물
프로젝트 계획 회의	서재원(조장)	요구사항 문서(플로우차트), 작업분해구조도
R&R 작성	서재원(조장)	R&R 문서
일정계획 수립	팀 전원	일정표, Gantt 차트, 리소스 배치계획
식당 Node 구축	서재원(조장)	식당 A 패키지
식당A GUI 및 관련기능 구축	서재원(조장)	식당 A GUI
Database 설계 및 구축	서재원(조장)	SQLite 기반 Database
식당B Node 구축	정의재	식당 B 패키지
식당B GUI 관련기능 구축	정의재	식당 B GUI
식당 Node 전체 취합	서재원(조장), 정의재	식당 패키지 종합 ver.
식당 - 로봇 통신 구축	정의재	식당 통신 모듈
로봇 노드 구축	김근제	로봇 패키지 취합본
로봇 GUI 및 관련 기능 구축	김근제	로봇 GUI
Navigation 구현	김근제	로봇 통신모듈

진행업무	담당자	결과물
메뉴 노드 구축	김차훈	메뉴 패키지
메뉴 GUI 및 관련 기능 구축	김차훈	메뉴 GUI
메뉴-식당 통신 기능 구현	김차훈	통신 모듈
DB - 식당 통신 기능 구현	서재원(조장)	DB 연동 통신모듈
음식점 서빙 시스템	팀 전원	프로토타입
초기 산출물 테스트	팀 전원	프로토타입 내부 평가
프로젝트 중간 회의	팀 전원	추가적인 개발 소요 예상 문서
각 영역별 추가 개발	팀 전원	
각 개발 영역별 내부 검수	팀 전원	
프로젝트 최종 산출물 점검	팀 전원	음식점 서빙 시스템 최종 ver.
프로젝트 발표자료 작성	정의재	발표 자료 최종본
프로젝트 발표 리허설	팀 전원	
최종 발표	서재원(조장)	







간트 차트(Gantt Chart)





리소스 배정표

 서재원 조장		 김근제	
<ul style="list-style-type: none"> 식당 A Node DB구축 (SQLite) 최종 발표 		<ul style="list-style-type: none"> Robot Node GUI 기능 구축 	
 김차훈		 정의재	
<ul style="list-style-type: none"> 메뉴 Node 구축 GUI 기능 구축 		<ul style="list-style-type: none"> 식당 B Node 구축 GUI 기능 구축 발표자료 작성 	

No.	작업 항목	담당자	필요 장비/ 도구
1	공통	팀 전원	회의실, LAPTOP, VSCODE, Ubuntu, ROS2 (※ 공통 장비는 하기 사항들에서 제외)
2	프로젝트 계획 회의	서재원(조장),정의재	SRS 양식, 다이어그램 Software
3	R&R 작성	서재원(조장)	R&R 문서 양식
4	일정계획 수립	팀 전원	일정표, Gantt 차트, 리소스 배치계획 Software
5	Node 구축	팀 전원	GUI Development Tools(GUI Node)
6	GUI 및 관련 기능 구축	팀 전원	GUI Development Tools(GUI Node)
7	Database 설계 및 구축	서재원(조장)	SQLite 기반 Database
8	통신 기능 구현	팀 전원	ROS2 통신 모듈, Turtlebot3
9	Navigation 구현	팀 전원	Turtlebot3, Turtlebot_nav2, Gazebo
10	각 영역별 추가 개발	팀 전원	Turtlebot3, Turtlebot_nav2, Gazebo, GUI Node, SQLite
11	각 개발 영역별 내부 검수	팀 전원	Turtlebot3, Turtlebot_nav2, Gazebo, GUI Node, SQLite
12	프로젝트 최종 산출물 점검	팀 전원	Turtlebot3, Gazebo
13	프로젝트 발표자료 작성	정의재	문서 작업 Software



Code Review (주문)

GUI 초기화, ROS2 노드 설정, 주문 전송 및 서비스 호출 관련 로직을 포함

핵심 코드 부분

```
class KioskGUI:
    def __init__(self, root, ros_node):
        self.root = root
        self.ros_node = ros_node
        self.cart = {}
        self.total_price = 0
        self.setup_sidebar()
        self.setup_menu_display()
        self.setup_cart_display()
        self.check_response_queue()

    def send_order(self):
        if not self.cart:
            messagebox.showwarning("Empty Cart", "장바구니가 비어 있습니다.")
            return
        if not hasattr(self, 'table_number') or self.table_number is None:
            messagebox.showwarning("Table Number Missing", "테이블 번호를 설정하세요.")
            return
        self.ros_node.send_order_request(self.cart, self.table_number)
```

```
class OrderNode(Node):
    def __init__(self, response_queue: Queue):
        super().__init__('order_client')
        self.client = self.create_client(OrdersService, 'order_service')
        self.response_queue = response_queue
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for service...')

    def send_order_request(self, cart_data: dict, table_index: int):
        message = "\n".join([f"({item_name}, {quantity})" for item_name, (price, quantity)
                               in cart_data.items()])
        request = OrdersService.Request()
        request.table_index = table_index
        request.order_detail = message
        future = self.client.call_async(request)
        future.add_done_callback(self.handle_response)

def main():
    rclpy.init()
    response_queue = Queue()
    ros_node = OrderNode(response_queue)
    root = tk.Tk()
    kiosk_gui = KioskGUI(root, ros_node)
    ros_thread = threading.Thread(target=rclpy.spin, args=(ros_node,))
    ros_thread.start()
    root.protocol("WM_DELETE_WINDOW", lambda: (rclpy.shutdown(), root.destroy()))
    root.mainloop()
    ros_thread.join()

if __name__ == "__main__":
    main()
```

상세 설명

- 고객 주문을 관리하는 GUI 애플리케이션으로, Tkinter를 통해 메뉴 표시, 장바구니 관리, 주문 전송을 수행
- Order Node는 ROS2 Order Service를 호출해 주문 데이터를 서버에 비동기로 전송하고 응답을 처리함
- GUI와 ROS 노드는 threading을 사용해 독립적으로 실행되며, 이벤트 큐로 데이터를 교환함



Code Review (주방)

각 클래스의 스레드 실행 및 이벤트 처리 로직

핵심 코드 부분

```
class ActionThread(threading.Thread):
    def __init__(self, node):
        super().__init__(daemon=True)
        self.node = node
        self.action_queue = queue.Queue()
        self.running = True

    def run(self):
        while self.running:
            try:
                event = self.action_queue.get(timeout=0.1)
                if event["type"] == "action_status":
                    self.handle_action_status(event)
            except queue.Empty:
                continue

    def handle_action_status(self, event):
        self.node.get_logger().info(f"[Action] Status: {event['status']}")
```

```
class ServiceThread(threading.Thread):
    def __init__(self, node):
        super().__init__(daemon=True)
        self.node = node
        self.service_queue = queue.Queue()
        self.running = True

    def run(self):
        while self.running:
            try:
                event = self.service_queue.get(timeout=0.1)
                if event["type"] == "order_request":
                    self.handle_service_request(event)
            except queue.Empty:
                continue

    def handle_service_request(self, event):
        self.node.get_logger().info(f"[Service] Processing request for Ta
```

상세 설명

- ROS2 노드의 액션 및 서비스 처리를 위한 멀티스레드 기반 클래스
- ActionThread는 액션 이벤트를 처리하고 로그를 기록함
- ServiceThread는 서비스 요청을 수신하고 처리함
- 각 클래스는 이벤트 큐를 사용해 데이터를 받고, run() 메서드에서 이벤트를 반복 처리
- add_event() 메서드를 통해 외부에서 이벤트를 추가 가능



Code Review (스레드)

GUI 초기화, ROS2 노드 설정, 주문 전송 및 서비스 호출 관련 로직을 포함

핵심 코드 부분

```
class KioskGUI:
    def __init__(self, root, ros_node):
        self.root = root
        self.ros_node = ros_node
        self.cart = {}
        self.total_price = 0
        self.setup_sidebar()
        self.setup_menu_display()
        self.setup_cart_display()
        self.check_response_queue()

    def send_order(self):
        if not self.cart:
            messagebox.showwarning("Empty Cart", "장바구니가 비어 있습니다.")
            return
        if not hasattr(self, 'table_number') or self.table_number is None:
            messagebox.showwarning("Table Number Missing", "테이블 번호를 설정하세요.")
            return
        self.ros_node.send_order_request(self.cart, self.table_number)
```

```
class OrderNode(Node):
    def __init__(self, response_queue: Queue):
        super().__init__('order_client')
        self.client = self.create_client(OrdersService, 'order_service')
        self.response_queue = response_queue
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for service...')

    def send_order_request(self, cart_data: dict, table_index: int):
        message = "\n".join([f"({item_name}), {quantity}" for item_name, (price, quantity)
                               in cart_data.items()])
        request = OrdersService.Request()
        request.table_index = table_index
        request.order_detail = message
        future = self.client.call_async(request)
        future.add_done_callback(self.handle_response)

def main():
    rclpy.init()
    response_queue = Queue()
    ros_node = OrderNode(response_queue)
    root = tk.Tk()
    kiosk_gui = KioskGUI(root, ros_node)
    ros_thread = threading.Thread(target=rclpy.spin, args=(ros_node,))
    ros_thread.start()
    root.protocol("WM_DELETE_WINDOW", lambda: (rclpy.shutdown(), root.destroy()))
    root.mainloop()
    ros_thread.join()

if __name__ == "__main__":
    main()
```

상세 설명

- Tkinter를 사용해 메뉴 항목 표시, 장바구니 관리, 테이블 번호 입력, 주문 전송 등의 기능을 제공합니다.
- ROS2 Order Service를 호출하는 Order Node와 연결되어 주문 데이터를 비동기식으로 서버에 전송하며, 응답은 큐를 통해 GUI로 반환됩니다.
- ROS 노드는 rclpy와 threading을 사용해 GUI와 독립적으로 실행됩니다.



Code Review (주방 통신)

ROS2 노드에서 주문 요청 처리 (Service), 로봇의 이동 (Action Client)

핵심 코드 부분

```
def handle_order_request(self, request, response):
    """주문 요청 처리 (동기 처리)"""
    if self.is_processing_request:
        # 이미 요청 처리 중인 경우 로그 출력
        self.get_logger().warn("A request is already being processed. Ignoring new request.")
        response.success = False
        response.message = "Server busy. Try again later."
        return response

    # 요청 처리 시작
    self.is_processing_request = True
    self.get_logger().info(f"Processing request for Table {request.table_index}")

    # 이벤트 큐에 요청 추가 (GUI 처리를 위해)
    self.event_queue.put({
        "type": "order_request",
        "table_index": request.table_index,
        "message": request.order_detail,
        "response": response,
    })
```

```
def navigate_to_pose_send_goal(self, goal_index):
    goal_msg = NavigateToPose.Goal()
    goal_msg.pose.header.frame_id = "map"
    goal_msg.pose.pose.position.x = self.goal_poses[goal_index][0]
    goal_msg.pose.pose.position.y = self.goal_poses[goal_index][1]
    goal_msg.pose.pose.orientation.w = 1.0
    send_goal_future = self.navigate_to_pose_action_client.send_goal_async(goal_msg)
    send_goal_future.add_done_callback(self.on_goal_reached)
```

상세 설명

주문 처리: KitNode의 `handle_order_request` 함수는 주문 요청을 받아 수락 또는 거절하고, 결과를 `response`로 반환하여 외부 시스템에 전달

로봇 이동: `navigate_to_goals()`와 `navigate_to_pose_send_goal()` 메서드는 미리 정의된 목표 지점(`goal_poses`)으로 로봇을 이동시키기 위해 `NavigateToPose` 액션을 사용



Code Review (주문 통신)

OrderNode 클래스(ROS2 클라이언트), 주문 요청 전송

핵심 코드 부분

```
class OrderNode(Node):
    def __init__(self, response_queue: Queue):
        super().__init__('order_client')
        self.client = self.create_client(OrderService, 'order_service')
        self.response_queue = response_queue
        while not self.client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for service...')

    def send_order_request(self, cart_data: dict, table_index: int):
        """서버로 주문 요청을 보냅니다."""
        # 주문 상세 메시지 생성
        message = "\n".join([f"{{item_name}}, {{quantity}}" for item_name, (price, quantity) in cart_data.items()])

        # OrderService 요청 생성
        request = OrderService.Request()
        request.table_index = table_index # GUI에서 전달된 테이블 번호 사용
        request.order_detail = message

        # 비동기 서비스 호출
        self.get_logger().info(f"Sending order request: Table {table_index}, Order: {message}")
        future = self.client.call_async(request)
        future.add_done_callback(self.handle_response)
```

상세 설명

OrderNode 클래스는 ROS2의 Node를 상속받아 클라이언트를 생성하고 'order_service'에 연결

wait_for_service()로 서비스가 활성화될 때까지 대기하며, 1초마다 로그 메시지를 출력

cart_data에서 주문 항목을 추출하고, OrderService.Request() 객체에 설정하여 서버로 전송합니다.

call_async(request)로 비동기적으로 주문을 서버에 전송하며, GUI가 차단되지 않도록 함

서버 응답은 비동기적으로 처리되며, 응답이 오면 handle_response가 호출되어 결과를 처리



Code Review (스레드 통신)

ActionThread 클래스, ServiceThread 클래스

핵심 코드 부분

```
class ActionThread(threading.Thread):
    def __init__(self, node):
        super().__init__(daemon=True)
        self.node = node
        self.action_queue = queue.Queue()
        self.running = True

    def run(self):
        while self.running:
            try:
                event = self.action_queue.get(timeout=0.1)
                if event["type"] == "action_status":
                    self.handle_action_status(event)
            except queue.Empty:
                continue

    def handle_action_status(self, event):
        self.node.get_logger().info(f"[Action] Status: {event['status']} for goal {event['goal_index']}")

    def add_event(self, event):
        """큐에 이벤트 추가."""
        self.action_queue.put(event)

    def stop(self):
        self.running = False
```

```
class ServiceThread(threading.Thread):
    def __init__(self, node):
        super().__init__(daemon=True)
        self.node = node
        self.service_queue = queue.Queue()
        self.running = True
```

상세 설명

큐 기반 통신 : 각 스레드는 자신의 큐(action_queue, service_queue)를 통해 외부 이벤트를 비동기적으로 처리

이벤트 큐 처리 : 외부에서 add_event 메서드로 이벤트를 추가하면, 스레드는 이를 queue.get()으로 처리

비동기적 이벤트 처리: 큐에 이벤트가 들어오면 스레드가 처리하고, 이벤트 종류 적합한 작업 수행

스레드 종료 : stop 메서드를 사용하여 스레드를 안전하게 종료할 수 있음

▶ databases 설계

full_order_history.db		current_table_orders.db	
full_order_history.db		Filter 2 tables...	
TABLES		Rows: 12	
full_order_history	sqlite_sequence	order_id	table_id
quantity	order_time	menu_id	
1	2024-11-10 13:47:47	오색 쥬얼리드	1
1	2024-11-10 13:49:55	오색 쥬얼리드	2
1	2024-11-10 13:49:55	카프레제 쥬얼리드	3
1	2024-11-10 13:53:07	오색 쥬얼리드	4
1	2024-11-10 13:53:07	카프레제 쥬얼리드	5
1	2024-11-10 14:27:52	과일 쥬얼리드	6
1	2024-11-10 14:27:52	그린 쥬얼리드	7
1	2024-11-10 14:27:52	새우 쥬얼리드	8
1	2024-11-10 14:27:52	스테이크 쥬얼리드	9
1	2024-11-10 14:27:52	치킨 쥬얼리드	10
1	2024-11-10 14:27:52	카프레제 쥬얼리드	11
1	2024-11-10 14:27:52	훈제연어 쥬얼리드	12

The screenshot displays a database interface with the following components:

- Database:** full_order_history.db
- Table:** current_table_orders.db
- Filter:** Filter 2 tables...
- Rows:** 35
- Table Structure:**
 - order_id (Primary Key)
 - table_id
 - menu_id
 - quantity
 - order_time
- Data Rows:**

order_id	table_id	menu_id	quantity	order_time
1	6	오색 샐러드	1	2024-11-10 13:54:40
2	7	오색 샐러드	1	2024-11-10 14:00:41
3	8	카프레제 샐러드	1	2024-11-10 14:00:41
4	9	오색 샐러드	1	2024-11-10 14:13:48
5	17	그린 샐러드	1	2024-11-10 14:48:26
6	18	오색 샐러드	1	2024-11-11 00:38:17
7	19	오색 샐러드	1	2024-11-11 00:38:37
8	20	카프레제 샐러드	1	2024-11-11 00:38:37
9	21	그린 샐러드	1	2024-11-11 00:38:37
10	22	과일 샐러드	1	2024-11-11 00:38:37
11	23	카프레제 샐러드	1	2024-11-11 00:39:23
12	24	오색 샐러드	1	2024-11-11 00:46:13
13	25	카프레제 샐러드	1	2024-11-11 00:46:13
14	26	그린 샐러드	1	2024-11-11 00:46:14

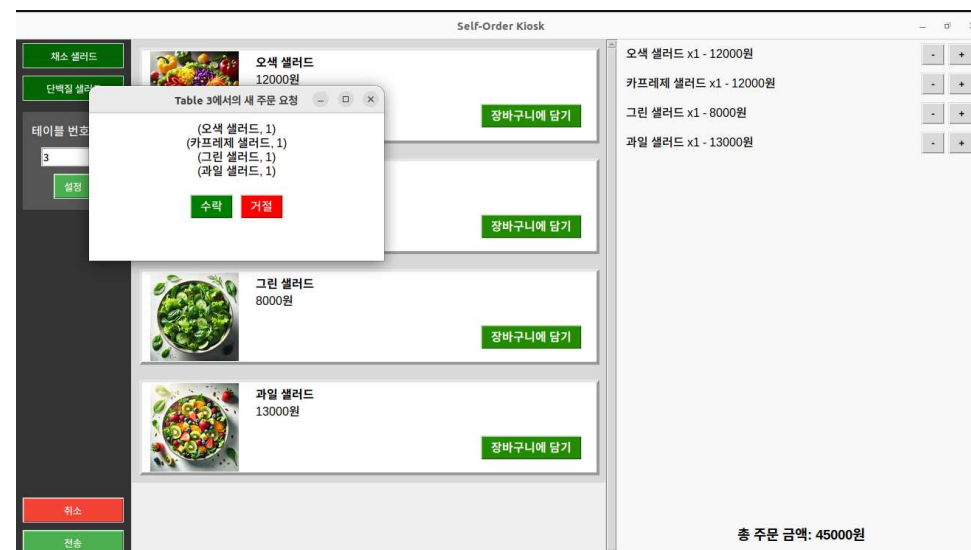
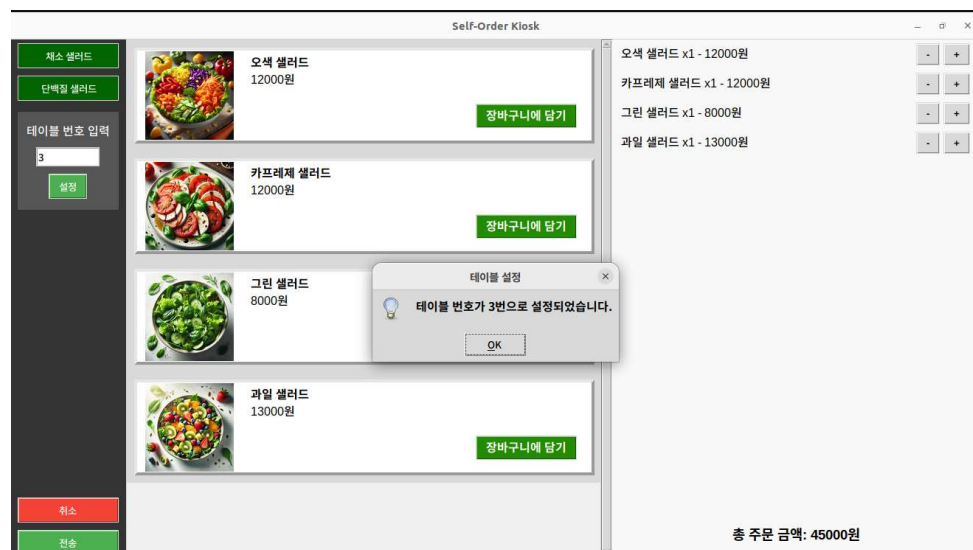
상세 설명

- 1. full_order_history.db
 - 테이블 : full_order_history
 - 전체 주문 목록
- 2. current_table_orders.db
 - 테이블:current_table_orders
 - 현재 테이블의 주문 상태 관리



최종 결과(1)

메뉴 / 수량 설정 → 테이블 번호 입력 → 전송





최종 결과(2)

해당 테이블 그리드 클릭 → 운반 확인 팝업 → 클릭 → 로봇에 해당 테이블 좌표로 명령 하달

테이블 및 좌석 현황 GUI

테이블 1번 주문 없음	테이블 2번 주문 없음	3번 테이블 과일 샐러드: 1개 그린 샐러드: 1개 오색 샐러드: 1개 카페프레제 샐러드: 1개
테이블 4번 주문 없음	테이블 5번 주문 없음	테이블 6번 주문 없음
테이블 7번 주문 없음	테이블 8번 주문 없음	테이블 9번 주문 없음

주문 큐

테이블 3 - (오색 샐러드, 1)*(카페프레제 샐러드, 1)

1	2	3
4	5	6
7	8	9

오늘의 매출

테이블 및 좌석 현황 GUI

테이블 1번 주문 없음	테이블 2번 주문 없음	3번 테이블 과일 샐러드: 1개 그린 샐러드: 1개 오색 샐러드: 1개 카페프레제 샐러드: 1개
테이블 4번 주문 없음	테이블 5번 주문 없음	테이블 6번 주문 없음
테이블 7번 주문 없음	테이블 8번 주문 없음	테이블 9번 주문 없음

주문 큐

테이블 3 - (오색 샐러드, 1)*(카페프레제 샐러드, 1)

1	2	3
4	5	6
7	8	9

오늘의 매출

운반 확인

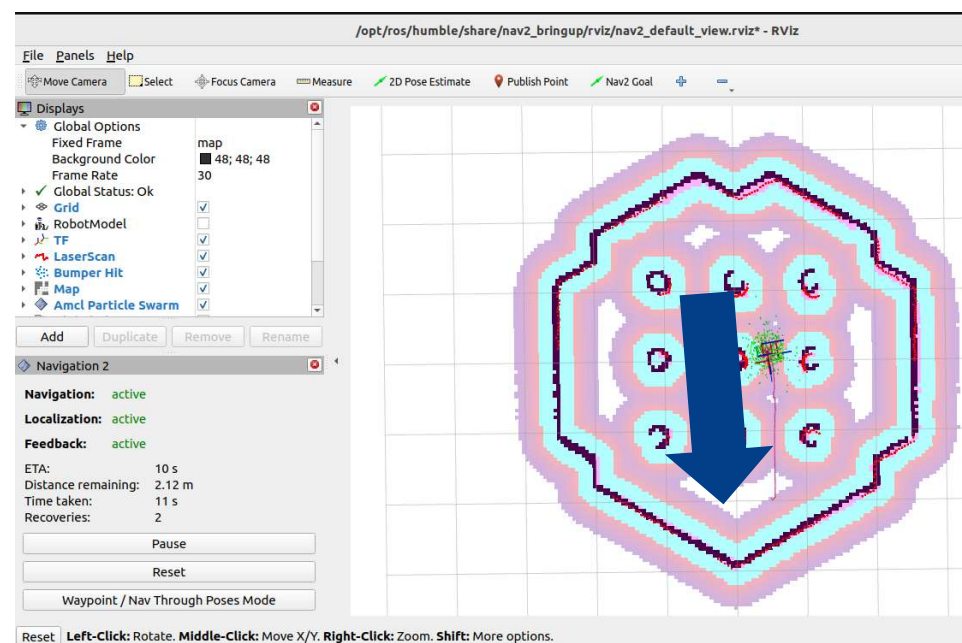
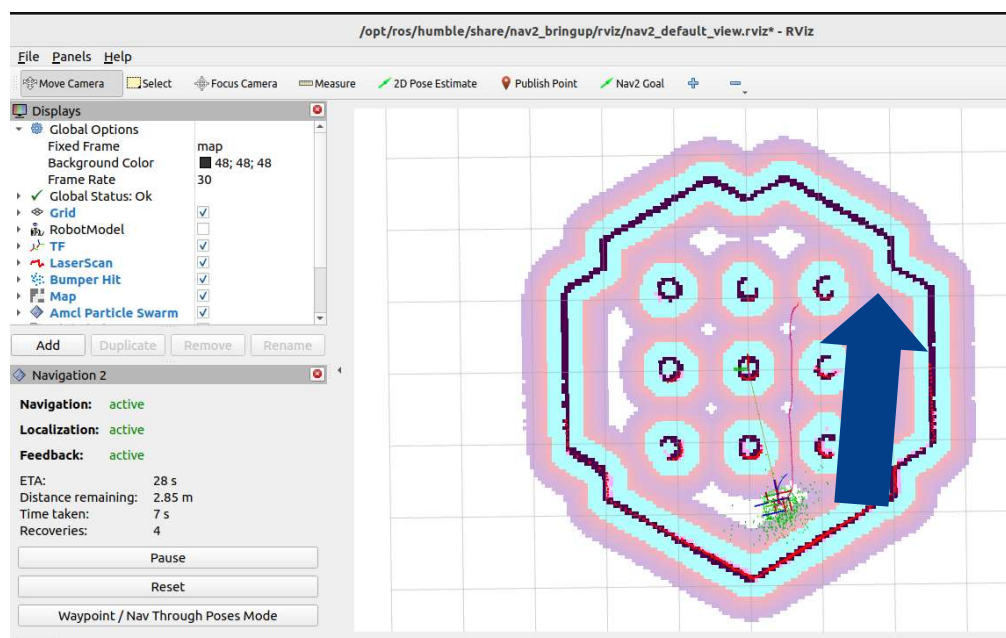
3번에 운반하겠습니까?

OK Cancel



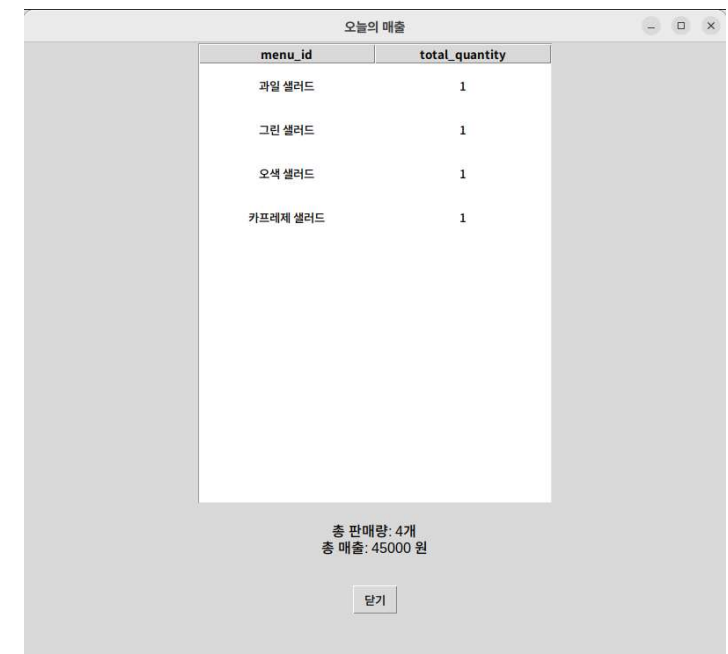
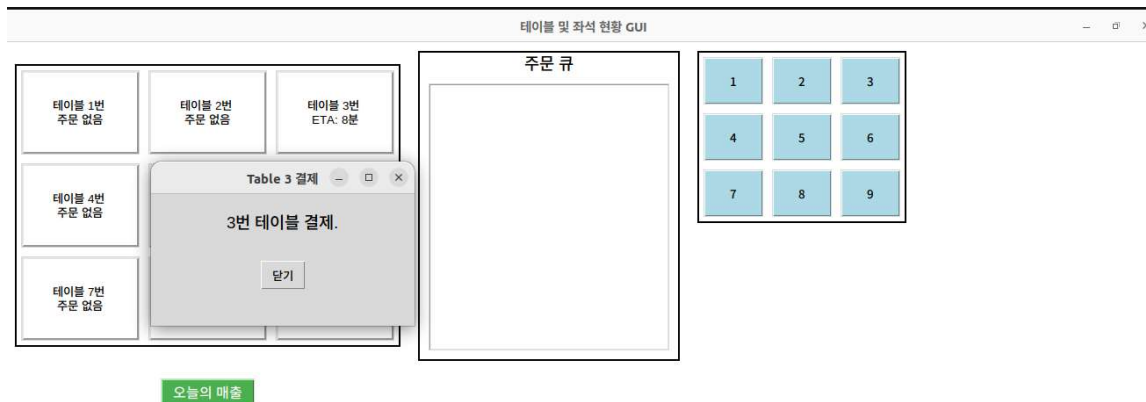
최종 결과(3)

명령 하달된 로봇 이동 → 복귀



▶ 최종 결과 - 부가기능

1. 결제 완료시에 해당 테이블의 결제 유무를 팝업으로 표출
2. 오늘의 매출을 클릭 시 당일의 총 주문 수량과 매출을 표출





최종 결과 - 부가기능

1. 취소 버튼 클릭 시, 취소의 사유를 선택할 수 있는 팝업창이 호출되며, 이를 주방에 전달함

