INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Inteligência Artificial para Sistemas Autónomos

Teacher:
Luís Morgado

Diogo Saraiva 49756

Lisbon, July 2023

Index

1. Introduction

Within the scope of the Artificial Intelligence for Autonomous Systems course, it was proposed to carry out an individual project, which was carried out during classes throughout the semester.

This course's main areas of knowledge required for its completion are Software Engineering and Artificial Intelligence, and its organization is designed so that both areas of computer engineering are targeted. Within these areas, the architectures of intelligent, reactive and deliberative agents, automatic reasoning and decision making, and reinforcement learning are worked on.

Software engineering is an area of engineering focused on the development, maintenance and creation of software to achieve greater organization, productivity and quality. Artificial intelligence refers to the development of computer systems capable of performing tasks that normally require human intelligence.
By using these two areas simultaneously, we can develop tools and systems that are autonomous and adaptable to different situations, creating an automatic reasoning system or an agent with certain adaptive capabilities within a given set of restrictions.

The objective of this report is to show and explain the decisions made in the development process of the applications developed in class.

2. Theoretical Framework

2.1 Artificial Intelligence

Artificial intelligence is a scientific area that studies the development of computational systems capable of intelligent behavior. This can be divided into its main paradigms, which are: symbolic, connectionist and behavioral.

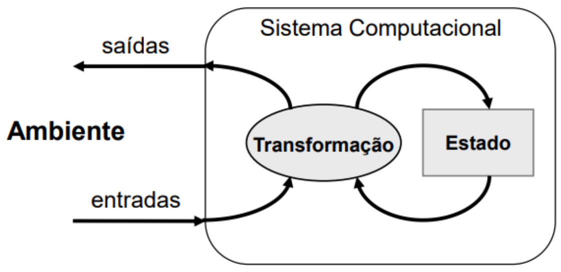
In relation to the symbolic paradigm, it is believed that intelligence is the result of the manipulation of symbols and symbolic structures through computational processes, in this case symbols are used to represent concepts related to a certain domain, such as the real world. In the connectionist paradigm, information is not represented and processed through discrete symbols, as in the symbolic paradigm. Instead, information is distributed and processed in the connections between neurons. Each neuron receives input signals, performs some type of processing in relation to these signals and generates an output signal that can be transmitted to other neurons in the network, it is believed that this intelligence emerges from the interactions of a large number of processing units elements interconnected with each other. In the behavioral paradigm, the focus is on the observable reactions and behaviors of a system. These reactions are modeled as responses to specific stimuli. Through observation and analysis of these relationships, it is possible to understand and predict the behavior of the system.

An intelligent agent is characterized by its autonomy, reactivity, proactivity and sociability, all these characteristics of an agent are associated with the achievement of the agent's purpose. To solve the proposed problems, the intelligent agent can reason, learn and know the environment in which it is surrounded.

2.2 Dynamic Models

The model of a computational system is characterized by three main perspectives: structure, dynamics and behavior.

The structure of a system is the set of parts and relationships between parts that constitute it. This is its organization in space, the relationships between the parts of a system and its state (memory). The dynamics of a system is its evolution over time and the set of states that a system can assume and the way in which they evolve over time, thus determining the system's behavior. The behavior of a system corresponds to the way the system acts when faced with information coming from the outside, expressing the structure and dynamics of the system.



The dynamics of a system can be expressed as a transformation function that, given the current state and current inputs, produces the next state and respective outputs.

2.3 Software Engineering

Software engineering is an area of engineering oriented towards the specification, development and maintenance of software, which aims to develop, operate and maintain software in a systematic and quantifiable way.

With the growth in the use of software-based systems, the complexity of these systems has increased. Complexity represents the difficulty of developing, operating and maintaining software, the increasing sophistication of the software produced, as well as the increasing amount of processing and memory used. Another consequence of the growth in the use of software-based systems is the changes that systems have to go through; this software need to be produced, or modified, to meet the needs of the respective contexts of use.

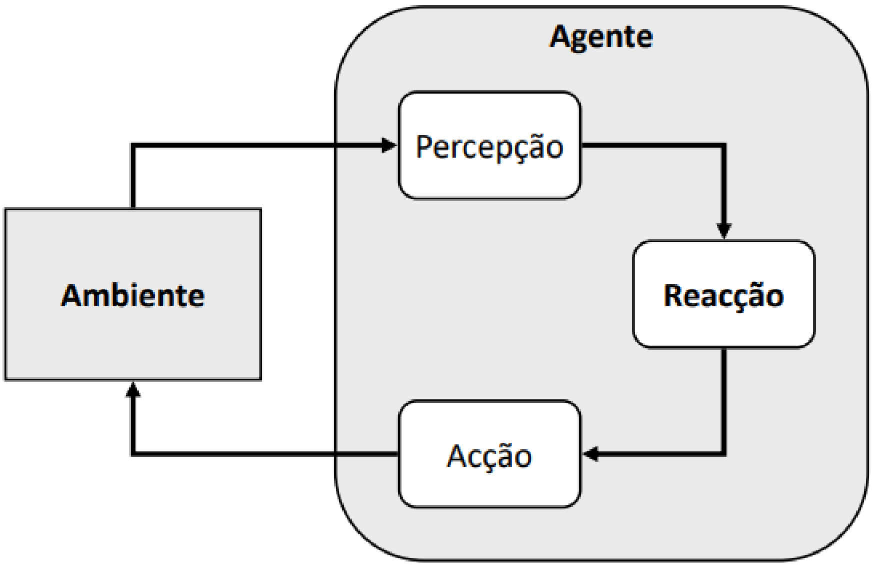
There are measures to quantify software architecture that indicate the quality of that architecture, these are coupling – degree of interdependence between subsystems, cohesion – level of functional coherence of a subsystem, simplicity – level of ease of understanding of the architecture, and adaptability – level ease of changing the architecture to incorporate new requirements or changes to previously defined requirements.

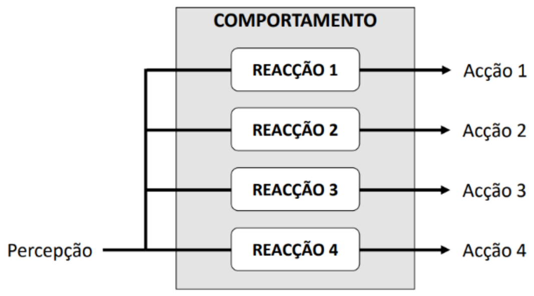
A good software engineering system is one that is efficient, reliable, of high quality and meets the needs of users. To this end, its development seeks to maintain a low level of coupling and high cohesion, without ever neglecting complexity management.

2.4 Reactive Agents

When it comes to intelligent agent architectures, there are several approaches that can be taken. For this course, reactive and deliberative agent architectures were explored. These two approaches have the same objective, allowing the agent to make decisions and act in an environment.

In this case the focus is the architecture of reactive agents, in this type of architecture the system's behavior is generated reactively, that is, based on associations between stimuli and responses.
These agents are designed to make decisions based only on immediate information from the environment, generally following a set of pre-defined rules. These rules can be implemented using conditionals or decision tables.





In the previous example we see that the agent has a simple behavior, that is, it is just composed of a reaction, association of stimuli with responses, however these agents can be composed of sets of reactions, which are organized in a modular way, these sets of reactions are compound behaviors as opposed to a single reaction which is a simple behavior.

In the case of having compound behaviors, we still must choose which actions to perform, this selection can be made through various combination and selection mechanisms: Hierarchy – behaviors are organized in a fixed hierarchy of subsumption; Priority – responses are selected according to an associated priority that varies throughout execution; Fusion – responses are combined into a single response by composition.

Within reactive agents there are several types of agents, including agents without memory, agents with memory, agents with state, etc.
All these agents have advantages and disadvantages, so the choice of the type of agent to use also depends on the agent's objectives in the problem.

2.5 Automatic Reasoning

Automatic reasoning refers to the ability of a computational system to automatically solve a problem based on a representation of knowledge, this process produces conclusions based on that knowledge.
The automatic reasoning process involves two main types of activities, exploring options and evaluating options to decide on the best options.

The exploration of options is based on prospective reasoning, anticipation of what may happen, and internal simulation of the problem domain, requiring forms of internal representation of the problem domain.

When evaluating options, the cost, necessary costs, and value, gain or loss, measured for example in terms of utility, are assessed.

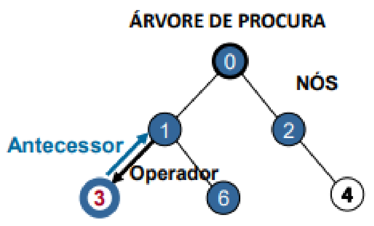
The representation of the problem model is made by states and operators. A state represents a configuration in solving a problem and contains a unique identification. An operator represents an action, that is, a state transformation.

2.6 State Space Search (PEE)

Automatic reasoning through search in state spaces is a method of solving problems, namely planning problems, in which the aim is to find a sequence of states and actions that lead an initial state to a final state.

This search process occurs by exploring the state space starting from the initial state. The state space is represented as a graph, where each vertex corresponds to a state and each arc corresponds to a transition between states. In each processing step, it is checked whether the current state corresponds to the objective. If the current state is not an objective, this state is expanded, and all successor states are generated by applying the various possible operators. For each successor state the process is repeated. If there are no successor states, the search process ends with the indication that there is no solution.

To maintain the information generated in each search step, an information structure is maintained, called a search tree, which consists of a tree structure, organized into nodes, that maintains information relating to each state transition explored.



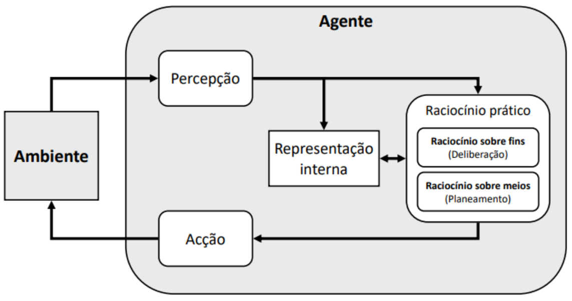
State space search has two types of search methods, these being uninformed search and informed search. Both search methods take advantage of knowledge of the problem domain to order the exploration frontier, however there is a big difference between them. While uninformed search methods make an exhaustive exploration of the state space, informed search methods make a selective exploration of the state space.

In this UC we saw uninformed search methods such as breadth-first search, uniform cost search and depth-first search, including limited depth search and iterative depth search. In addition to these, there are also informed search methods, the method focused on in this UC was the best-first search, but within this search we studied the main variants, uniform cost search, hasty search and A* search.

For different problems we can use different search methods, to choose the best search method to use for a given problem we have to take into account 3 main aspects, they are, checking whether the method is complete, that is, whether the search guarantees that, if there is a solution, it will be found, check whether the search method is optimal, which guarantees that, if there are several solutions, the solution found is the best and finally the complexities, temporal and spatial, time required and memory required to find a solution respectively.

2.7 Deliberative Agents

As stated in point 2.4, there are several agent architectures, at this point we will focus on the architecture of deliberative agents.
The main difference between reactive and deliberative agents is that in the architecture of deliberative agents, memory plays a central role in generating the agent's behavior. Some of these agents can even anticipate future states.



In a deliberative architecture, the components of practical reasoning, deliberation and planning, are supported by an internal representation of the environment, which can be updated based on perceptions obtained from the environment.
In a deliberative architecture, practical reasoning supports the general decision-making process that determines the agent's behavior, that is, what actions to take given the perceptions obtained and the state of the internal model of the world.

For the general decision-making and action process, there are 5 steps:
- Observe the world, generating perceptions.
- Update the model of the world, based on perceptions.
- Deliberate what to do, generating a set of objectives.

- Plan how to do it, generating an action plan.
- Execute the action plan.

However, the 3rd and 4th steps are only taken if it is necessary to reconsider the model of the world.

2.8 Sequential decision processes

Sequential decision processes are those in which a series of decisions are made over time, rather than being made in a single, isolated manner. In this type of process, decisions are made based on the information available at each stage and considering the impact of previous decisions.

In a sequential decision process, the evolution between states occurs because of actions that, in general, may be non-deterministic, that is, their result may not be completely predictable, there may be uncertainty in the result of the action.

The Markov Decision Process (PDM) is a mathematical model used to study sequential decision-making problems in stochastic environments. It is called a Markov process because it obeys the Markov property, which states that the future state depends only on the present state and not on previous states.

The objective of PDM is to find an optimal policy that maximizes the total expected reward over time. This is done by reinforcement learning algorithms.

S – conjunto de estados do mundo

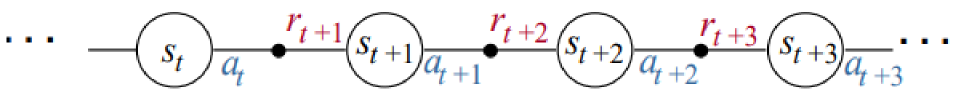
$A(s)$ – conjunto de acções possíveis no estado $s \in S$

$T(s,a,s')$ – probabilidade de transição de s para s' através de a

$R(s,a,s')$ – recompensa esperada na transição de s para s' através de a

γ – taxa de desconto para recompensas diferidas no tempo

$t = 0, 1, 2, \ldots$ – tempo discreto



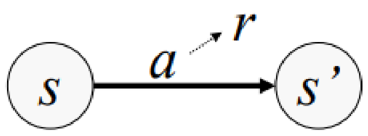
Cadeia de Markov

In this domain, utility is the cumulative effect of the evolution of the situation and reward is the gain or loss in each state, whether these finite values are positive or negative. The policy is the way of representing the agent's behavior and defines what action should be carried out in each state. Policy can be deterministic or non-deterministic.

Following the principle of optimal solution, dynamic programming is used, that is, the problem is decomposed into several smaller problems. In a PDM this derives from the assumption of path independence and the utilities of the states can be determined depending on the utilities of the successor states.

2.9 Reinforcement learning

Reinforcement learning is learning obtained from interaction with the environment, between the state (s, s'), the action (a) and the reinforcement (r) whether gain or loss.



In reinforcement learning, the following elements are defined:
- States(s): these represent the different situations or configurations in which the agent can be.
- Actions (a): are the options available to the agent in each state. The agent chooses an action based on the current state.
- Transition function (T): describes the probability of transition from one state to another after executing an action.
- Reward function (R): assigns a reward or penalty to the agent based on the current state and the action performed.
- Policy (ε): defines the agent’s strategy, that is, the action to be taken in each state.

3. Project Realized

3.1 Game

For the first project the objective was to implement a game with a virtual character that interacts with a human player. The game consists of an environment where the character aims to record the presence of animals through photographs.
The concept of the problem domain is the game, which is divided into two, the environment and the character.

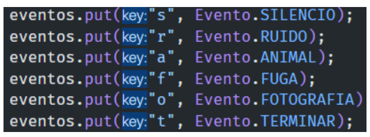
3.1.1 Project Development

To begin the development of this project we had to be able to interpret UML language diagrams, because all the architecture, structure models and organization of subsystems were provided to us through sequence and class diagrams. The correct analysis and interpretation of these given diagrams was essential for the project to work as intended.

Through the architecture and structure models of the game, the environment and the character, we implemented the following classes:

- Game – class responsible for running the game.
- Environment – evolutionary structure with which the character will interact.
- Character – class that represents the character and interacts with the environment.
- Event – list of possible events to occur in the game.
- Perception – class that updates the character's perception depending on the environment.
- Action – list of possible actions for the character to perform.

In this project, all interaction with the player is done in text mode through the console. Therefore, letters were assigned to each possible event.



```
eventos.put(key:"s", Evento.SILENCIO);
eventos.put(key:"r", Evento.RUIDO);
eventos.put(key:"a", Evento.ANIMAL);
eventos.put(key:"f", Evento.FUGA);
eventos.put(key:"o", Evento.FOTOGRAFIA)
eventos.put(key:"t", Evento.TERMINAR);
```

It was then necessary to transform a state machine into a character-independent library for use in general contexts. For this we used generic types, both for events and actions.

We then implemented the necessary classes for this state machine through class and sequence diagrams, they are:

- MaquinaEstados – class that represents the generic state machine.
- State – class responsible for representing a state in a transition model.
- Transition – class that represents a state machine transition, contains a successor state and an associated action.
- Control – class responsible for associating an action with each state.

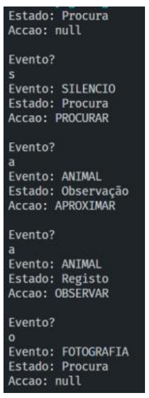
In the control class, the possible states for the character and all existing transitions were defined.

Estado	Evento	Novo estado
Procura	Animal	Observação
Procura	Ruido	Inspecção
Procura	Silencio	Procura
Inspecção	Animal	Observação
Inspecção	Ruido	Inspecção
Inspecção	Silencio	Procura
Observação	Fuga	Inspecção
Observação	Animal	Registo
Registo	Animal	Registo
Registo	Fuga	Procura
Registo	Fotografia	Procura

Estado	Evento	Acção
Procura	Animal	Aproximar
Procura	Ruido	Aproximar
Procura	Silencio	Procurar
Inspecção	Animal	Aproximar
Inspecção	Ruido	Procurar
Inspecção	Silencio	
Observação	Fuga	
Observação	Animal	Observar
Registo	Animal	Fotografar
Registo	Fuga	
Registo	Fotografia	

3.1.2 Tests

As we can see in the following output, the results are as expected and we can conclude that the agent always took the appropriate actions, thus we consider the first project as completed.



```
Estado: Procura
Accao: null

Evento?
s
Evento: SILENCIO
Estado: Procura
Accao: PROCURAR

Evento?
a
Evento: ANIMAL
Estado: Observação
Accao: APROXIMAR

Evento?
a
Evento: ANIMAL
Estado: Registo
Accao: OBSERVAR

Evento?
o
Evento: FOTOGRAFIA
Estado: Procura
Accao: null
```

3.2 Reactive Agent

For this second project the objective was also to create a reactive agent, however for this project the agent has to collect targets and avoid obstacles.

3. 2 .1 Project Development

Since this project is significantly more complex than the previous one, it was decided to divide the modules into 3 different packages, so that organization and implementation would become simpler, they are:

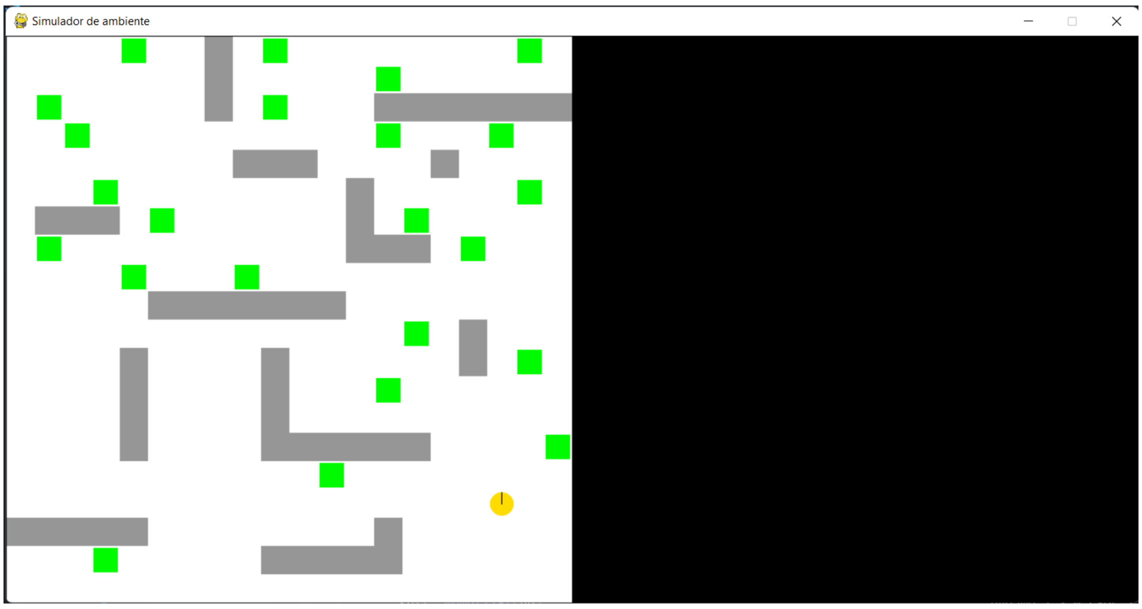
- Test – package that contains the project test.
- Reactive Control – package that contains all behaviors, reactions and stimuli associated with the agent.
- ECR – package that contains all the base classes, these serve to support reactive control.

Firstly, we started by implementing the respective classes to the ECR package. The first thing was to implement the Reaction, for this it was necessary to also implement the Stimulus interface and the Response class. This package serves to define the responses of a reactive agent to different stimuli. Here, sensory perceptions are mapped to specific actions, to guide the agent's behavior.

It was then necessary to implement reactive control, this will ensure that the actions taken by the system are determined exclusively by the current environmental conditions, without considering the past history or the future state. With this, the reactions themselves are implemented. Here there is still another division in the packages, namely, approach, avoid, explore, collect and response. They all communicate with each other so that the agent can react based on a given stimulus.
After this implementation we can then move on to testing.

3. 2 .2 Tests

Executing the test_react.py class it is possible to see that the program works correctly, the agent explores the environment until it finds a target or an obstacle, if it encounters an obstacle it ends up avoiding it, in the case of finding a target it moves if even at the same.



3.3 State Space Search

This project aimed to implement a deliberative agent with a search mechanism.

3. 3 .1 Project Development

For this project it was also necessary to divide its modules into several packages, because the complexity of the projects is considerable.
These were the packages created to facilitate understanding the problem:

- Mod – this package represents the model of a problem, encompassing the operator and problem state classes.
- Pee – here you will find all the search mechanisms used in this project, which is made up of the mec_proc, prof, larg and melhor_prim modules.
- Plan_traj – package that contains classes related to route planning.
- Controlo_delib – package that contains the classes necessary for the decision-making process.

Firstly, we implemented the model of a general problem, this being the Problem, State and Operator classes, these are essential when it comes to representing a problem.
Next, it was necessary to create an abstract MecanismoProcura class to represent all the search mechanisms that would be implemented next, it was also necessary to create several classes that represented the borders, nodes and solutions. All of these are quite necessary for a search engine.
Two types of boundaries also had to be implemented, FIFO (First-In, First-Out) and LIFO (Last-In, First-Out) to be used in the different uninformed search mechanisms. As the names indicate in the FIFO frontier the first nodes to be expanded are the oldest nodes, in the LIFO frontier the first nodes to be expanded are the most recent nodes. For informed search mechanisms, the Priority source was used. This is used in these mechanisms because it allows the use of an evaluator function for searches, in addition to using the heuristic function to order the exploration frontier.
Regarding search engines, classes were created for all mechanisms used in this project, namely:

- Depth-first search – used with the LIFO frontier.
- Graph search – implements graph search, preventing the repetition of states and the formation of cycles.
- Breadth-first search – inherits from the graph search and uses the FIFO frontier, this search is optimal and complete.
- Limited depth search – same as depth search, but only searches up to the previously defined maximum depth.
- Iterative depth search – iterates searches in limited depth, where in each iteration its limit is increased by 1.
- Search Best First – uses the priority frontier where nodes are ordered by a given priority.
- Uniform Cost Search – inherits from the best-first search, but uses the uniform cost evaluator to define the priority of each node.
- A* Search - inherits from the previously created informed search class and uses the A* evaluator.
- Procura Sôfrega – inherits from the informed search class and uses the sof evaluator.

For informed searches, it was necessary to create an abstract class that represents the heuristics that can be used in this type of searches.

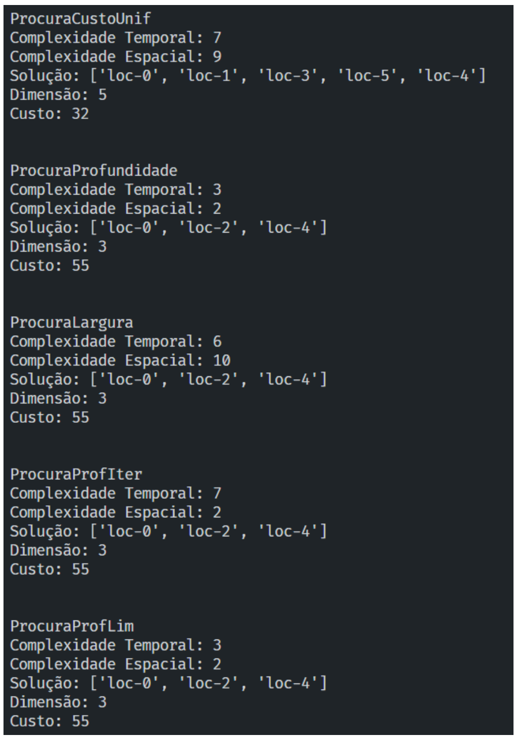
The plan_traj package is where the problem becomes more specific. A planner_trajeto class is created that allows planning a route, generating a solution, receiving all connections, the starting location and the ending location. In this package, the Connection class is also created, which represents the route from one location to another, this is done by defining an origin, a destination and a connection cost. The problema_plan_traj class is created, which is responsible for defining a route planning problem based on the list of connections, the starting location and the ending location. Finally, a locality representation state and an operator representing the making of a connection are also created.

Regarding the PEE planner, it is implemented with a model of the world, which contains the targets, also receives the representation of the world in its problem and receives the final state. This can be used as a heuristic. Your objective is to collect all the targets.

3. 3 .2 Tests

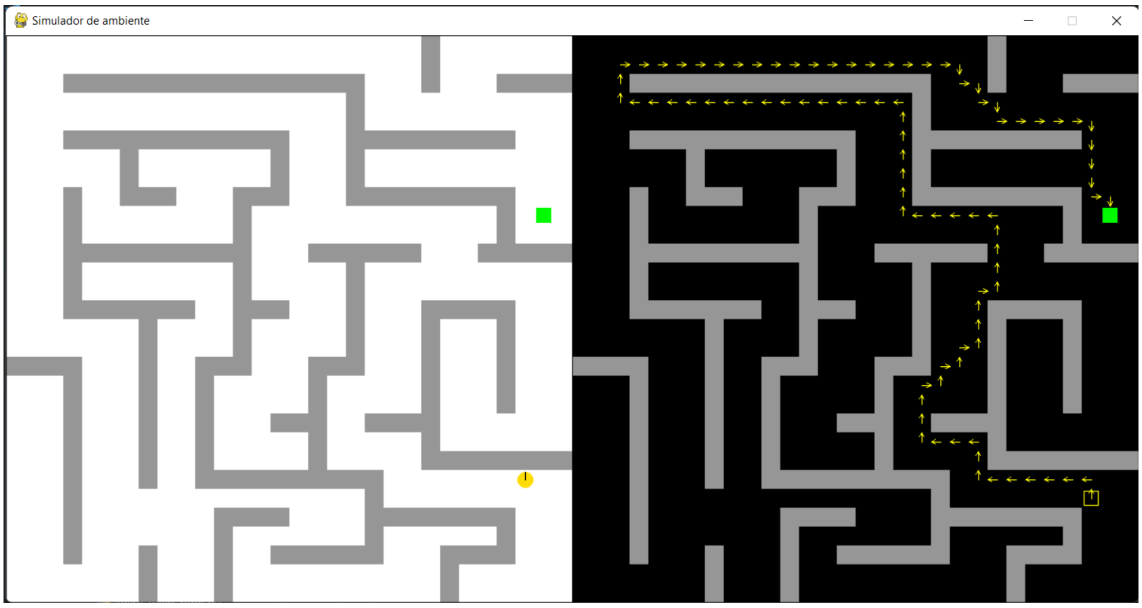
The objective of this test was to observe the difference between the search engines created. All search engines other than uniform cost search found the same solution, however different resources were used between them, whether temporal or spatial. Due to the small size of the problem, this is not the best case to draw 100% certain conclusions, however it is already possible to observe the main differences between search methods.

In this case the uniform cost search was the only one that found a solution different from the others, however it also consumed a lot of resources. We can also see that although the searches in width and uniform cost are optimal and complete, they return different results, this happens because while in width the optimum has to do with the smallest size, in the uniform cost search the optimum has to do with the smallest cost.



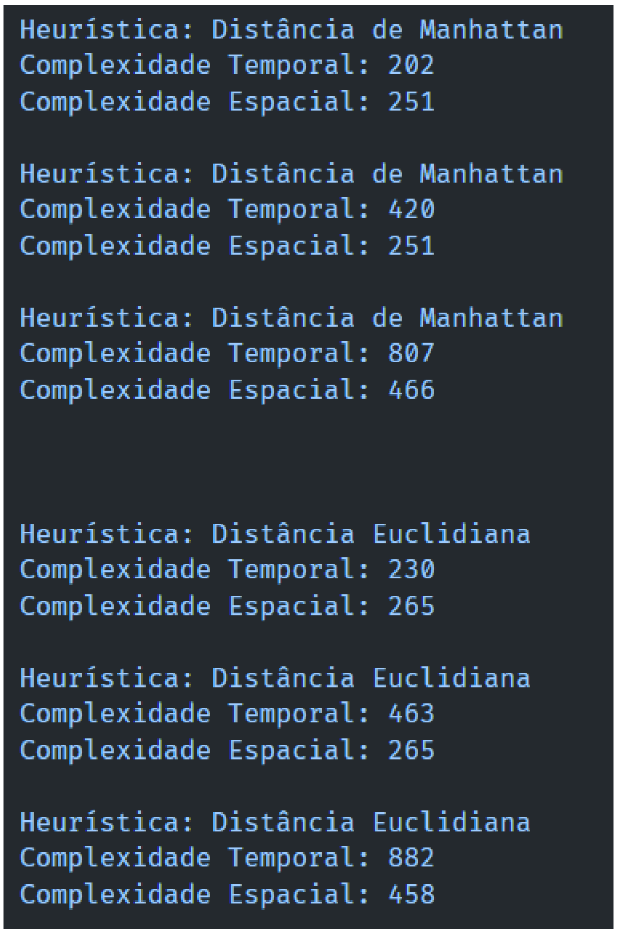
```
ProcuraCustoUnif
Complexidade Temporal: 7
Complexidade Espacial: 9
Solução: ['loc-0', 'loc-1', 'loc-3', 'loc-5', 'loc-4']
Dimensão: 5
Custo: 32

ProcuraProfundidade
Complexidade Temporal: 3
Complexidade Espacial: 2
Solução: ['loc-0', 'loc-2', 'loc-4']
Dimensão: 3
Custo: 55

ProcuraLargura
Complexidade Temporal: 6
Complexidade Espacial: 10
Solução: ['loc-0', 'loc-2', 'loc-4']
Dimensão: 3
Custo: 55

ProcuraProfIter
Complexidade Temporal: 7
Complexidade Espacial: 2
Solução: ['loc-0', 'loc-2', 'loc-4']
Dimensão: 3
Custo: 55

ProcuraProfLim
Complexidade Temporal: 3
Complexidade Espacial: 2
Solução: ['loc-0', 'loc-2', 'loc-4']
Dimensão: 3
Custo: 55
```

For the test in graphical mode, the A* search engine was used, and two heuristics were used, Euclidean and Manhattan.



In both heuristics, the agent was able to complete the objective without any problems. However, if we look at the results for the complexities of both heuristics we can draw some conclusions.

When analyzing the results it is visible that the results of the two Heuristics are very similar, however in all results the Manhattan distance heuristic achieved more efficient results, that is, lower temporal and spatial complexities. This result was as expected because while in the Euclidean distance heuristic we only remove one restriction, in the Manhattan distance heuristic we remove two restrictions, which means that in the Manhattan heuristic the results of estimating the cost of the route from the node in question to the objective are closer to reality, thus obtaining faster results and using less memory.



```
Heurística: Distância de Manhattan
Complexidade Temporal: 202
Complexidade Espacial: 251

Heurística: Distância de Manhattan
Complexidade Temporal: 420
Complexidade Espacial: 251

Heurística: Distância de Manhattan
Complexidade Temporal: 807
Complexidade Espacial: 466



Heurística: Distância Euclidiana
Complexidade Temporal: 230
Complexidade Espacial: 265

Heurística: Distância Euclidiana
Complexidade Temporal: 463
Complexidade Espacial: 265

Heurística: Distância Euclidiana
Complexidade Temporal: 882
Complexidade Espacial: 458
```

3.4 Markov Decision Process (PDM)

This project aimed to implement a deliberative agent through the Markov decision process.

3. 4 .1 Project Development

As in previous projects, this one was also divided into packages. For this project it was only necessary to create two packages, because for this project we used the deliberative control created previously.

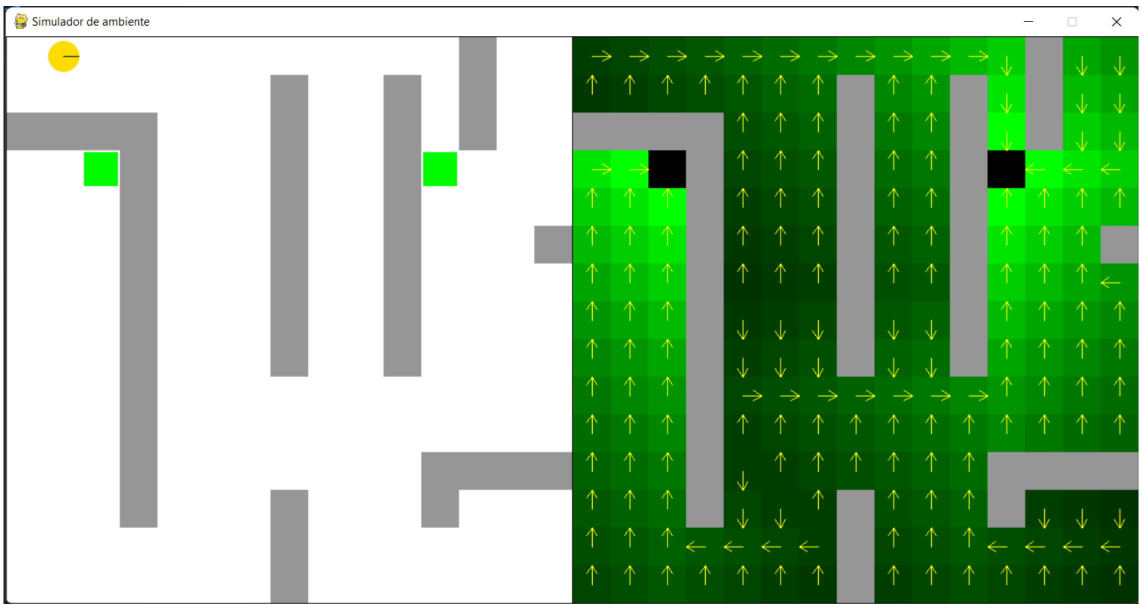
The two packages are then:

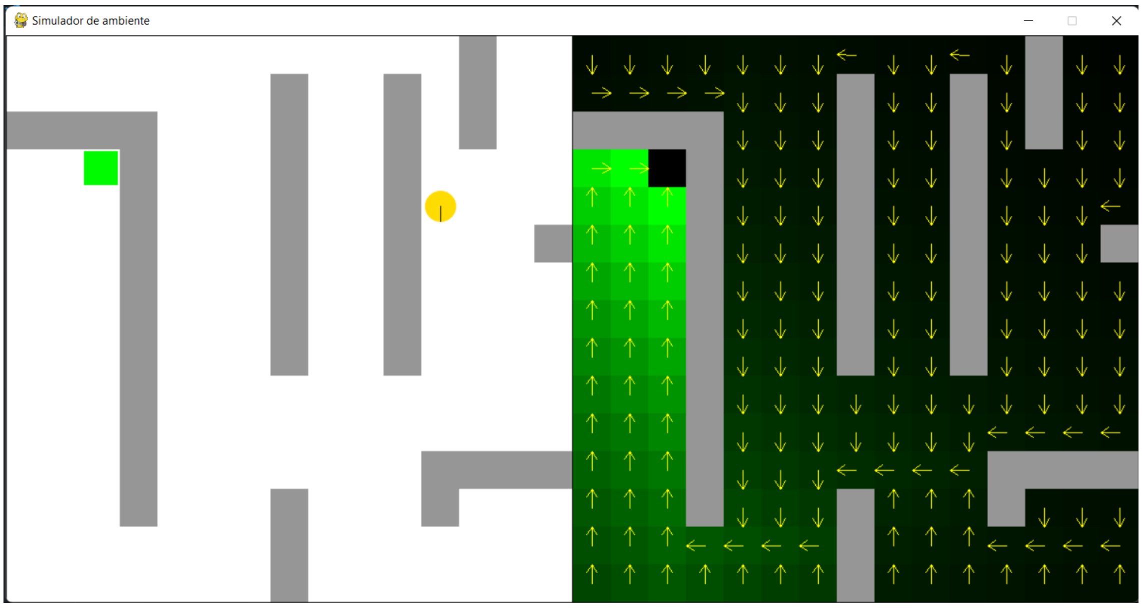
- PDM – package that contains the PDM, MecUtil and ModeloPDM classes.
- Plan_PDM – this package contains the planner and the plan for this process.

As stated in point 2.8, PDM is a mathematical model used to study sequential decision-making problems in stochastics.

We start by creating the PDM, MecUtil and ModeloPDM classes that together allow us to define essential components such as utility, policy and range. Finally, we implemented planner_PDM, which uses the created classes and solves the problem.

3. 4 .2 Tests





As we can see in the tests, the project works as intended.

In the images above, the green squares show the utility, and the yellow arrows show the policy.

An advantage of this process is that wherever the agent is, he always knows which path he must take, this becomes obvious when we look at the behavioral policy obtained. We also see that the utility is greater the closer the agent is to the goal. All of this is in line with the characteristics of the Markov decision process.

4. Review of the Completed Project

Throughout the project deliveries, there was only one project where I was unable to deliver the test working. This delivery failure was due to a syntax error in one of the classes, but during the week following delivery I was able to debug the code and correct the error that prevented the test from running. These syntax errors were recurrent throughout the semester, causing some problems and errors throughout the classes, but in all submissions except for the one mentioned above I was able to resolve these errors by the time the project was submitted. Most of these errors happened due to the spelling agreement, in the teacher's pdf's the words still do not have the spelling agreement and during classes sometimes when I wrote more quickly, I ended up writing the words with the new agreement, and this duality generated some errors. These errors were even more difficult to find because Python is a language that does not do type checking.

The other project that I was unable to deliver correctly happened because it took me a little longer to understand the theoretical part of the project, which took away the time needed to finish the project on time. Even though everything I had done until the moment of delivery was correct, I was unable to deliver the complete project.

In all other deliveries I managed to deliver the project 100% functional, however in some of them I was unable to deliver the complete documentation, I was fully aware of this, but I decided to prioritize the code part and get the project working and leave the documentation for last. . However, I was always able to deliver both the project and the documentation 100% in the next delivery.

5. Conclusion

I therefore conclude that the UC's objectives were successfully achieved. All the concepts given in this course are very important for anyone in the field of software engineering or artificial intelligence. With the evolution of technologies, the complexity of your projects also increases significantly, and this chair also allowed me to know how and when to reduce complexity without losing any legibility.

I realized that artificial intelligence is more present in our lives than I ever thought, whether in household appliances, in our cell phones, in cars themselves, etc.

On a personal note, I think there has been a remarkable evolution in myself in terms of both the clarity of my programming and documentation. This UC allowed me to know better and learn to react better to situations where we have little time to do a certain task, namely in the small tests that the teacher proposed at the end of some classes. I believe that everything I learned in this course will be useful both for the rest of my degree and for my professional life.

In short, I feel satisfied with what I achieved throughout the semester and with the knowledge I acquired.