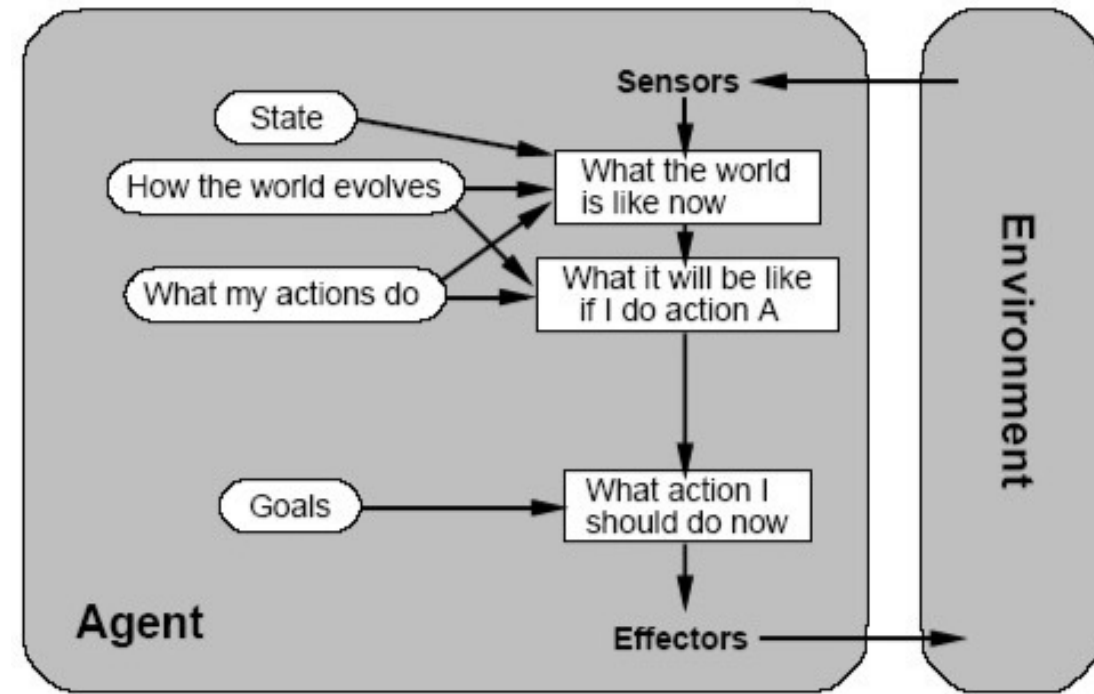


# Implementing A Deliberative Agent

LIA, I&C, EPFL

# What is Deliberative Agent

- Also named **mental** or **rational** Agents
- The agent has an **explicit model of the world** in which it lives. Seeks to perform **goals**. It is fully aware of the world it is acting in.
- A **planner reasons** on the world model and decides which actions to realize by producing a plan, in order to achieve its goals.



**TLDR: A deliberative agent has goals (e.g. to deliver all tasks) and is fully aware of the world it is acting in.**

# Deliberative: Can Carry Multiple Tasks

- Sequence of actions such that all tasks are delivered
  - Different from the reactive agent, **vehicle can carry multiple tasks** as long as the sum of the weights doesn't exceed the capacity of the vehicle.
- How to define the cost:
  - distance x costPerKm

Vehicle interface (logist.simulation.Vehicle)

The capacity of the vehicle.

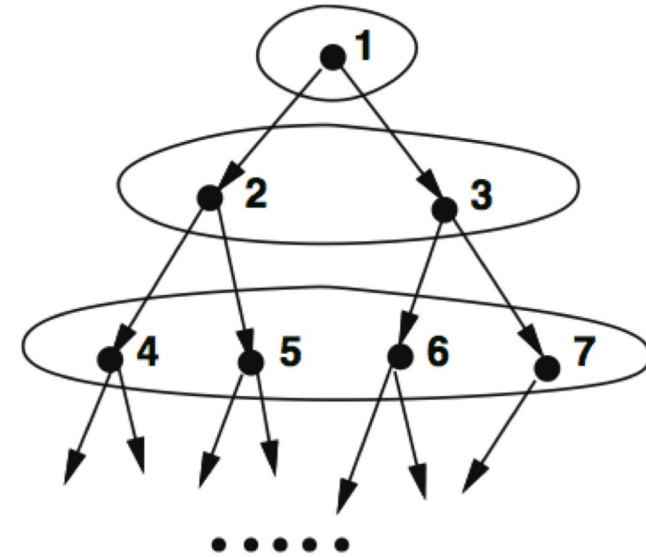
```
int costPerKm()
```

- Lausanne to Geneva: 63.4 Km
- Cost of BMW i3: £0.02 per Km

**Overall cost = 63.4 X 0.02= £ 1.268**

# Breadth-First Search Algorithm

```
1. Q ← initial node
2. C ← empty
3. repeat
4.   if Q is empty, return failure
5.   n ← first element of Q , Q ← rest(Q)
6.   if n is a final node, return n
7.   if n is not a member of C then
8.     add n to C
9.     S ← succ(n)
10.    Q ← append(Q, S)
11.  endif
12. end
```



## Advantages:

- Always finds the shortest plan.
- Not penalized by bad initial decisions.

## Disadvantage:

- Requires large amounts of memory to store all nodes of the tree at each level.

# A\* Search Algorithm

## Algorithm A\* (best-first)

```
1. Q ← initial node
2. C ← empty
3. repeat
4.   if Q is empty, return failure
5.   n ← first element of Q, Q ← rest(Q)
6.   if n is a final node, return n
7.   if n ∉ C, or has lower cost than its copy in C then
8.     add n to C
9.     S ← succ(n)
10.    S ← sort(S,f)
11.    Q ← merge(Q,S,f)
    (Q is ordered in increasing order of f(n) = g(n) + h(n))
12.  endif
13. end
```

$$f(n) = g(n) + h(n) ,$$

where n is the previous node on the path. g(n) is the cost of the path from the start node to n. h(n) is a heuristic that estimates the cost of the **cheapest** path from n to the target node.

# Step By Step: Overview

1. **Representation for the states**, transitions and goals (or final states)
2. Implement BFS and a state-based A\* search algorithm
3. Implement the deliberative agent.
4. Compare the performances of the BFS and the A\* search algorithms for different problem sizes.
5. Run the simulation with 1, 2 and 3 deliberative agents and report the differences of the joint performance of the agents.

# Some Hints about How to Define State

9 usages

```
private final City currentCity;
```

Current location of the vehicle

8 usages

```
private final TaskSet carriedTasks;
```

Already loaded tasks in this vehicle.

8 usages

```
private final TaskSet availableTasks;
```

Available tasks that haven't been taken

3 usages

```
private LinkedList<Action> actionsToReach;
```

Actions already taken to reach this state.

3 usages

```
private double costToReach;
```


Cost to reach this state

3 usages

```
private int currentVehicleCapacity;
```

Current available capacity

# Final State



```
public boolean isFinal() {  
    return this.carriedTasks.isEmpty() && this.availableTasks.isEmpty();  
}
```

- This vehicle already delivered all of its carried tasks.
- All the tasks have been delivered.



# Some Reminders

- Heuristic function:  $h(n)$  is the estimate of **mimimal** cost from state  $n$  to the target state.
- Start your method by define state and then implement the setup() and plan() function in `logist.behavior.Deliberative` interface.
- Get familiar with **Logist Platform(18 pages pdf file)** , which is very important for the following experiments.

**Thanks!**

# Deliberative Interface: setup()

- The behaviour of a deliberative agent is defined by a class that implements the `logist.behavior.Deliberative` interface.
- The setup method is called exactly once, before the simulation starts and before any other method is called. The agent argument can be used to access important properties of the agent.

```
void setup( Topology topology,  
            TaskDistribution distribution,  
            Agent agent )
```

# Deliberative Interface: plan()

- This signal asks the agent to compute the transportation plan for its vehicle. All tasks in tasks must be picked up and delivered by the plan.
- In a single agent system, the agent can assume that the vehicle is carrying no tasks initially.

```
Plan plan( Vehicle vehicle,  
           TaskSet tasks )
```