Engineering Method
Problem-Solving Application
Fishing Now VideoGame

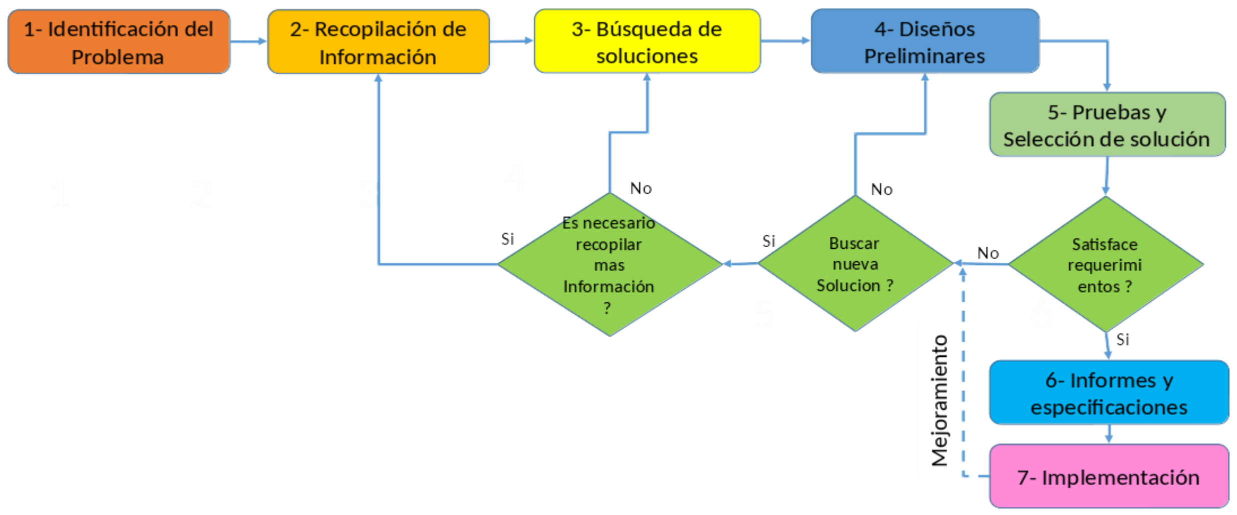
Nathalie Sanchez Trujillo A00405157
Diana María Garzón Toro A00405150

Problem Context

Icesi Games has tasked **TemuDev** with developing an interactive fishing game featuring two distinct maps: one for water fishing and one for lava fishing. Players will explore these maps, find fishing spots, and plan the best strategies to catch fish while managing their energy efficiently.

Solution Development

To solve the situation, the Engineering Method was chosen to develop the solution, following a systematic method aligned with the outlined problem.


1- Identificación del Problema
2- Recopilación de Información
3- Búsqueda de soluciones
4- Diseños Preliminares
5- Pruebas y Selección de solución
Satisface requerimientos ?
No
Si
Buscar nueva Solucion ?
No
Si
Es necesario recopilar mas Información ?
No
Si
6- Informes y especificaciones
7- Implementación
Mejoramiento

Step 1
Problem Identification

Identification needs and symptoms

- Players need to navigate the fishing map and locate optimal fishing spots.
- The problem requires two distinct scenarios: **water fishing** and **lava fishing**, each with unique map conditions and challenges.
- The solution must allow players to calculate and optimize routes based on the specific conditions of each scenario
- The solution must provide logic to determine the most efficient paths for lava fishing, considering accessibility and energy costs.

Problem Definition

Icesi Games requires the development of a videogame about fishing, with two different scenarios: water fishing and lava fishing. The game should let players explore maps, find fishing spots, and plan the best routes to succeed in each scenario.

Step 2
Information Gathering

Definitions

https://www.shiksha.com/online-courses/articles/graphs-in-data-structure-types-representation-operations/

https://www.programiz.com/dsa/graph-adjacency-matrix

https://www.programiz.com/dsa/graph-adjacency-list

https://www.programiz.com/dsa/graph-bfs

https://www.programiz.com/dsa/linked-list

https://www.programiz.com/dsa/graph-dfs

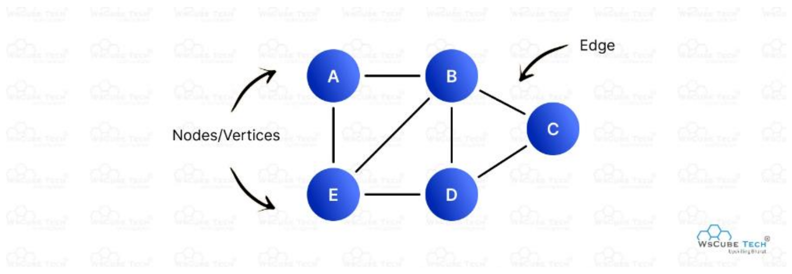
https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php

https://www.programiz.com/dsa/floyd-warshall-algorithm

<u>Structures</u>

Graph

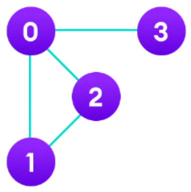
Graphs are non-linear data structures comprising a set of nodes (or vertices), connected by edges (or arcs). Nodes are entities where the data is stored, and their relationships are expressed using edges. Edges may be directed or undirected. Graphs easily demonstrate complicated relationships and are used to solve many real-life problems.



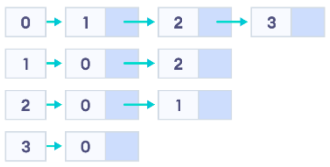
Adjacency List

An adjacency list represents a graph as an array of linked lists. The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

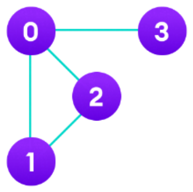
For example, we have a graph below.



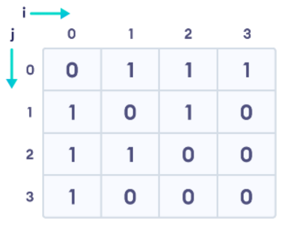
We can represent this graph in the form of a linked list on a computer as shown below.



Adjacency Matrix
An adjacency matrix is a way of representing a graph as a matrix of booleans (0's and 1's). A finite graph can be represented in the form of a square matrix on a computer, where the boolean value of the matrix indicates if there is a direct path between two vertices.
For example, we have a graph below.

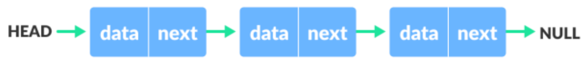


We can represent this graph in matrix form like below.



j \ i	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Linked List
A linked list is a linear collection of elements called nodes. The order between them is set by pointers; addresses or references to other nodes.



Traversal Algorithms

BFS (Breadth-First Search)

Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.
A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty

Depth First Search (DFS)

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.
A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path from one vertex to all other vertices.
It does so by repeatedly selecting the nearest unvisited vertex and calculating the distance to all the unvisited neighboring vertices.

Floyd-Warshall Algorithm
Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs.It works by means of keeping a matrix of distances between each pair of vertices and updating this matrix iteratively till the shortest paths are discovered.

Step 3
Finding Creative Solutions

For The Structure:

Method using Graphs (Adjacency List or Matrix)

The fishing game's map can be naturally modeled as a **graph** because the locations (fishing points) are **connected** to each other through paths (edges), and players need to navigate between these locations. A graph allows a clear representation of the relationships between nodes (fishing spots) and their connections (routes between spots). This makes it easy to model and simulate the game's world, where players explore and move between fishing points. In water mode, where players can move freely between fishing spots, a graph can easily accommodate this by having bidirectional edges between nodes. In lava mode, where movement consumes energy and is limited by accessibility (some paths might be blocked or require extra energy), a graph allows the addition of weighted edges to represent different energy costs or accessibility.

For the Traversal Algorithms

For Lake:

Alternative 1. BFS for Lake
In the fishing game's Lake Mode, players can move freely between fishing points without any restrictions such as energy consumption.
BFS explores nodes level by level, ensuring that the first time a node is reached, it is through the shortest possible path.
Since there are no weights to consider (like in Lava Mode), BFS guarantees that players will find the most efficient route in terms of distance or number of moves between fishing points

Alternative 2. DFS for Lake
DFS allows to explore all accessible fishing spots without worrying about route optimization. We would use recursion to go deeper into each available route until we reach a point with no more connections. It allows for in-depth exploration of the map, as players can visit all fishing spots even if they're far apart, without worrying about finding the most optimal or shortest path.

Alternative 3. Dijsktra for Lake
Dijkstra's algorithm calculates the shortest path between fishing points by evaluating the distance from the starting point to all other points.
A priority queue is used to select the next point to explore based on the smallest cumulative distance.
Although Lake Mode assumes free movement without energy costs, the graph can assign equal weights (e.g., 1) to each connection, allowing Dijkstra's algorithm to still determine the shortest path in terms of hops.
Dijkstra can provide the shortest paths to all points from a given starting location, enabling players to plan efficient movements across the map

Alternative 4. Floyd – Warshall for Lake
Floyd-Warshall computes the shortest paths between all pairs of fishing points, creating a complete distance matrix for the map.
The algorithm requires a matrix representation of the graph, where each cell represents the distance between two points. In Lake Mode, weights can be uniform (e.g., 1) to reflect equal costs for moving between points.
Once the matrix is calculated, players can query the shortest path between any two fishing points instantly, enhancing performance during gameplay

For Lava:

Alternative 1. Dijsktra for Lava
Dijkstra calculates the most energy-efficient path between the player's current position and other fishing points by considering energy costs as edge weights.
A priority queue ensures that the next point explored is always the one with the lowest cumulative energy cost, optimizing pathfinding.
Dijkstra works well for Lava Mode as it manages energy use and finds the best paths in weighted graphs.

Alternative 2. Floyd – Warshall for Lava
Floyd-Warshall computes the shortest paths between all pairs of fishing points, creating a complete The game map is represented as a weighted adjacency matrix, where each cell indicates the energy cost to move between two fishing points.
The algorithm iteratively updates the matrix by evaluating whether passing through an intermediate point results in a lower energy cost for reaching a destination.
For Lava Mode, this ensures that the most efficient routes, in terms of energy, are always ready to be used, even when players are jumping between multiple locations.

BFS and DFS are not suitable for Lava Mode because they do not account for energy costs on weighted graphs.

Step 4.
Transition from Ideas to Preliminary Designs

For Lake:

The first thing we do in this step is to discard ideas that are not feasible. In this sense , we discard ***the***

Alternative 2 and 3 (Dijsktra and Floyd – Warshall) because they are more suitable for the Lava mode, where energy costs must be considered for each move. Since Lake mode has no such constraints, these algorithms would be unnecessary and inefficient for this part of the game.
Using Dijkstra here would be an over-complication, as the algorithm would perform unnecessary calculations that are irrelevant to the game mechanics in Lake mode. Essentially, it would be computationally inefficient for the problem at hand. Given that the graph in Lake mode is unweighted and paths do not involve energy consumption or other costs, the Floyd-Warshall algorithm would be overkill also..

Review of the other alternatives:

Alternative 2. DFS

- **Unnecessary deep exploration**: DFS explores deeply into one branch before backtracking, which means it can end up exploring many unnecessary paths before finding the correct route.
- Since DFS doesn't focus on nearby nodes first, it could waste time exploring distant parts of the map before finding the target.

Alternative 1. BFS

- It finds the shortest path to a point, which helps players reach their destination faster without unnecessary steps.
- In Lake mode, where there's no need to worry about energy, BFS works well to find the best route in an easy way.
- Since players in Lake Mode can move freely between fishing points, the game's movement model is essentially about traversing through connected nodes. BFS fits naturally into this movement logic because it explores all possible paths from the current position and ensures that the closest or shortest fishing points are reached first.

For Lava:

The first thing we do in this step is to discard ideas that are not feasible. In this sense , we discard ***the***

Alternative 1 (Dijsktra) because **Floyd-Warshall** is better suited for Lava Mode due to the following reasons:

- Floyd-Warshall efficiently handles graphs with weights (like energy consumption), which is essential in Lava Mode where energy plays a crucial role in the player's movement between points and Dijkstra works from a single starting point to others, but **Floyd-Warshall** finds the shortest paths between all points on the map. Since Lava Mode requires checking many paths frequently, it's better to have all paths precomputed.

Step 5. Evaluation and Selection of the Best Solution

Criteria

Criteria must be defined to evaluate the solution alternatives, and based on the results, choose the solution that best satisfies the needs of the problem.The criteria we selected in this case are listed below. Next to each one, a numerical value has been assigned to indicate a weight, showing which of the possible values for each criterion have more weight.

Criterion A.

Ease of Managing the Scenario

How well does the algorithm handle the rules of the game?

Points:

[3] Handle game conditions without complications.

[2] Can be adjusted with moderate effort.

[1] Not ideal for this scenario.

Criterion B.

Evaluation Based on Ease:

Points:

[3] Excellent: Guarantees the best result according to the objective (optimal routes, lowest cost).

[2] Good: Accurate enough for the problem, but not always optimal.

[1] Poor: May not be adequate to ensure optimal solutions.

Criterion C.

Algorithm Robustness

[3] Excellent: Efficiently handles complex graphs with many connections, isolated nodes or loops.

[2] Good: Works well in most cases, but may fail in very complex graph structures.

[1] Bad: Has significant limitations when handling graphs with atypical structures.

Lake:

Evaluation
By evaluating the above criteria for the remaining alternatives, we obtain the following table:

	Criterion A	Criterion B	Criterion C	Total
Alternative 1 BFS	3	3	3	9
Alternative 2 DFS	2	2	2	6
Alternative 3 Dijsktra	1	2	3	6
Alternative 4 Floyd – Warshall	1	1	2	4

Selection
According to the previous evaluation, Alternative 1 should be selected, since it obtained the highest score according to the defined criteria.

Lava:

Evaluation

By evaluating the above criteria for the remaining alternatives, we obtain the following table:

	Criterion A	**Criterion B**	**Criterion C**	**Total**
Alternative 1 Dijsktra	3	2	3	8
Alternative 2 Floyd – Warshall	3	3	3	9

Selection

According to the previous evaluation, Alternative 2 should be selected, since it obtained the highest score according to the defined criteria.

Step 6. Preparation of Reports and Specifications

Problem Specification (in terms of input/output)

Problem > creating an interactive fishing game with two modes, water and lava, where players need to efficiently navigate between fishing points while managing energy in lava mode.

Input

RF1 Create Game Map

- Location
- Conection

RF2: Create Two Game Modes

- Game Mode (Buttons)

RF3 Hook Movement

- Motion Direction

RF4 Movement of the Fish

- Current Position of the Fish

RF5 Completion of the Game

- Player Energy
- Distance Between Nodes

Outputs:

RF1 Create Game Map

Visual Representation of two maps, and the menu.

RF2: Create Two Game Modes

Visual Change to the game screen lake or lava

RF3 Hook Movement

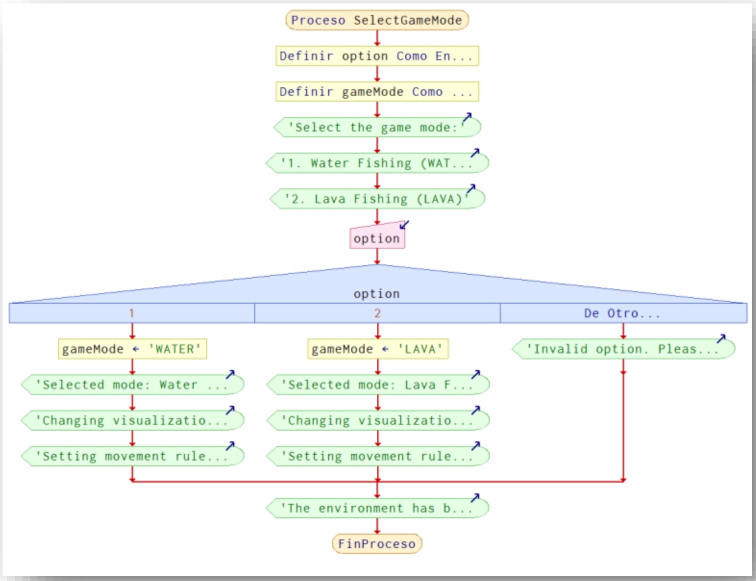
Updated hook position.

RF4 Movement of the Fish

Updated fish position.

RF5 Completion of the Game

"Game Over" o "You Win".


Proceso SelectGameMode
Definir option Como En...
Definir gameMode Como ...
'Select the game mode:'
'1. Water Fishing (WAT...
'2. Lava Fishing (LAVA)'
option
option
1
2
De Otro...
gameMode ← 'WATER'
gameMode ← 'LAVA'
'Invalid option. Pleas...
'Selected mode: Water ...
'Selected mode: Lava F...
'Changing visualizatio...
'Changing visualizatio...
'Setting movement rule...
'Setting movement rule...
'The environment has b...
FinProceso

```
Process SelectGameMode
    Define option As Integer
    Define gameMode As Text

    Write "Select the game mode:"
    Write "1. Water Fishing (WATER)"
    Write "2. Lava Fishing (LAVA)"
    Read option

    Switch option Do
        Case 1:
            gameMode <- "WATER"
            Write "Selected mode: Water Fishing"
            Write "Changing visualization to the water environment..."
            Write "Setting movement rules with no energy cost."
        Case 2:
            gameMode <- "LAVA"
            Write "Selected mode: Lava Fishing"
            Write "Changing visualization to the lava environment..."
            Write "Setting movement rules with energy cost."
        Default:
            Write "Invalid option. Please try again."
    EndSwitch

    Write "The environment has been configured for the mode: ", gameMode
EndProcess
```

Step 7. Design Implementation

Implementation in a Programming Language. (JAVA) and JAVAFX

List of Tasks to be implemented:
RF1 Create Game Map
RF2: Create Two Game Modes
RF3: Hook Movement
RF4: Movement of the Fish
RF5: Completion of the Game.
RF6: Graphical Interface