Engineering Method

Daniel Alejandro Castro Escobar

Katerine Valens Orejuela

Icesi University

Computation and Discrete Structures

November 5, 2023

***First Step: Identification of the problem***

**SOFTWARE ENGINEERING PROBLEM SPECIFICATION TABLE, identifying the following elements**

|  |  |
| --- | --- |
| **CLIENT** | Computation and discrete structures |
| **USER** | Players |
| **FUNCTIONAL REQUIREMENTS** | R1 - Create and show an 8x8 functional board with a randomly placed source and drain.  R2 - Location of the different type of pipes on the board.  R3 - Simulation of the water flow through the pipes in the board.  R4 - Register and show a leaderboard with the scores. |
| **CONTEXT OF THE PROBLEM** | Computation and discrete structures require students to develop a pipe simulation game, which starts with the creation of an 8x8 board with a randomly placed source of water and drain. The user should be able to select the type of pipe they wish to use and place it on the given coordinates on the board. Also, it should allow the user to, whenever they’re ready, simulate the waterflow through the pipes to confirm if their solution is working. Finally, the program is also required to calculate a score for the game and register it on a leaderboard. |
| **NON-FUNCTIONAL REQUIREMENTS** | RN1 – The board must be modelled using graphs.  RN2 – The game must work properly with both types of graphs, being adjacency matrices and adjacency lists.  RN3 – The program requires a binary search tree to show the leaderboard. |

**Functional Requirements Analysis Table**

| Name or identifier | R1 - Create and show an 8x8 functional board with a randomly placed source and drain. | | |
| --- | --- | --- | --- |
| Summary | When a game starts, an 8x8 functional board will be created. In this board, the water source and the drain are randomly located. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
| graphType | String | Button 1 - Adjacency matrix  Button 2 – Adjacency list |
| Result or postcondition | A board designed with the selected type of graph is created. The vertices of the graph contain objects of type Box. These boxes contain the different types of pipes. The representation of the board is an 8x8 square matrix with empty boxes, one source, and one drain’. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| table | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R2 - Location of the pipes on the board. | | |
| Summary | The game must allow to place different pipes on the board depending on the indications of position that the user input (Rows and columns). Then the user chooses the option of pipe to put in that location. The player can repeat this process as much as he needs, until the fountain is linked with the drain. The menu must be presented when the user inserts a new pipe.  The program must not allow the option to change the position of the water source and the drain. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
| Location | String | Row, column |
| pipeType | String | Button 1: Vertical pipe  Button 2: Horizontal pipe  Button 3: Cicular pipe |
| Result or postcondition | The program selects the box in the graph board according to the coordinate provided by the user. Inside the box contained by the node, a pipe of the selected type of pipe is created and placed. The program keeps showing the board and the menu until the user decides to go back to menu or simulate the waterflow. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
|  | board | String | Representation of the updated board in a matrix form. |
|  | menu | String | 1. Place pipe  2. Simulate  3. Go back to menu |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R3 - Simulation of the water in the pipes | | |
| Summary | The program must allow, during a game session, to simulate the waterflow through the pipes from the source to the drain. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | A method starts at the source and looks for the connected vertices to validate the contents of the boxes and verify if the pipes allow the proper flow of water. Returns true if the pipes located allow water to go from the source to the drain, and false if this is not possible. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| allowsFlow | boolean | True = The water flow worked  False = The water flow did not work |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R4 - Register and show scores. | | |
| Summary | The software must register the score of each game in a binary search tree. The scores are calculated with a formula that considers the total time in seconds and the number of pipes used. As the scores are organized from the highest to the lowest, the scores are showed as a leaderboard. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | When this function is called, the program adds all the players to the binary search tree through the method compareTo, and it shows the scores for all the games played for all players until that moment, organized from highest to lowest. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| leaderboard | string | Binary search tree representing the leaderboard according to the scores obtained in the games |

***Second Step: Information Gathering***

**Definitions:**

* **Structure:** A structure, in computer science, is an organized way of storing data. They are very important for making different operations on some stored collections efficiently. For example, we can find some structures in computing such as : arrays, linked lists, trees, stacks, queues and many others. Each of these structures has its own characteristics, it depends on the context of the problem to use their functionalities and advantages to develop a solution.
* **Linked List**: Is a data structure used for organizing collection of nodes. Linked list store elements in the nodes that are connected to each other. Its functionality consists in a value stored in the node and pointers that allow it to create a link between two nodes making a chain starting with a head node. There are two types of list: singly linked list and doubly linked list.
* **Graph**: A graph is a data structure composed of a set of vertices and edges, which form connections between the vertices. There exist many different types of graphs. This structure can be represented using:
* Edge Lists
* Adjacency Lists
* Adjacency Matrix
* Incidence Matrix

The representation that is going to be used to the development of this project are: Adjacency List and Adjacency Matrix

* **DFS**: (Depth first search) It is a search algorithm with the function of making a tour through all the vertices of a graph and searching for information. The technique implemented by the algorithm is to seek the greatest depth and when it reaches it begins to go back.
* **BFS:** (Breadth first search) It is a search algorithm with the function of searching and traversing a graph. It starts at the root and visits neighboring nodes. It is usually most useful when the shortest path between nodes in the network is needed.
* **Dijkstra:** The algorithm of Dijkstra, in graph theory, is intended to find the shortest possible path between two nodes in a graph. It maintains a list of visited nodes and assigns an initial distance to each one. During each iteration, it selects the node with the shortest distance, marks it as visited, and updates the tentative distances to its neighboring nodes.

***Third Step: Search Creative Solutions***

In this step we are going to consider as many util possibilities that we can find to give a solution for the problem to solve. Starting from the objective to develop an application that has all the functionalities, we need to find the best alternative to the software.

* **1- Implementation of a graph:** We can represent the board and the connections implementing a graph. The vertex can be the slots of the board and the edges are the possible ways. Also, based on this structure we can implement some of the searching algorithms for verifying the ways from the source to the drain. This makes this solution very efficient to verify the connectivity and allow us to represent complex relations between the board and the pipes.
* **2- Implement a doubly linked list:** We can represent each node as a slot of the board and the connections as the possible ways. This one is the simplest option because we only need to create methods that allow the insertion of pipes, verification and simulation based on recursivity.
* **3- Implement a bidimensional matrix:** This option makes it possible to create an array of two dimensions (columns and rows) and store the pipes in the positions allowed in the board (slots). We need to develop methods to insert, verify and simulate based on recursivity and double paths in the matrix.

***Fourth Step: Preliminary Designs***

Since we have already discussed and analyzed in detail the 3 previous solutions and described their possible functionalities, we decided to discard the third alternative ***"Implement a two-dimensional matrix"*** since it generates a high time complexity due to the multiple paths that have to be used in the insertion and search methods. In addition, it has a high memory consumption, since it is a predefined 8x8 board.

In this way, we have 2 solutions left:

**1- Implement a graph:**

* We can use efficient algorithms for searching and verify
* The solution allow us to represent complex connections between the nodes
* It is harder to develop this structure in comparison with other solutions
* Allow the application of advanced algorithms that makes the software more complete
* Sometimes can consume more computational resources

**2- Implement a doubly linked list:**

* Its flexible to create the methods to add and delete pipes
* Offers a mostly efficient use of memory
* It is easy to execute validations and methods for flow control.
* The access to the information in the list is slow
* Allows navigation in both directions
* It is adaptable to changes
* Most temporal complexity

***Fifth Step: Selection of the Best Solution***

Based on the previous analysis, we have two possible solutions to the problem. For choosing the best option we need to set different criteria to evaluate the advantages and disadvantages. The criteria are:

* **Criteria A: Program completeness**

[1] Not complete

[2] Partially completed

[3] Very complete

* **Criteria B: Efficiency**

[1] Inefficient

[2] Low efficient

[3] Normal

[4] Efficient

[5] Very efficient

* **Criteria C: Ease of implementation**

[1] Hard to implement

[2] Medium difficulty to implement

[3] Easy to implement

* **Criteria D: Adaptability for the complexity**

[1] Not adaptable

[2] Low adaptability

[2] Medium adaptability

[4] High adaptability

***EVALUATION:*** *Now we need to rate the solutions depending on the previous criteria*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Solution*** | ***Criteria A*** | ***Criteria B*** | ***Criteria C*** | ***Criteria D*** | ***Total*** |
| ***Implement a graph*** | ***3*** | ***5*** | ***2*** | ***4*** | ***14*** |
| ***Implement a doubly linked list*** | ***2*** | ***3*** | ***3*** | ***2*** | ***10*** |

Since we have evaluated both solutions and implemented the criteria by way of evaluation we can conclude from the results that the most optimal solution is the first one, implementing a graph, since this structure allows us to create efficient algorithms for both the search and the insertion of information into the graph. It also gives us flexibility in the long run if the problem increases in difficulty. It also offers greater ease of handling connections.

***Sixth Step: Preparation of Reports and Specifications***

***Seven Step: Implementation***