**Engineering Method**

**Phase 1:**

The development of a graph-based game is proposed, inspired by the game Quoridor, with a minimum of 50 vertices and 50 edges. At least two graph algorithms must be applied during development.

Problem Identification: There is a need to develop a game that implements graphs and their features. To achieve this, the game "Quoridor" is used as a reference, where players compete to reach the other side of the board before their opponents. Each player can place barriers on the board to block the progress of others. The board is square and has a grid design. The game board will be modeled as a graph, where each square is a vertex, and connections between squares represent possible allowed movements.

**Symptoms and Needs:**

* To grasp the theory of graphs, implementation in a game is sought to gain a better understanding of the subject.
* The player needs to be able to move around the board.
* The player must be able to place obstacles or walls.

**Phase 2:**

* **Graphs:** Graphs are mathematical structures that represent relationships between objects. An object is represented as a vertex, and a relationship between two objects is represented as an edge. Graphs can be used to model a variety of situations, such as social networks, transportation systems, or biological systems. They are a powerful tool that can be used to solve a variety of problems, such as finding the shortest path between two points or identifying communities in a social network.
* **Floyd-Warshall:** This algorithm uses dynamic programming to find the shortest path between all pairs of nodes in a directed graph with positive or negative weights, but without negative cycles. The time complexity of this algorithm is Θ(|V|^3), and the space complexity is Θ(|V|^2).
* **Dijkstra:** This algorithm uses a priority queue to find the shortest path from a source node to all other nodes in a directed graph with non-negative weights. The time complexity of this algorithm is O(|E| + |V| log |V|), and the space complexity is O(|V|).
* **BFS:** This algorithm uses a queue to find the shortest path from a source node to all other nodes in an unweighted graph. The time complexity of this algorithm is O(|E| + |V|), and the space complexity is O(|V|).
* **DFS:** This algorithm traverses all nodes of a graph, visiting the adjacent nodes of each node in depth before backtracking. The time complexity of this algorithm is O(|E| + |V|), and the space complexity is O(|V|).
* **Prim:** This algorithm uses a priority queue to find the minimum spanning tree of an undirected weighted graph. The time complexity of this algorithm is O(|E| log |V|), and the space complexity is O(|V|).
* **Kruskal:** This algorithm uses a disjoint-set data structure to find the minimum spanning tree of an undirected weighted graph. The time complexity of this algorithm is O(|E| log |E|), and the space complexity is O(|V|).
* **Hashmap** is a data structure in which pairs of values are stored, where the key is a unique value used to access the corresponding value. Hashmap is an unordered data structure, meaning that elements are not stored in a specific order. It works by using a hash function to convert the key into an index in a hash table. The hash function is a mathematical function that ensures each key becomes a unique index. The corresponding value is stored in the hash table at the corresponding index.

**Fase 3:**

En la implementación nos vamos a centrar en las posibles soluciones que nos permiten encontrar caminos.

* **Find the fastest path**
* BFS
* DFS
* Dijkstra
* Floyd
* Prim
* **Find path**
* BFS
* DFS
* Dijkstra
* Floyd
* Prim

**Phase 4:**

* **Find the fastest path**
  + **Depth-First Search (DFS):** DFS is not suitable for finding the shortest path as it does not guarantee to find the shortest path between two nodes. DFS might discover a longer path before finding the shortest one.
  + **Prim's Algorithm:** Prim is used to find a minimum spanning tree, which is different from finding the shortest path between two nodes. A minimum spanning tree aims to connect all nodes in the graph with the minimum total weight but does not necessarily find the shortest path between specific nodes.

**Fase 5:**

* **Find the fastest path**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Efficiency** | **Usability** | **Maintainable** | **Scalability** | **Total** |
| **BFS** | **2** | **5** | **4** | **4** | **15** |
| **Dijkstra** | **5** | **4** | **4** | **3** | **16** |
| **Floyd** | **2** | **3** | **3** | **2** | **10** |

* **Find path**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Efficiency** | **Usability** | **Maintainable** | **Scalability** | **Total** |
| **BFS** | **3** | **4** | **4** | **4** | **15** |
| **Dijkstra** | **3** | **4** | **3** | **3** | **13** |
| **Floyd** | **2** | **3** | **2** | **2** | **9** |
| **DFS** | **3** | **5** | **4** | **4** | **16** |
| **Prim** | **2** | **4** | **3** | **2** | **11** |

Teniendo en cuenta los criterios de evaluación, podemos concluir lo siguiente:

* **Find the fastest path: Dijkstra's** Algorithm stands out as an excellent choice for finding faster paths in weighted graphs due to its efficiency, usability, and maintainability. Its specific focus on distance-based node selection ensures an effective search for shorter paths, making it an efficient and easily implementable choice across various applications. However, in terms of scalability, Dijkstra's efficiency may diminish in extremely large graphs, particularly due to the need to maintain a priority queue. Overall, considering the criteria of efficiency, usability, maintenance, and scalability, Dijkstra's Algorithm remains a robust option for finding faster paths in weighted graphs.
* **Find Path:** emerges as a strong choice for the task of verifying the existence of paths in a graph, addressing various evaluation criteria. Its usability stands out as it is an algorithm that is easy to implement and understand, simplifying its integration into various applications. The simplicity of its logic contributes to efficient maintenance, allowing for adjustments and corrections to be made easily. Although its efficiency may vary depending on the graph's structure, DFS demonstrates acceptable scalability for moderately large graphs. In summary, DFS meets the criteria of usability, maintenance, and scalability, making it a suitable choice for the task of verifying the existence of paths in contexts where the length of the path is not the primary consideration.