**MAS PROJECTS**

The topic will be chosen randomly from the 7 topics below.
The project will be developed in teams of up to 3 students using the ActressMas framework. The responsibilities of each team member must be clearly outlined.
The program will be accompanied by documentation of at least five A4 pages written in Times New Roman size 12 font, single line spacing (Line spacing - Single). No blank lines should exist between sections of the document. The documentation shall contain:

• Full names of the authors
• A general description of the project (half a page)
• Usage of the program
• The internal architecture of the program, including significant parts of the source code, commented using Courier New font or equivalent, size 10, single line spacing
• Screenshots demonstrating the application in execution

**Topic 1**

Implement a multi-agent system for the purchase of plane tickets for a destination provided by the user. The system will contain multiple specialized agents, representing various airlines. They will know the list of the flights (departure time, arrival time, and cities), and the cost of the flight.
The user has a personal assistant agent who shall provide the most convenient offers based on travel time and price.

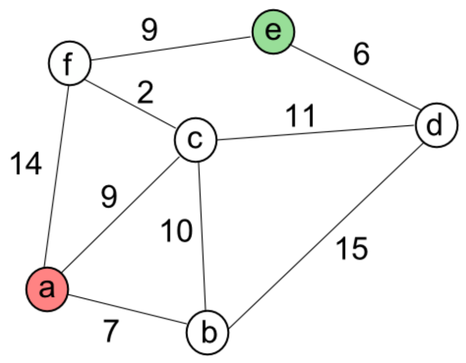
Search items:
• Destination: the city (airport)
• Dates: these can be flexible. For example, 25.08.2011-30.08.2011 (+/- 3 days)

Routes may involve stopovers or changes of aircraft, in which the waiting time in airport shall also be considered.

It could be that the best route choice may involve both a change in aircraft and the operating company. Therefore the personal assistant has to calculate the best route from the starting city to the destination accounting for the fact that multiple flights may be more cost-effective than a single flight. The personal assistant shall search for the best routes according to price firstly, and travel time secondly (i.e. for routes with similar prices within +/-10%, the fastest one is the preferred one).

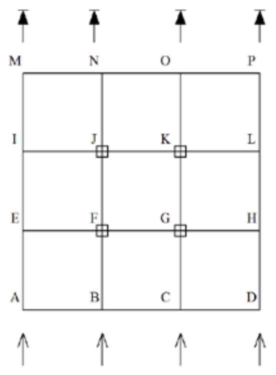
The personal assistant agent will talk to the airline agents and will assemble a list of travel options for the user. The user should receive up to four of the most convenient travel options, in acending order of their cost.

An example of travel routes is provided in the graph below. Each airline agent will have a list of available flights between specific pairs of nodes. Two distinct airlines may offer flights between the same two nodes, but at different costs. Based on the possible routes between the departure and destination nodes, the personal assistant agent should query the airlines and search for the best routes in terms of cost and travel time.



**Topic 2**

Implement a multi-agent system which manages traffic on a street layout with multiple intersections. The streets and intersections are configured as in the following image:



Each car is represented by an agent. Cars are incoming from entry points A, B, C, or D and each wants to get to one of the exits M, N, O or P.
E … L are intersections; in each intersection there is a traffic light. Traffic lights should be represented by agents as well. Each traffic light will have its own switching time interval. The switching times should be initialized with equal values at first.
At each intersection, the car agents decide which way to go in a *greedy* manner, i.e they will choose a direction which:

- is either left, right or forward
- is selected such that the total distance from entry point to exit is equal to the Manhattan distance between the entry point and the exit
    - o for example, if a car wants to get from B to O, it should stay within the rectangle B-C-O-N
- has no obstacles / the least amount of obstacles

Possible obstacles are:

- **binary**: a red street light for that direction
- **continuous**: the amount of traffic in the desired direction (the number of cars in the corresponding road segment)

The cost of choosing a road segment $k$ at a moment in time $t$ can be computed as follows:

$$c_k(t) = \frac{l_k}{v_k(t)}$$

where $l_k$ is the length of the road segment and $v_k(t)$ is the speed the car will be able to use on that road segment:

$$v_k(t) = \overline{v_k} \cdot \left[1 - \frac{q_k(t)}{y_k(t)}\right]$$

where:
$\bar{v}_k$ is the maximum speed attainable on road segment $k$
$q_k(t)$ is the number of vehicles on road segment $k$ at time $t$ (the vehicle flow on that segment)
$y_k(t)$ is the maximum number of vehicles that can be present on road segment $k$ (i.e. the capacity of road segment $k$)

The system should be controllable by multiple user-defined parameters, which should be read from a simple config file. The control parameters are:

- the intelligence of traffic lights
    - o non-intelligent traffic light: the switching time is constant, no matter the traffic
    - o intelligent: the switching time depends on the traffic
        - ▪ in this case, another parameter whould be the level of intelligence:
            - • level 1: the street light has knowledge of the amount of traffic in its adjacent street segments
            - • level 2: the street light has knowledge of the amount of traffic as far as two street segments away from its position
            - • level 3: the street light has full knowledge of the amount of traffic, on all street segments
- the rate with which cars are generated at each entry point A, B, C, D. Rate = cars/second, cars/minute etc.
- whether cars will prioritize street lights or the amount of traffic, meaning:
    - o whether a car at an intersection will choose a street segment with a green light or
    - o whether a car will wait at a red light if there is lower traffic on the following street segment

**A visual representation of the street segments, street lights and cars is required in order to better analyze the behavior of the system.**
**Note**: you do not have to implement the intelligent behavior of the street lights, but you have to provide the functionality for the street light to receive the required amount of knowledge.

**Topic 3**

Implement a multiagent system that simulates an emergency evacuation scenario. The agent environment is a 2D grid symbolizing the floorplan of the building that needs to be evacuated, while one or several cells serve as the exits. In the grid there are multiple evacuating agents, while the whole environment is managed by a monitor agent.

Each agent has its own field of view, i.e. they cannot see the whole grid, but a limited portion in their immediate vicinity (for instance, an agent might only see what is in an area of 9x9 grid cells around itself).

The simulation takes place as follows:

- Initially, the agents move randomly, ignoring the exits.
- After a certain time limit expires, the monitor will declare an emergency (and will broadcast an emergency message to the other agents).
- The agents, upon being notified of the emergency, will start moving in a random constant direction
- If an agent hits a wall, they will choose another random direction, preferably away from the wall.
- If an exit is in their field of view, they will prioritize moving towards it. If multiple exits are in their field of view, they should move toward the closest one. Two agents should not occupy the same grid cell, so if the path to the exit is blocked by other agents, they should wait until it clears and they can move towards their objective.
    - If the agent sees the exit and the direct path is blocked by other agents, they should check if there is an available free cell closer to the exit. If so, they should move to that cell and then continue attempting to reach the exit
- If an exit is not in their field of view, then
    - If the agent sees other agents, they should ask them if they see an exit. If an agent responds that they indeed see an exit, then the asking agent should follow the closest answering agent while it is in their field of view, until an exit is visible. If that does not occur for a certain amount of time, the agent should resume moving in a random direction.
    - In order to avoid deadlocks (i.e. two agents asking each other repeatedly if each sees an exit), an agent cannot ask the same agent another question for a certain amount of time.
    - If the agent receives negative responses from the other agents, they should move on, in a random direction
- Once an agent has reached an exit, it should be removed from the grid.

Measure how long it takes for all agents to evacuate. Experiment with various positions for the exits, and determine a placement which minimizes the evacuation time.

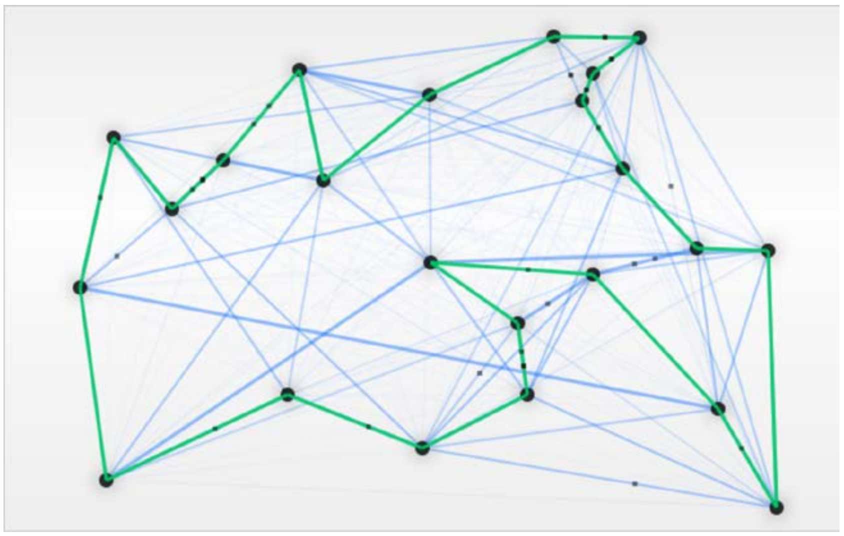
**Topic 4**

Implement the *Ant colony* path optimization algorithm using an agent system. The agent world is a connected graph.

- One node is the home base, i.e. the anthill.
- The others may have "food" - the amount of food is represented by an integer value. Nodes with value 0 have no food, while others have food units in various amounts (i.e. non-zero values).
- The edges of the graph have weights which are initially 0.
- Several ant agents start at the home base and, by default, start searching for food randomly, i.e. they randomly move from node to node along the edges of the graph.
- Once an ant has found food (has landed on a non-zero node), it takes away a food unit (the value of the node decreases by 1) and:
    - Version 1: returns the food unit to the anthill node
    - Version 2: deposits the food unit on a node half-way between its current node and the anthill node.
- In both versions, the ant carrying back a food unit increases the weight of each edge it traverses by 2. Once it deposits the food unit, the ant starts searching again.
- If an ant with no food unit (which is searching) lands on a node with an adjacent non-zero edge, it follows that edge and all subsequent connecting edges with non-zero weights, always preferring the edges with the highest weight. If it lands on a node with no non-zero edges (other than the one that was used to reach the node), it starts randomly searching again. The ant searches until it finds a non-zero node, in which case it returns the food item as described above.
- The weights of the nodes decay at a certain time interval (for instance, by -0.1 per second).

Implement both versions of the ant behavior and compare the two approaches in terms of the time taken to collect all food units (time until all nodes end up with zero values). A visual representation of the graph illustrating the node and weight values would help analyze the steps of the algorithm.

The example below illustrates an ant colony graph. The importance of the routes is represented via the thickness of the connecting graph and the  most important routes are highlighted in green. The individual ants are represented through the smaller black dots.



**Topic 5**

Implement a multi-agent application for load balancing in a distributed system. The system will process large data (i.e. encrypt a document, perform large matrix operations, various statistical calculations based on experimental data, etc.). Each component of the distributed system will be represented by a ProcessorAgent. Each processor agent can handle a specific work load.

The system will contain two more typed of agents:

The DistributorAgent will get the input data and randomly assign tasks to each ProcessorAgent. Obviously, this will lead to an imbalance, as some ProcessorAgents may be assigned more work than others. Some of the ProcessorAgents may be assigned more work than they can handle, in which case they will send an appropriate message to the dispatcher. In order to counteract the imbalance, the DispatcherAgent will ask the other ProcessorAgents which of them can handle extra work (which is working below their capacity).

- If there are replies, the Dispatcher will assign the extra work to the replying ProcessorAgents according to their remaining capacity.

- If there are no replies, the Dispatcher will generate HelperAgents, which can perform a fixed (and small) number of operations. Depending on the imbalance, HelperAgents will take over some of the tasks assigned to the ProcessorAgents. A ProcessorAgent may be aided by multiple HelperAgents according to its work load. Enough HelperAgents should be generated so as to cover the work that the ProcessorAgents can't handle on their own.

Each agent should display complete details on their status, the processed data, generated agents, operations distributed / carried out, where appropriate.

**Topic 6**

Implement a maze-solving algorithm using a multi-agent system. The maze is an n x m grid of cells, some of which can be passed-through (the "floor"), and some of which cannot (the "walls"). Multiple agents try to map out the maze and find an exit. During the search for the exit, each agent behaves as follows:

- Each agent keeps a map of the maze where for each cell the agent assigns 4 weight values, one for each direction: up, down, left, right. Initially, the directions leading into “floor” cells have a weight of 1 and the directions leading into “wall” cells will always have a weight of 0.

- The weights are in the range [0, 1]. A weight of 1 means that the agent may freely move in the corresponding direction, while a weight of 0 means the agent will always avoid moving in that direction. The lower the weight of a direction, the less likely that the agent will choose to move in that direction.

- An agent chooses the direction with the highest weight that isn’t occupied by another agent.

- If there exist multiple directions with the same weight, the agent will choose a direction not travelled by it (i.e. it will choose not to turn back).

- Whenever an agent moves in a certain direction, they will broadcast this direction to the other agents. The others will slightly decrease their own weights for that direction. This means that an agent will be less likely to travel to cells that have already been explored by others.

- If an agent encounters a dead end, they will turn back until they can once again move in multiple directions. They will set the weights of the directions leading to the dead end to 0 and they will inform the other agents accordingly.

- At some point an agent will discover a cell marked as “exit” and will broadcast its position to the others. The other agents will move to that position so that each agent exits the maze.

Run the application for multiple numbers of agents. Compare the time it takes for the maze to be solved by a single agent, or multiple collaborating agents, and present your findings in the documentation.

You may use a predefined maze or generate it procedurally. For procedual generation, the easiest method is *Prim’s algorithm* for generating perfect mazes (perfect maze = the is a unique path between any two positions).

A simple visual representation of the maze and the moving agents would be helpful in assessing the execution of the application.

**Topic 7**

Develop a system that implements a few voting protocols using agents. The system should contain the following types of agents:
- A moderator, who will manage the entire electoral process. It will collect the voters' choices and display the results of the election.
- At least 10 candidate agents, who will make their offers public via services, using at least 5 criteria (the same for all candidates). For each criterion, the candidate will submit an offer as a value in the [1, 10] range.
- At least 100 voter agents, which, for each criterion, will have their own preference also expressed by an amount in the range [1, 10]. They will receive each candidates’ offers and will calculate the likelihood of voting for each candidate using the formula from Equation 1. Each voter will vote for the candidate for which the highest probability was obtained. If there are multiple candidates with the same maximum probability, one will be chosen randomly.

$$P = \frac{\sum_{i=1}^{n} \frac{1}{|pr_i - of_i| + 1}}{n} \qquad (1)$$

where:
*P* - likelihood to vote for candidate
*n* - number of criteria
*pri* - voter preference from criterion i
*ofi* - offer made by the candidate for criterion i

The following protocols shall be implemented (the users should be able to choose the desired protocol):
Plurality voting:
- Each voter votes for the preferred candidate
- The candidate with the most votes wins

Approval voting:
- each voter votes for as many candidates as they want (in descending order of their preference)
- The candidate with the most votes wins

Single transferrable vote:
- Each voter votes for the preferred candidate
- If no candidate obtained more than 50% of the available votes, the candidate with the fewest votes is eliminated from the election
- Each voter who opted for the eliminated candidate transfer the corresponding votes to another candidate from the ones remaining; in this case, a new voting round takes place.

Run each voting protocol a few thousand times. For each protocol, calculate the probability that each candidate will win (the number of runs when the candidate has won / the total number of runs)