**Dylan Bermudez Cardona – Luis Charria Meneses – Víctor Garzón Meneses**

**ENGINEERING METHOD – ARTEMIS CREW**

* ***STEP 1: Identification of the problem.***

The Artemis crew needs a computing solution that provides them the information that they need to make a space travel as efficient as possible. This trip is carried out to find a planet that is the most like Earth that can host human life. For example, among the data they require is the shortest path to go from one planet to another consuming the least amount of fuel possible. The following are the requirements that the software must fulfill.

|  |  |
| --- | --- |
| Client | NASA |
| User | * Artemis Crew |
| Functional requirements | * RF1: Update the planet that Artemis is on. * RF2: Consult the information of a planet. * RF3: Show if Artemis can go to a planet departing from the planet that they select. * RF4: Change the planets Artemis previously visited as visited. * RF5: Show the path that consumes less fuel departing from the planet that Artemis is on and going to any other planet. * RF6: Show the actual fuel of the spaceship. |
| Problem context | On each planet there is only one possible direction of travel to another planet, likewise, each trip from one planet to another has a different fuel cost. In turn, Artemis cannot return to a previously visited planet, for reasons of saving resources. It is important to know that there are only 50 planets. |
| Non-functional requirements | * RNF1: The software must be as efficient as possible. * RNF2: The software must be secure enough. |

* **RF1: Update the planet that Artemis is on.**

The system must allow to the Artemis crew know in which planet they are on, therefore, the software must show the name of the current planet.

* **RF2: Consult the information of a planet.**

The system must allow to the Artemis crew know the information of any planet. The information of a planet includes its name and if it has been visited by Artemis or not.

* **RF3: Show the planets where Artemis can go, departing from the planet that they select.**

The software must allow the crew to know which planets they can go to from the planet they select. This indicates that they can know which planets they can travel to from the planet they are currently on or from a planet they want selecting it.

* **RF4: Change the planets Artemis previously visited as visited.**

El sistema debe actualizar el estado de cada planeta que fue visitado previamente por la tripulación como visitado y, entonces, Artemis no puede volver a visitar este planeta.

* **RF5: Show the path that consumes less fuel departing from the planet that Artemis is on and going to any other planet.**

The software must allow the crew to consult the path that consumes less fuel, departing from the planet where it is and arriving at any other planet they select. The software must show the amount of fuel that is necessary to make the trip.

* **RF6: Show the actual fuel of the spaceship.**

The software must allow to Artemis know how many fuel has in an specific moment, whenever the crew wants to know.

* ***STEP 2: Information recompilation.***

To be clear about the problem and the concepts that compose it, we recompile as much information as possible that is contained in the problem and that can provide a solution.

* **Path:**

Is the course or direction in which a person or thing is moving. A route, course, or track along which something moves.

* **Planet:**

Is a celestial body distinguished from the fixed stars by having an apparent motion of its own (including the moon and sun), especially with reference to its supposed influence on people and events.

* **Spaceship:**

Is a vehicle used for space travel.

* **Space crew:**

Is a group of people who work on closely and operate a spaceship.

* **Map:**

Is a diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc.

* **Graph:**

Is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called link or line).

* **Fuel:**

Is a material such as coal, gas, or oil that is burned to produce heat or power.

* **Generics**

Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using generics, it’s possible to create classes that work with different data types.

* **Graph**

A graph in the field of computer science is an abstract type of data, consisting of a set of nodes and a set of arcs that establish relationships between the nodes. The concept of TAD graph descends directly from the mathematical concept of graph

* **Java FX**

JavaFX is a family of products and technologies from Oracle Corporation, for the creation of Rich Internet Applications, that is, web applications that have the features and capabilities of desktop applications, including interactive multimedia applications.

**Sources:**

* + Oxford Languages (<https://languages.oup.com/google-dictionary-en/>)
  + Dictionary.com (<https://www.dictionary.com/>)
  + Merriam-Webster (<https://www.merriam-webster.com/dictionary>)
  + Wikipedia (<https://en.wikipedia.org/>)
* ***STEP 3: Search for creative solutions***

The technique that we used was brainstorm of ideas (each of us generate spontaneous ideas) and reviewing lists (examine different points and possibilities of design). We know that the problem consists in simulate a map of the 50 planets with their respective paths and the spaceship of the crew. We are going to take some data structures for each general requirement, that requires it, that can solve the problem.

* **Update and see the planet that Artemis is on. Visualization of the program.**

We know that the program is going to be created in Java, so we can search some options to find one that can represent the problem.

* + *JavaFX:* We can use JavaFX to simulate the spaceship of the crew and update it while it travels. It can work adding a canvas and painting constantly the map with the planets, the paths and the spaceship.
  + *JOption:* JOption is a library that afford us to make dialog windows or input windows. It can be used to show the information of the spaceship and planets.
* **Simulate the paths, the planets and their respective information and attributes.**
  + *Linked list:* It’s a collection of nodes where every node has a reference of the node that it’s next to it. It can be represented using a head (first node) and a tail (last node). Then, applied to the problem, we can create some linked lists that represents the planets that are united by a path. Each planet is a node, the first planet is the head and the last is the tail. If a node has a reference of the next node, then they are united by a path and the consume of fuel is represented by a ‘double’ type number.
  + *Double linked list:* It’s a collection of nodes where every node has a reference of the node that it’s next to it and has a reference of the node that it’s before it. It can be represented using a head (first node) and a tail (last node). Then, applied to the problem, we can create some double linked lists that represents the planets that are united by a path. Each planet is a node, the first planet is the head and the last is the tail. If a node has a reference of the next node, then they are united by a path in that direction, if a node has a reference of the previous node, then they are united by a path in that direction, and the consume of fuel is represented by a ‘double’ type number.
  + *Hash table:* It’s a collection composed by nodes that has a value and a unique key. It can be represented by an array. If two keys generate a hash pointing to the same index, the corresponding records can be stored in a linked list. It can be modeled with an array or a linked list. Applied to the problem we can create an array composed by a node, each node is composed by 3 elements: planet, planet that are connected to it and the consume of fuel (the last planet is the planet of arriving).
  + *Graph:* Is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called link or line). The graph can be simulated using matrixes of adjacency or incidence or lists of adjacency or incidence. So, applying it to the problem, each vertex (node or point) is a planet, the edge is the path and each edge has its own weight (the consume of fuel). Also, in the matrix or list we specify in the respective planet the arriving planet and the weight of that connection (edge).
  + *Array list:* Is a collection of elements that is dynamic, it isn’t necessary at the moment of instantiate it to specify its size. It’s easy to operate, because it’s not necessary to check if the list is null in some index and has a lot of methods like add, remove, size, etc. Applying it to the problem, we can add a n number of arrays, which each array contains the planet and the weight of its edge and in which planet it arrives.
  + *Array:* Is a static collection of elements, it’s necessary to specify its size at the moment of instantiate it. It can take some verifications when we add or remove any element. We can use it according to the problem creating two arrays, one for all the planets and another for all the edges with their weights.
* ***STEP 4: Transition from Ideas to Preliminary Designs***

In this step we discard the ideas that aren’t feasible. First, we discard, for the requirement of visualization, the JOption option, because, despite we can represent the information more visually than by console, we can’t show the map of the planets and spaceship and move the crew at the time they change of planet. Second, in the requirement of simulate the paths, the planets and their respective information and attributes, we discard the Array option, because we can’t easily represent the map with the planets and edges because of the many validations we would’ve to do and because it’s static, if we want to add more planets or edges, we would’ve to change the size of the collection. Third, and last, we discard the Array list option, not exactly because of something bad that it has, but rather because it would be contained in the option of the graph, since it can be represented by matrices or by an array list. So, the final and most optimum option for the visualization requirement, knowing that the program is being to be solved in Java, is JavaFX.

* + *Linked list:* In addition to the previous information of the linked list, the operation of adding an element to the start or to the end of a linked list, doesn’t require a traversal, if you maintain a reference to the start and to the end of the list, its temporal complexity is O(1). The same goes for the operations of removing an element that is in the start or in the end, its temporal complexity is O(1). Anyway, the operations of adding, of removing and of searching an element that isn’t in the start or in the end of the list, take a temporal complexity of O(n), because they require traversal.
  + *Double linked list:* In addition to the previous information of the double linked list, like the linked list, the operations of adding or of removing an element to the start or to the end of a linked list, they don’t require a traversal and their temporal complexity is O(1). Anyway, the linked list, the operations of adding, of removing and of searching an element that isn’t in the start or in the end of the list, take a temporal complexity of O(n), because they require traversal.
  + *Hash table:* In addition to the previous information of the hash table, the temporal complexity of all of its operations is O(1). Also, in the hash table each value must be in a different slot then we need to create a hash function and, because of that, it’s possible that some values got the same slot (collision), so we must create a method that allows to solve these collisions.
  + *Graph:* In addition to the previous information of the graph, there are 4 types of graphs: simple graph, multi-graph, pseudograph and directed graph, all those, can be weighted or not. All those to model any system, in which its vertices have edges in any direction. Also, the temporal complexity of the operations of adding and of removing is O(1), because the implementation of the graph can be with lists (array lists, for example) or with matrixes. The temporal complexity of the operation of searching is O(n). There are a lot of algorithms previously created and used to find the path to go to a vertex that consumes less or to find the path to travel through all the vertices that costs less (supposing that each edge has a different cost).
* ***STEP 5: Evaluation and selection of the best solution***

Criteria:

* + ***Criterion A:******Efficiency –*** *The efficiency is measured by the temporal complexity of the operations of the alternative. The efficiency can be:*

[4] Constant

[3] Logarithmic

[2] linear

[1] Quadratic

* + ***Criterion B: Ease –*** *The ease is evaluated by the time taken to create, for a programmer with basic knowledge in the alternatives, the methods that are necessary to solve the requirement of find the less path (any type of path). The ease can be measure:*

[3] A time of 20 minutes is more than enough.

[2] A time of 40 minutes is more than enough.

[1] A time of 60 minutes is more than enough.

* + ***Criterion C: Flexibility*** *- The flexibility is evaluated by any change in the planets (quantity) or connections (both quantity and direction). The flexibility is measured by the number of modifications in code:*

[2] One method modified.

[1] Two or more methods modified.

We are going to evaluate each possible alternative and the one that gets the best punctuation (the highest), it’s the best alternative for the problem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Alternative** | **Criterion A** | **Criterion B** | **Criterion C** | **Total** |
| **Linked list** | **2** | **3** | **1** | **6** |
| **Double linked list** | **2** | **3** | **1** | **6** |
| **Hash table** | **2** | **2** | **1** | **5** |
| **Graph** | **2** | **3** | **2** | **7** |

**Then, it’s clear that the best alternative to solve the problem and its requirements is: GRAPH.**