# Coordination of a Multi-Agent System for Emergency Response

Team 05

January 11, 2025

## Overview

- Task 1 focused on environment and agent design
- Task 2 explored coordination mechanisms
- Now, we integrate these into a practical implementation
- Implementation utilizes **CrewAI** and **Ollama**
- In this presentation, we will cover:
  - Agents and their tasks
  - Crews module and data models
  - Database and tools
  - Scripts
  - Testing
  - Integration in `main.py` and data folder

## Agents and Tasks

- For each crew, we define agents and tasks using .yaml files.
- Each crew is represented as a Python class, with agents and tasks defined as methods within the class.
- Agents are instantiated with configuration settings, and tasks are created by linking them with specific tools and data models.

### Example Code: Medical Services Crew

```
hospital_coordinator:
  role: Hospital Coordinator
  goal: >
    ...
  backstory: >
    ...
  allow_delegation: false
  verbose: true
  llm: ollama/llama3.1
  temperature: 0.4
  max_tokens: 800
```

```
rank_hospitals:
  description: >
    ...
  expected_output: >
    ...
  agent: hospital_coordinator
  context:
    - fetch_hospital_information
```

# Crews Module

# Data Models

# Scripts

### Map Analysis

```
places = ["Barcelona, Spain",
    "Seville, Spain",
    "Salamanca, Spain",
    "Tossa de Mar, Spain",
    "Lloret de Mar, Spain",
    "New York, NY, USA"]
...
columns = ["n",
    "m",
    "k_avg",
    "edge_length_total",
    "edge_length_avg",
    "streets_per_node_avg",
    "intersection_count",
    "edge_density_km",
    "street_density_km",
    "clean_intersection_density_km"]
...
df = pd.DataFrame(mapp_stats, index=
    places)
```

### Database Initialization

populate_incident_table()

```
CREATE TABLE incidents (
    summary TEXT,
    timestamp TEXT,
    fire_severity TEXT,
    fire_type TEXT,
    location_x REAL,
    location_y REAL)
```

populate_hospital_table()

```
CREATE TABLE hospitals (
    hospital_id TEXT,
    location_x REAL,
    location_y REAL,
    available_beds INTEGER,
    available_ambulances INTEGER,
    available_paramedics INTEGER)
```

# GPS Tool

```python
from crewai.tools import BaseTool
import osmnx as ox

class RouteDistanceTool(BaseTool):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.city_map = ox.load_graphml(GRAPHML_FILENAME)

    def _find_distance(self, x_origin, y_origin, x_destination, y_destination):
        origin_node = ox.distance.nearest_nodes(self.city_map, x_origin, y_origin
            )
        destination_node = ox.distance.nearest_nodes(self.city_map, x_destination
            , y_destination)
        route = ox.shortest_path(self.city_map, origin_node, destination_node,
            weight="travel_time")
        edge_lengths = ox.routing.route_to_gdf(self.city_map, route)["length"]

        return round(sum(edge_lengths)) / 1000
```
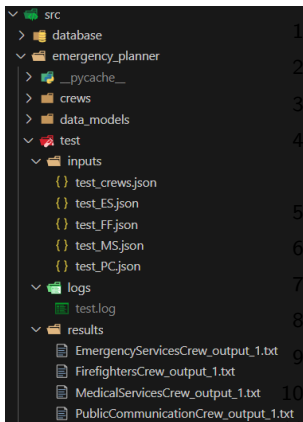
# Database Tools

| Tool | Input | Output |
|------|-------|--------|
| Hospital Reader | None | List of hospitals with their available resources |
| Hospital Updater | Hospital ID<br>Beds Reserved<br>Ambulances Dispatched<br>Paramedics Deployed | None |
| Incident Retrieval | Location X<br>Location Y<br>Fire Severity<br>Fire Type<br>Summary | Related cases |

## emergency_planner/test.py

```python
def process_crew_test(crew_name: str, crew_inputs: Dict[str, Any], test_index:
     int) -> None:
    ...
    crew = instantiate_crew(crew_name)
    logger.info("Agents loaded")
    for agent in crew.crew().agents:
        logger.info(f"Role: {agent.role}")
    ...
    for task in crew.crew().tasks:
        result = task.execute_sync(agent=task.agent, context=task.context, tools=
            task.tools)
    ...
    final_result = crew.crew().kickoff(inputs=crew_inputs)
    ...
def main() -> None:
    ...
    test_cases = load_test_cases(JSON_FILE)
    for i, test_case in enumerate(test_cases, start=1):
        ...
        # Input model validation
        ...
        process_crew_test(crew_name, crew_inputs, i)
    ...
```

# emergency_planner/test/

Listing 1: Input Example

```
src
  database
  emergency_planner
    __pycache__
    crews
    data_models
    test
      inputs
        test_crews.json
        test_ES.json
        test_FF.json
        test_MS.json
        test_PC.json
      logs
        test.log
      results
        EmergencyServicesCrew_output_1.txt
        FirefightersCrew_output_1.txt
        MedicalServicesCrew_output_1.txt
        PublicCommunicationCrew_output_1.txt
```

```
1  {
2  "test_cases": [
3      {
4          "crew_to_test": "
              EmergencyServicesCrew",
5          "crew_inputs": {
6              "transcript":
7                  "A fire of electrical..."
8          }
9      }
10 ]
11 }
```

# emergency_planner/main.py

```python
class EmergencyPlannerFlow(Flow[EmergencyPlannerState]):
    @start()
    def get_call_transcript(self): ...

    @listen(get_call_transcript)
    def emergency_services(self): ...

    @listen(emergency_services)
    def firefighters(self): ...

    @listen(emergency_services)
    def medical_services(self):
        if not self.state.call_assessment.medical_services_required:
            return
        ...

    @listen(or_(and_(firefighters, medical_services), "
        retry_public_communication"))
    def public_communication(self): ...

    @router(public_communication)
    def check_approval(self): ...
```

# data/

```
@start()
def get_call_transcript(self):
    with open(EMERGENCY_CALL_TRANSCRIPTS_FILENAME, "r"
        ) as f:
        self.state.call_transcript = f.readlines()[
            TRANSCRIPT_INDEX]
```

```
@listen("save full emergency report")
def save_full_emergency_report(self):
    full_emergency_report = f"""
# Emergency Report

## Call Transcript
{self.state.call_transcript}
...
"""
with open(EMERGENCY_REPORT_FILENAME, "w") as f:
    f.write(full_emergency_report)
```

- ∨ data
  - ∨ inputs
    - ◈ .gitkeep
    - ≡ call_transcripts.txt
    - ≡ lloretDeMar.graphml
  - ∨ outputs
    - ◈ .gitkeep
    - ⬇ emergency_report_2.md
    - ⬇ emergency_report.md
    - ≡ logs1.txt
    - ≡ logs2.txt

# Thank you!

Questions?