Search Project

Artificial Intelligence

Maitane Gómez Sainz 72316851 – K
Jose Mª González Gamito 45821905 - V

1. **Identify this problem's environment properties and define an appropriate performance measure.**

   Performance Measure, Environment, Actuators and Sensors (PEAS):

   - **Performance Measure**, that describes the success criteria for the system's behavior: arrive to the parking (the F) following the possible path.

   - **Environment**, context where the system perceives and acts: A grid of X rows and Y columns in which each of them can have an O, and X or an F (with walls around) and a car.

   - **Actuators or actions,** that are the system's response to each percept set or sequence: Turn 90º to the left or to the right and go ahead if we are in an X or maintain the same way if we are in an O.

   - **Sensors,** that are the system's perceptual inputs from the environment at a certain instant: Detect which tile we are in, its type and if it have walls.

   This system is **fully observable** that means that the system's sensors give it access to the complete state of the environment at each point in time.

   It is a **single system** because operates by itself in the environment.

   It is **deterministic** because the next state of the environment is completely determined by the current state and the action executed by the system.

   It is **sequential**, because the current decision can affect all future decisions.

   It is **static**, because the environment is unchanged while the system is deliberating.

   It is **discrete** because it has a limited number of distinct, clearly defined actions.

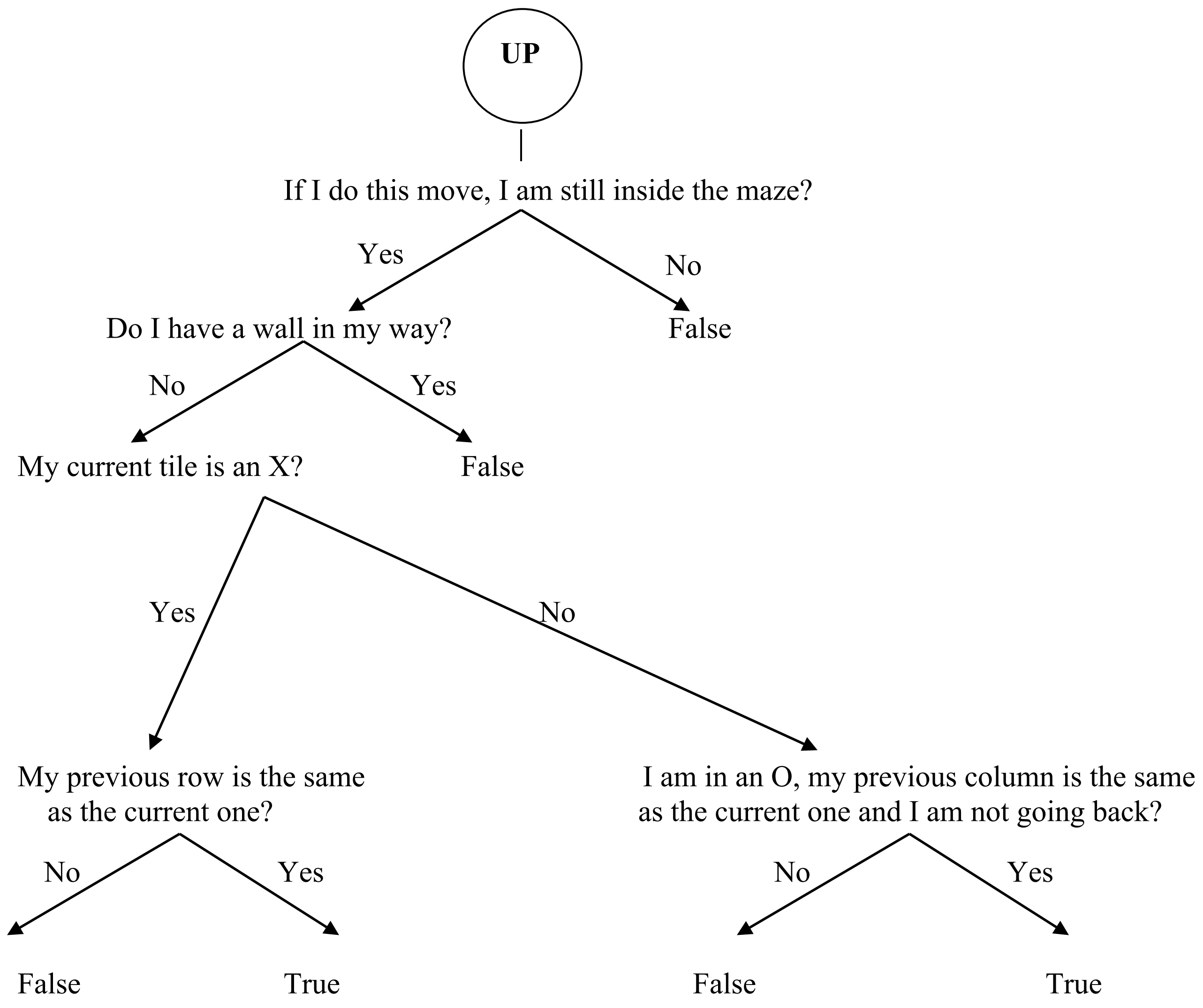
   And finally, it is **known** because the system knows the outcomes of all actions.

2. **Give the precise problem formulation needed for the problem to be solved by a search method.**

   - *States:* they are the position of the car in every moment.
   - *Initial state:* the car is in some place out of the maze.
   - *Goal test:* is the car in the flag?
   - *Actions:* move if the type of the square allows me to move and there is not a wall blocking the movement.
   - *Transition model:* the car has moved so we are in a different square.
   - *Path cost:* the number of movements.

The flow diagram for the operator to show how the actions are done is the following:

It is done for the Up action because the others are the same.

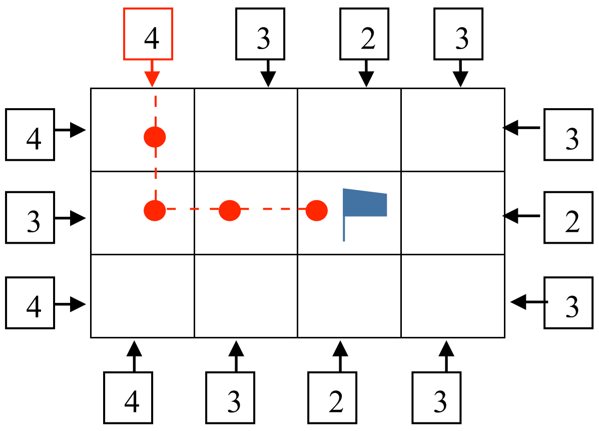

UP
If I do this move, I am still inside the maze?
Yes
No
Do I have a wall in my way?
False
No
Yes
My current tile is an X?
False
Yes
No
My previous row is the same as the current one?
I am in an O, my previous column is the same as the current one and I am not going back?
No
Yes
No
Yes
False
True
False
True

3. **Reason whether it is possible to apply the Relaxed Problem Technique to define an admissible and consistent heuristic. If so, apply this technique and give the resulting heuristic. Otherwise, explain whether you can define any other admissible and consistent heuristic.**

   The relaxed problem technique consists in removing some restrictions to the actions of the problem, we think that it is possible to apply this technique to our problem, and we are going to explain it.

   The restrictions of our problem will be, that the tile of the parking has walls around, so there is only one possible entrance. On the other hand, every tile has its own movement, so in some tiles we can only go ahead and in others we can only turn. We have decided to remove those restrictions to the actions so we can allow the car to move without taking into account the type of the tile and the walls. However, we have to think that the diagonal movements are not allowed.

For example, in the following picture we can see the path without restrictions, realizing that the number of actions of the path coincides with a Manhattan Distance. So we have decided to apply that distance as our heuristic. Justifying that we can use the relaxed problem technique.



4. **Explain which search method is best to solve this problem. In your explanation you should include a comparison of the method of your choice against others you discard.**

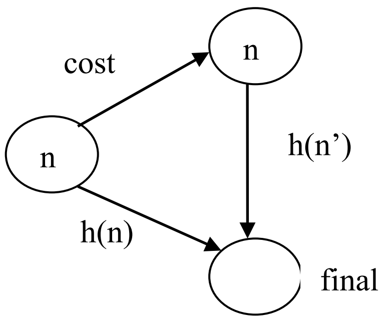
To solve this problem we have decided that the best method is to use an Informed search algorithm, in this case A* based on graph search.

We have chosen this method attending to these criteria:

- Informed search uses a heuristic, to help us getting the solution in a more optimal way. So we have discarded Blind search algorithms.
- Greedy best first is complete with graph search, so we discard this algorithm based on basic search. We should use this method but it is not optimal too so we discard it too.
- Finally, we have A* based on graph search, that is optimal if its heuristic is admissible and it takes into account the repeated states to find the best path.

An heuristic of a given node (n) is admissible when it is not bigger than the sum of the cost and the other heuristic of n', that is:

$h(n) < cost + h(n')$

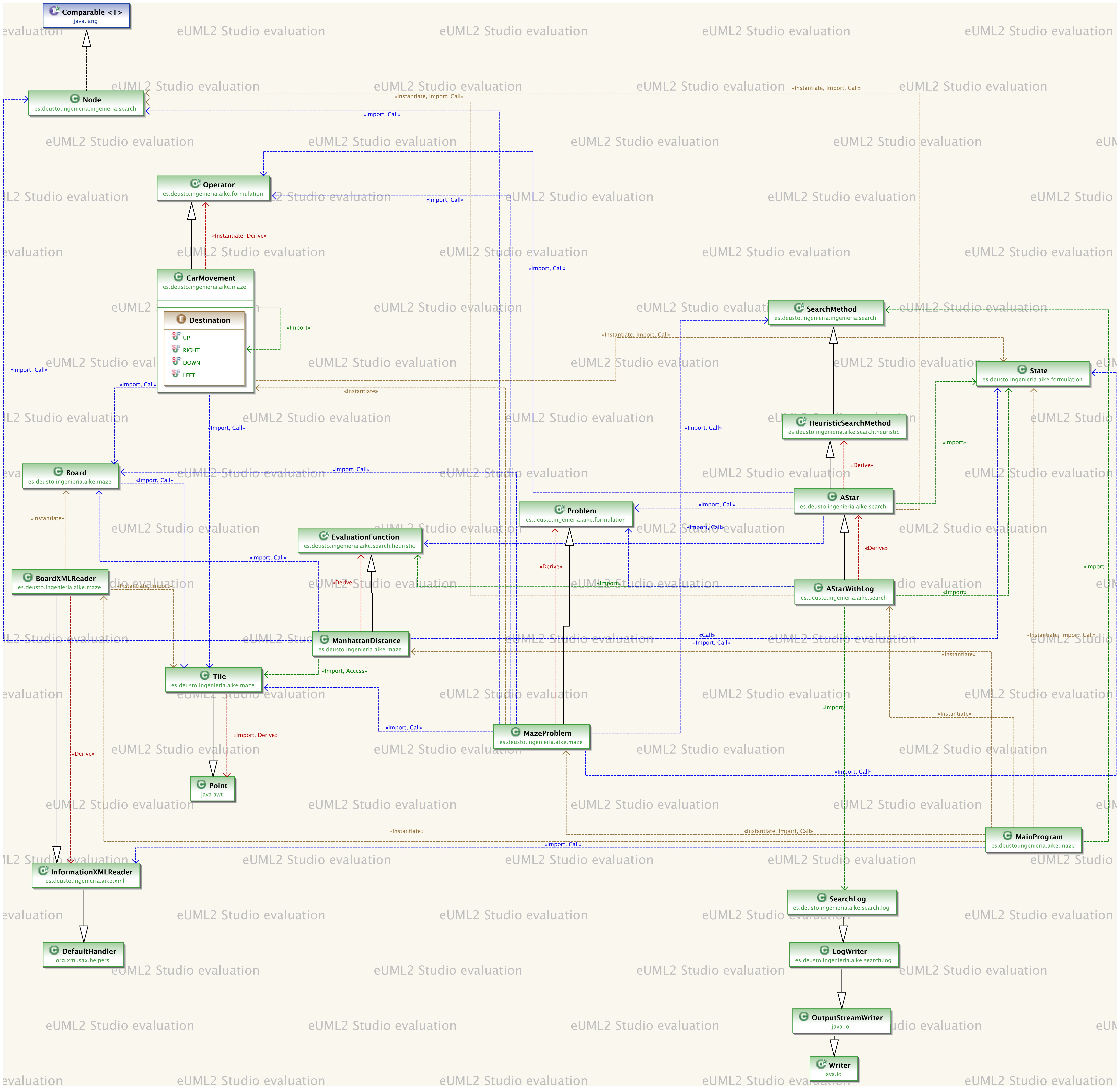


This algorithm is not provided with the templates so it has been decided to implement it in another new classes, using some classes of the templates as Node or State.

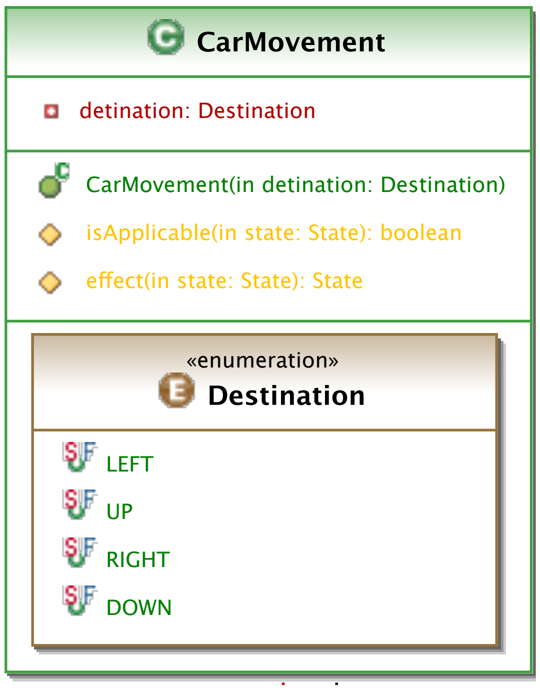
5. UML class diagram that include the objects from the templates FORMULATION and SEARCH discussed during the practical sessions.

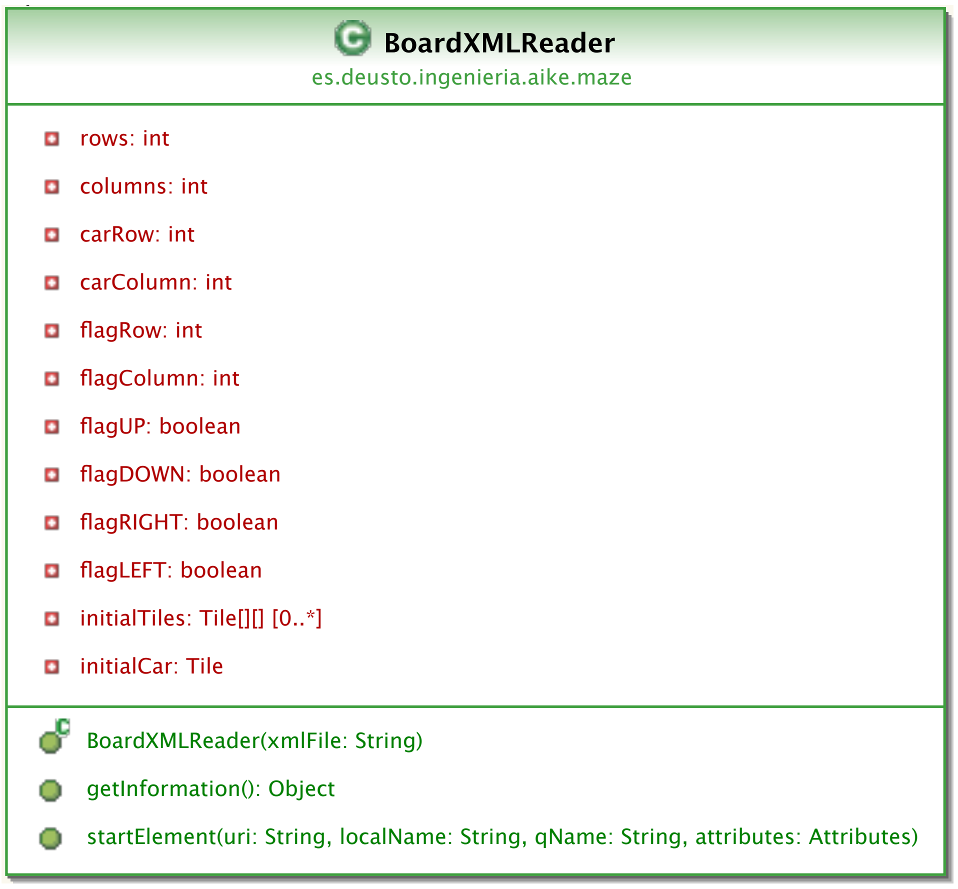
The UML Diagram of the project has been divided into two sections just because the huge amount of relations between classes. This way, it will be easier for the reader of the document to understand the whole diagram.
The first one shows the relation between all the classes in the project (and templates) whilst the second one shows the structure of each new class generated for the implementation of the activity.

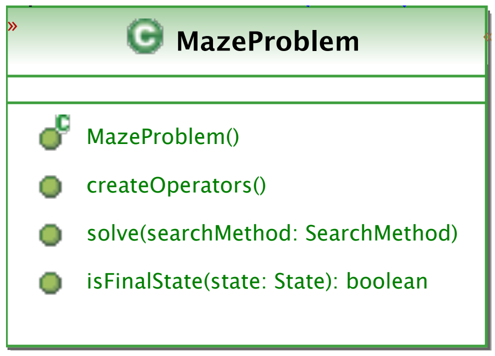
Both diagrams can be found in the following pages:

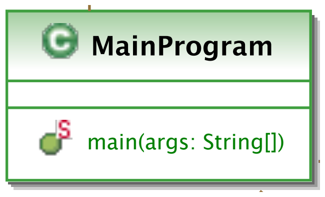


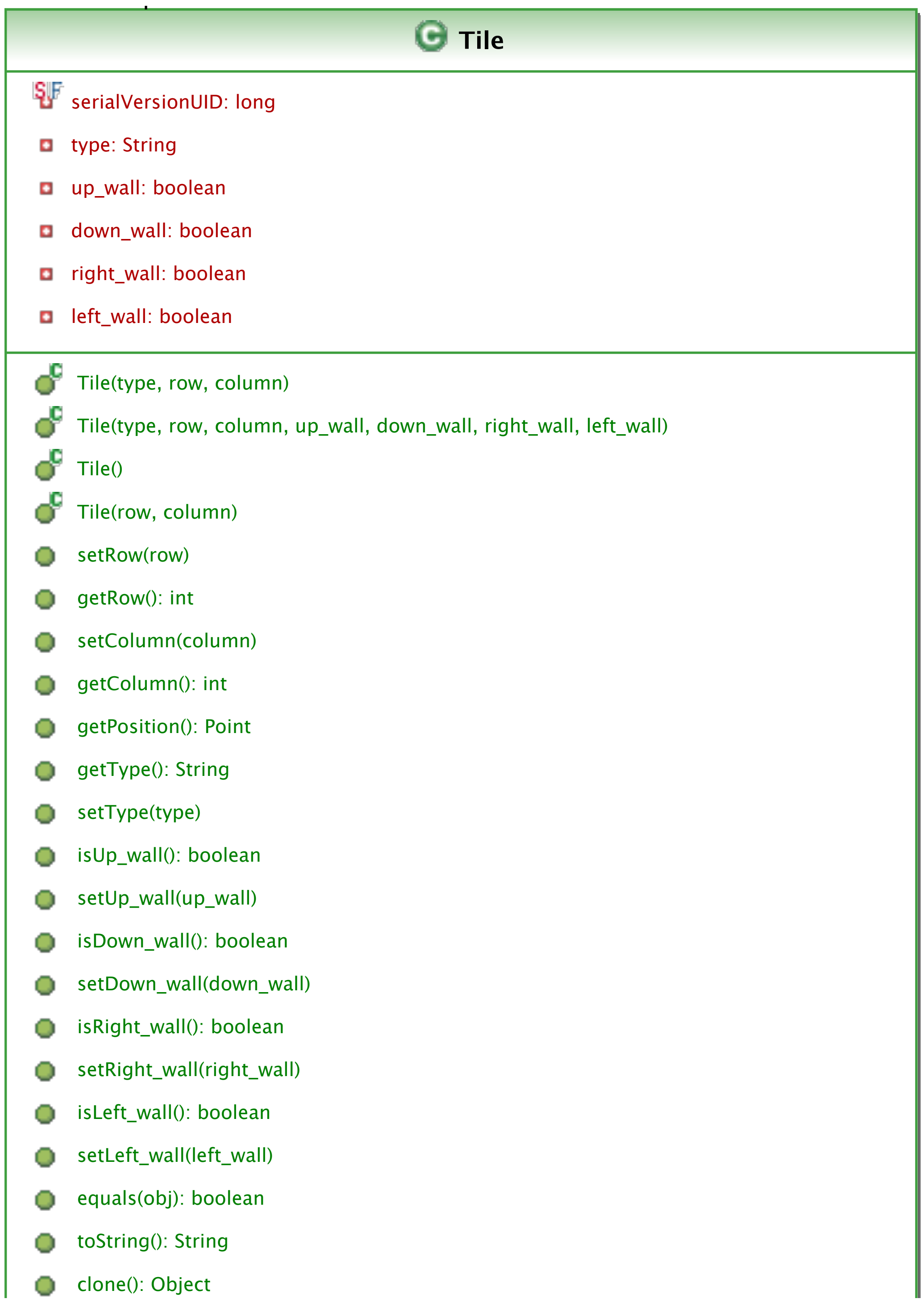
Comparable <T>
java.lang
Node
es.deusto.ingenieria.ingenieria.search
Operator
es.deusto.ingenieria.aike.formulation
CarMovement
es.deusto.ingenieria.aike.maze
Destination
UP
RIGHT
DOWN
LEFT
SearchMethod
es.deusto.ingenieria.ingenieria.search
State
es.deusto.ingenieria.aike.formulation
HeuristicSearchMethod
es.deusto.ingenieria.aike.search.heuristic
AStar
es.deusto.ingenieria.aike.search
Problem
es.deusto.ingenieria.aike.formulation
EvaluationFunction
es.deusto.ingenieria.aike.search.heuristic
Board
es.deusto.ingenieria.aike.maze
BoardXMLReader
es.deusto.ingenieria.aike.maze
AStarWithLog
es.deusto.ingenieria.aike.search
ManhattanDistance
es.deusto.ingenieria.aike.maze
Tile
es.deusto.ingenieria.aike.maze
MazeProblem
es.deusto.ingenieria.aike.maze
Point
java.awt
MainProgram
es.deusto.ingenieria.aike.maze
InformationXMLReader
es.deusto.ingenieria.aike.xml
DefaultHandler
org.xml.sax.helpers
SearchLog
es.deusto.ingenieria.aike.search.log
LogWriter
es.deusto.ingenieria.aike.search.log
OutputStreamWriter
java.io
Writer
java.io
«Instantiate, Import, Call»
«Import, Call»
«Instantiate, Derive»
«Import»
«Instantiate»
«Derive»
«Import, Access»
«Instantiate, Import»
«Import, Derive»
«Call»
eUML2 Studio evaluation

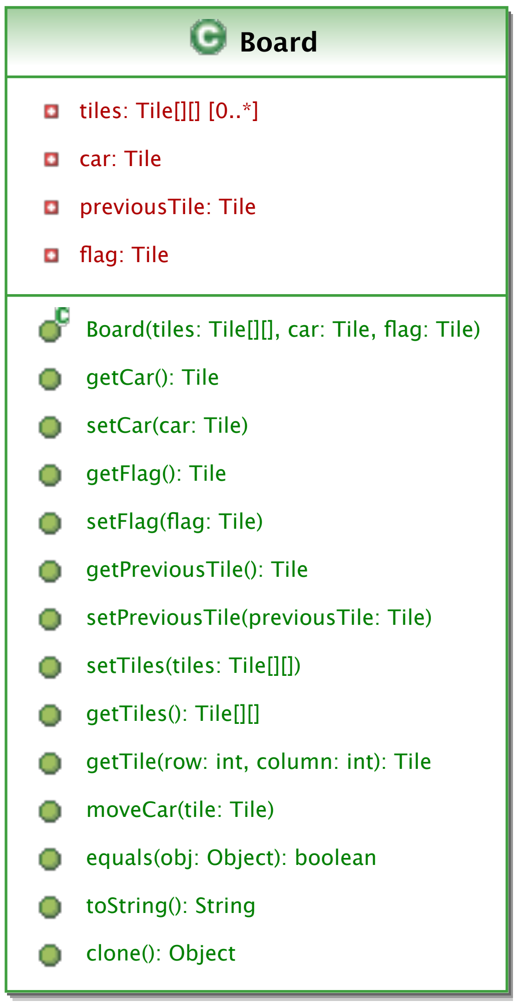

CarMovement
detination: Destination
CarMovement(in detination: Destination)
isApplicable(in state: State): boolean
effect(in state: State): State
«enumeration»
Destination
LEFT
UP
RIGHT
DOWN

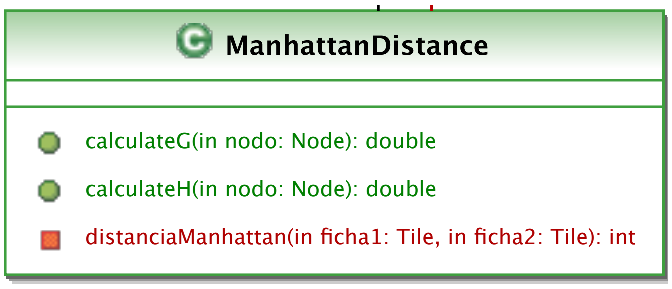

BoardXMLReader
es.deusto.ingenieria.aike.maze
rows: int
columns: int
carRow: int
carColumn: int
flagRow: int
flagColumn: int
flagUP: boolean
flagDOWN: boolean
flagRIGHT: boolean
flagLEFT: boolean
initialTiles: Tile[][] [0..*]
initialCar: Tile
BoardXMLReader(xmlFile: String)
getInformation(): Object
startElement(uri: String, localName: String, qName: String, attributes: Attributes)

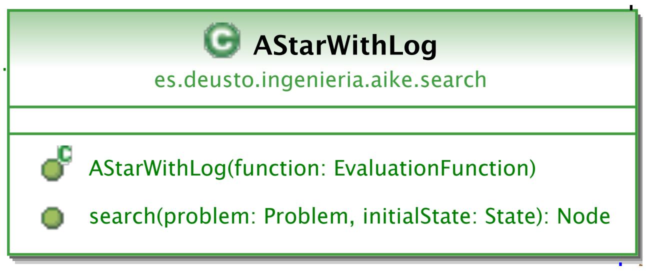

MazeProblem
MazeProblem()
createOperators()
solve(searchMethod: SearchMethod)
isFinalState(state: State): boolean

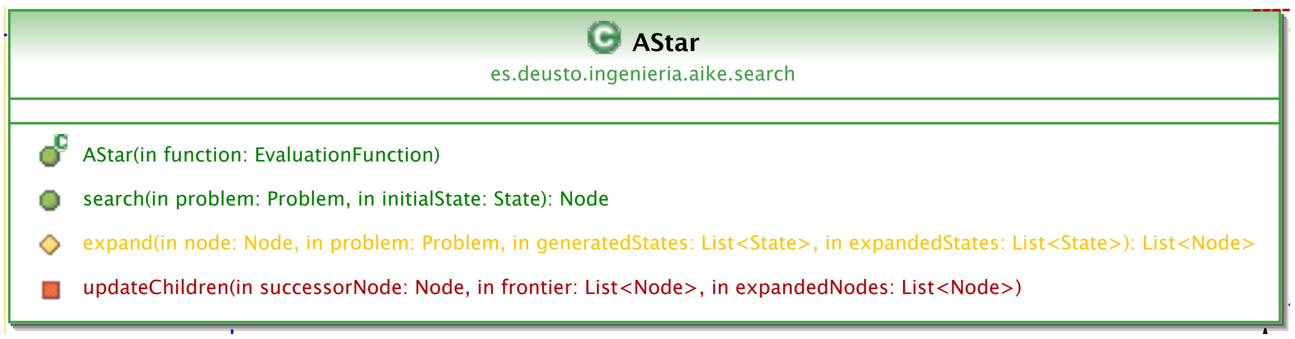

MainProgram
main(args: String[])


Tile
serialVersionUID: long
type: String
up_wall: boolean
down_wall: boolean
right_wall: boolean
left_wall: boolean
Tile(type, row, column)
Tile(type, row, column, up_wall, down_wall, right_wall, left_wall)
Tile()
Tile(row, column)
setRow(row)
getRow(): int
setColumn(column)
getColumn(): int
getPosition(): Point
getType(): String
setType(type)
isUp_wall(): boolean
setUp_wall(up_wall)
isDown_wall(): boolean
setDown_wall(down_wall)
isRight_wall(): boolean
setRight_wall(right_wall)
isLeft_wall(): boolean
setLeft_wall(left_wall)
equals(obj): boolean
toString(): String
clone(): Object


Board
tiles: Tile[][] [0..*]
car: Tile
previousTile: Tile
flag: Tile
Board(tiles: Tile[][], car: Tile, flag: Tile)
getCar(): Tile
setCar(car: Tile)
getFlag(): Tile
setFlag(flag: Tile)
getPreviousTile(): Tile
setPreviousTile(previousTile: Tile)
setTiles(tiles: Tile[][])
getTiles(): Tile[][]
getTile(row: int, column: int): Tile
moveCar(tile: Tile)
equals(obj: Object): boolean
toString(): String
clone(): Object


ManhattanDistance
calculateG(in nodo: Node): double
calculateH(in nodo: Node): double
distanciaManhattan(in ficha1: Tile, in ficha2: Tile): int


AStarWithLog
es.deusto.ingenieria.aike.search
AStarWithLog(function: EvaluationFunction)
search(problem: Problem, initialState: State): Node


AStar
es.deusto.ingenieria.aike.search
AStar(in function: EvaluationFunction)
search(in problem: Problem, in initialState: State): Node
expand(in node: Node, in problem: Problem, in generatedStates: List<State>, in expandedStates: List<State>): List<Node>
updateChildren(in successorNode: Node, in frontier: List<Node>, in expandedNodes: List<Node>)