Mariana Agudelo Salazar - A00398722
Sharik Camila Rueda Lucero - A00399189
Natalia Vargas Parra - A00398706

ENGINEERING DESIGN METHOD

Phase 1: Problem Identification

- *Problem/Need:*

  The problem or need that led to the development of the Froggish game is the desire to create an engaging and educational gaming experience that introduces players to fundamental graph data structures and algorithms while providing entertainment. Specifically, there is a need for a game that:

  - Combines fun and learning: Our solution has to address this need by offering an enjoyable gaming experience while teaching players about graph structures and algorithms.
  - Promotes critical thinking and decision-making.
  - Offers an interactive and visually appealing interface.

- *Key Functionalities:*

  To address the problem/need mentioned above, the key functionalities of the Froggish game include:

  - Graph Data Structure: The game incorporates a graph data structure to represent the interconnected nodes within each level, allowing players to navigate the frog through the game world.

  - Game Levels: Froggish is divided into three distinct levels(Plant Pot, River, and Garden). Each level presents unique challenges, such as varying numbers of lotus flowers and leaves, creating a structured progression in complexity.

  - Energy Mechanism: Leaves cost a random amount of energy (ranging from 1 to 5) when jumped on, while lotus flowers replenish energy. Players must make strategic decisions to minimise energy consumption and reach the frog's home.

  - Hint System: Players can seek hints at a cost, which increases as they progress through the levels. Hints provide guidance on the best path to navigate through the level, helping players learn and make informed decisions.

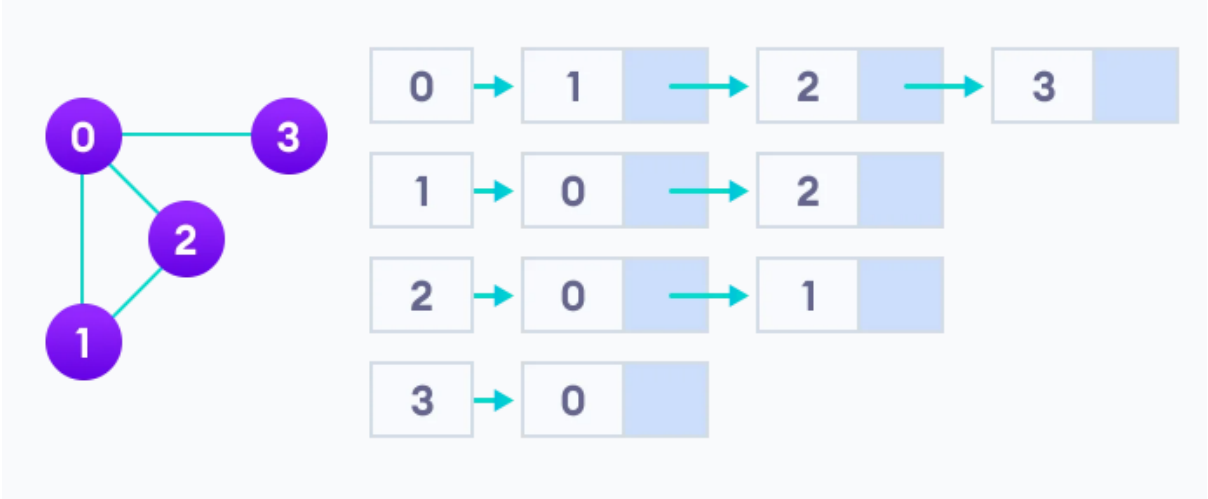
  - Graph Algorithms: The game implements three fundamental graph algorithms - BFS, DFS, and Dijkstra's algorithm. These algorithms are used to compute optimal paths, generate hints, and challenge players with graph-related puzzles.

- Visual Interface: The game is built using JavaFX to provide players with an engaging and user-friendly graphical interface. The interface is designed to be visually appealing and intuitive, enhancing the overall gaming experience.

Phase 2: Research and Gather Necessary Information

- *Graph:* Is a data structure (V, E) that consists of
  - A collection of vertices V
  - A collection of edges E, represented as ordered pairs of vertices (u,v)

On the right side of this image there's a matrix view of the graph on the left, it is called adjacency matrix. Adjacency matrices use 1's and 0's to represent if there's connection between two vertices or not respectively.



On the right side of this image there's an adjacency list that shows how the vertices of

the graph on the left are related to one another.

- *DFS:* Stands for Depth First Search which works with the LIFO (Last In First Out) principle. Traversing in DFS starts at the root node and it then expands each node recursively until it finds the node it is searching for. This algorithm is also useful for finding connected nodes in a graph, solving puzzles within one solution, to find if the graph is strongly connected. It is an edge-based technique.

- *BFS:* Stand for Breadth First Search which mostly implements Queue data structure. When traversing a graph with BFS any node can be considered as a root. It is a vertex-based technique to find the shortest path in a graph.

- Dijkstra: In essence, the Dijkstra's algorithm starts from a selected source node, analyses the graph's structure and available paths. The objective of this is to determine the most efficient route to connect the chosen node with all others within the graph. Throughout the process, Dijkstra's algorithm maintains a record of the shortest known distances from the source node to each node, constantly updating these values when it discovers a more efficient path. Once the algorithm identifies the shortest path connecting the source node to a destination node, it designates the relevant node as 'visited. This algorithm will continue until all reachable nodes are visited.

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

Phase 3: Search of Creative Solutions

Alternative 1. Froggish game of chance

The implemented solution consists of a virtual die, in which the player simulates rolling it and obtains a random number. This number thrown by the die will be the vertex where the frog will go. However, this approach complicates the frog's arrival home, since, if the die does not favour the player, he could lose all his lives. In addition, it implements a condition algorithm that is inefficient, since it would have an excessive amount of validations.

Alternative 2. Froggish game with Floyd-Warshall

The solution implemented to determine the shortest path that allows the frog to reach home, considering the costs associated with each edge, is the use of the Floyd-Warshall algorithm. This algorithm offers a more efficient way to preserve the frog's energy and find the optimal route home. However, the Floyd-Warshall

algorithm uses a matrix to store and update the distances between nodes, performing summation between row and column for each node.This process is repeated for all nodes, having a complexity of O(n^3), which could limit its effectiveness in the program.

Alternative 3.Froggish game with Dijkstra

The solution to determine the shortest path that allows the frog to reach its home, taking into account the costs associated with each edge, is Dijkstra's algorithm. This algorithm is more efficient in larger graphs, avoiding unnecessary computations by focusing on the relevant node and calculating the different paths optimally. When it finds a shortest path, it updates these values in an adjacency matrix. The complexity in the program is very important, Dijkstra has a complexity of O(n^2) and only adapts to positive costs. In addition, the system will employ the BFS or DFS structure to know which flower or lotus leaf the frog is on, providing hints or help to the player when required.

Phase 4: Idea Formulation to Preliminary Design

After an analysis of the alternatives, the first option is discarded, because it does not implement any graph algorithm that indicates which is the shortest path, so this option does not guarantee the efficiency of the route, because it does not take into account the cost of the edges nor in which edge the frog is located, so it cannot help the player by giving advice when needed. Finally, this alternative does not fully guarantee that the frog will reach its home, because it may fall into infinite loops.

The detailed review of the other alternatives leads us to the next one:

Alternative 2. Froggish game with Floyd-Warshall

The program meets the algorithm needed to find the shortest path inefficiently, having a higher time complexity of O(n^3). Now, this solution does not implement any algorithm such as BFS or DFS, so it cannot be placed at the vertex where the frog is currently, therefore, the player could not ask the program for hints.

Alternative 3.Froggish game with Dijkstra

The program complies with the required algorithms, to obtain the shortest path to the frog's house, achieving its goal more efficiently with a lower time complexity which is O(n^2). In addition, to implement algorithms such as BFS or DFS in which we can find the vertex where the frog is currently optimal, so we can give advice to the player and get the next shortest path.

Phase 5: Evaluation and Selection of the Best Solution

According to the problem at hand the evaluation of the remaining alternatives goes as following:

Alternative 2: Create froggish
Pros:

- Uses graph

Cons: Relies on the user's and client's usage, might not be usable for more complex methods or features.

Alternative 3: Create..

Pros:

Cons:

Based on the evaluation, the 2 Alternative is a better solution because it's the most efficient choice as it works closely with the client's requirements.

Phase 6: Reports and Specifications Preparation

Phase 7: Complexity and pseudocode

Bibliography

- *Graph Data Structure*. (n.d.). https://www.programiz.com/dsa/graph
- Murillo, J. (2022, July 18). DFS vs BFS. *Encora*.

  https://www.encora.com/es/blog/dfs-vs-bfs
- *BFS vs DFS | What's the difference? - javatpoint*. (n.d.). www.javatpoint.com.

  https://www.javatpoint.com/bfs-vs-dfs