```
import concurrent.futures
from typing import List, Union
from loguru import logger
from pydantic import BaseModel
from swarms.structs.agent import Agent
from swarms.schemas.agent_step_schemas import ManySteps


class AgentRowMetadata(BaseModel):
    row_index: int
    agent_runs: List[ManySteps]


class AgentMatrixMetadata(BaseModel):
    matrix_runs: List[AgentRowMetadata]


class AgentMatrix:
    def __init__(
        self, agents: Union[List["Agent"], List[List["Agent"]]]
    ):
        """
        Initializes the matrix with the provided list of agents or list of lists of agents.

        Args:
            agents (List[Agent] or List[List[Agent]]): A list of agents or a list of lists of agents (matrix).
```

```
        """
        if isinstance(agents[0], list):
            self.agents_matrix: List[List["Agent"]] = (
                agents  # List of lists (matrix)
            )
            self.rows: int = len(agents)
            self.cols: int = len(agents[0]) if agents else 0
        else:
            self.agents_matrix: List[List["Agent"]] = [
                agents
            ]  # Single row of agents (1D list)
            self.rows: int = 1
            self.cols: int = len(agents)

        # Store metadata for all runs
        self.matrix_metadata = AgentMatrixMetadata(matrix_runs=[])
        logger.info(
            f"AgentMatrix initialized with {self.rows} rows and {self.cols} columns of agents."
        )

    def execute_in_order(self, query: str) -> None:
        """Executes the agents in row-major order."""
        logger.info(
            f"Executing all agents in row-major order with query: {query}"
        )
        for i, row in enumerate(self.agents_matrix):
```

```
        row_metadata = AgentRowMetadata(
            row_index=i, agent_runs=[]
        )
        for j, agent in enumerate(row):
            logger.info(f"Executing Agent [{i}][{j}]")
            out = agent.run(query)
            logger.info(f"Output from Agent [{i}][{j}]: {out}")

            agent_metadata = agent.agent_output
            row_metadata.agent_runs.append(agent_metadata)
        self.matrix_metadata.matrix_runs.append(row_metadata)


def execute_by_row(
    self, row_index: int, query: str, sequential: bool = True
) -> None:
    """
    Executes all agents in a specific row, either sequentially or concurrently.

    Args:
        row_index (int): The index of the row to execute.
        query (str): The query to run.
        sequential (bool): Whether to execute agents sequentially (True) or concurrently (False).
    """
    if not (0 <= row_index < self.rows):
        logger.error(f"Invalid row index: {row_index}")
        return
```

```
    logger.info(
        f"Executing row {row_index} with query: {query}. Sequential: {sequential}"
    )
    row_metadata = AgentRowMetadata(
        row_index=row_index, agent_runs=[]
    )

    if sequential:
        self._execute_row_sequentially(
            row_index, query, row_metadata
        )
    else:
        self._execute_row_concurrently(
            row_index, query, row_metadata
        )

    self.matrix_metadata.matrix_runs.append(row_metadata)

def _execute_row_sequentially(
    self,
    row_index: int,
    query: str,
    row_metadata: AgentRowMetadata,
) -> None:
    """Executes agents in a row sequentially, passing output from one agent to the next."""
```

```
    logger.info(
        f"Executing agents in row {row_index} sequentially."
    )
    current_input = query
    for j, agent in enumerate(self.agents_matrix[row_index]):
        logger.info(
            f"Executing Agent [{row_index}][{j}] sequentially with input: {current_input}"
        )
        current_output = agent.run(current_input)
        agent_metadata = agent.agent_output
        logger.info(
            f"Output from Agent [{row_index}][{j}]: {current_output}"
        )
        row_metadata.agent_runs.append(agent_metadata)
        current_input = current_output


def _execute_row_concurrently(
    self,
    row_index: int,
    query: str,
    row_metadata: AgentRowMetadata,
) -> None:
    """Executes agents in a row concurrently."""
    logger.info(
        f"Executing agents in row {row_index} concurrently."
    )
```

```
def agent_task(agent, query):
    return agent.run(query)


with concurrent.futures.ThreadPoolExecutor() as executor:
    future_to_agent = {
        executor.submit(agent_task, agent, query): agent
        for agent in self.agents_matrix[row_index]
    }
    for future in concurrent.futures.as_completed(
        future_to_agent
    ):
        agent = future_to_agent[future]
        try:
            output = future.result()
            logger.info(
                f"Output from concurrent agent: {output}"
            )


            # Capture metadata
            agent_metadata = agent.agent_output
            row_metadata.agent_runs.append(agent_metadata)


        except Exception as exc:
            logger.error(
                f"Agent generated an exception: {exc}"
```

```
            )


def execute_by_column(self, col_index: int, query: str) -> None:
    """Executes all agents in a specific column."""
    if not (0 <= col_index < self.cols):
        logger.error(f"Invalid column index: {col_index}")
        return

    logger.info(
        f"Executing column {col_index} with query: {query}"
    )
    for i in range(self.rows):
        logger.info(f"Executing Agent [{i}][{col_index}]")
        out = self.agents_matrix[i][col_index].run(query)
        logger.info(
            f"Output from Agent [{i}][{col_index}]: {out}"
        )

        # Capture metadata for the column run
        row_metadata = AgentRowMetadata(
            row_index=i, agent_runs=[]
        )
        agent_metadata = self.agents_matrix[i][
            col_index
        ].agent_output
        row_metadata.agent_runs.append(agent_metadata)
```

```
            self.matrix_metadata.matrix_runs.append(row_metadata)

    def export_metadata(self) -> str:
        """Exports the metadata to a JSON format."""
        logger.info("Exporting metadata to JSON.")
        return self.matrix_metadata.json(indent=4)


# Example usage with pre-created agents

# # Assuming you have pre-created agents, here's an example:
# # agent_1, agent_2, ..., agent_n are instances of the `Agent` class

# agents_row_1 = [agent_1, agent_2, agent_3]
# agents_row_2 = [agent_4, agent_5, agent_6]
# agents_row_3 = [agent_7, agent_8, agent_9]

# # Matrix of agents (list of lists)
# agents_matrix = [agents_row_1, agents_row_2, agents_row_3]

# # Initialize the AgentMatrix with the list of lists
# agent_matrix = AgentMatrix(agents_matrix)

# # Execute all agents in row 1 sequentially (output of one agent passed to the next)
# agent_matrix.execute_by_row(1, "What is the process for getting a ROTH IRA started?", sequential=True)
```

Execute all agents in row 1 concurrently (all agents run independently)

agent_matrix.execute_by_row(1, "What is the process for getting a ROTH IRA started?", sequential=False)

Export and print the run metadata in JSON format

metadata_json = agent_matrix.export_metadata()

print(metadata_json)