*Group : David Alejandro Troya, Joshua Rivera, Juan Amorocho*

*Phase 1. appropriate description of the problematic context - identifying causes and symptoms - (2) identification and concrete and unambiguous definition of the problem (3) specification of the functional requirements associated with the needs raised in the statement*

For this task we are asked to develop a game for one or more players that can be modeled using graphs with the following characteristics: at least 50 vertices and 50 edges. For the graphs already mentioned, it is necessary to implement at least two of the algorithms seen in class that allow for a solution. Some of them are: Routes over Graphs (BFS, DFS), Minimum Weight Paths (Dijkstra, Floyd-Warshall), Minimum Covering Tree -MST- (Prim, Kruskal).

In this way, we have decided to make a labyrinth or treasure hunt type game where arrival stations will be located in significant parts (start, end, corners) of the map, these stations will have a cost of points for the player, if the player can discover by himself the path to the treasure is the winner, otherwise, the built-in Dijkstra, Floyd-Warshall algorithms in the program will discover the most convenient and economical path for the player.

In order for our game mentioned above to be possible, we have identified the following requirements or needs that it should meet:

**Visit Nodes:** The player must have the ability to access a station represented as a node in exchange for a price of points.

**Intuitive User Interface:** The use of a user-friendly interface. It should also give a brief explanation of the game and the option to use the help of the algorithms at any time.

**Needs and Symptoms Identified:**

**Error handling:** If the player tries to make wrong moves Ex: Choosing an arrival station not adjacent or not connected to his previous station. In the game these should be handled as alerts and given a new opportunity to make your move

**Problem Definition:**

The main problem we found in carrying out this task was making it possible for a person (User) to use the algorithms to solve the game. In any case, the game we model has the following

**Functional Requirements:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier and name** | *[RF1-****Creation of a Graph****]* | | | |
| **Summary** | * The system must allow the creation of a graph with a minimum of 50 vertices and 50 edges, which will allow the game to be modeled using this data structure. This requirement is essential to meet the project criteria and ensure that the game has the required complexity in terms of vertices and edges. | | | |
| **Entries** | **Name** | **Type** | | **Condition** |
| vortex, edges | Int | | “The system must allow the user to define the data structure that will represent the network. This input is essential to determine how vertices and edges will be stored and manipulated in the game.” |
| **Result or Postcondition** | After the user specifies the minimum number of vertices and edges, and the data structure to represent the network, the system will create a valid network that meets the requirements of at least 50 vertices and 50 edges. | | | |
| **Exit** | **Name** | | **Type** | **Format** |
| msg | | String | “Generated after a successful election to inform the user that the process has been successfully completed.” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * **Identifier and name** | *[RF2-* **Single And Multiplayer Gameplay***]* | | | |
| **Summary** | * “The system must provide a user login functionality, allowing users to enter their nickname and password for authentication. The password input field should hide the characters entered for security reasons. Upon clicking the "Login" button, the system should verify the provided credentials and grant access if they are correct. If successful, the system should proceed to the user's session” | | | |
| **Entries** | **Name** | **Type** | | **Conditions** |
| numberPlayers | int | | “The system should allow a variable number of players to participate in the game, with a minimum requirement of 1 player for single-player mode and a minimum of 2 players for multiplayer mode” |
| namePlayers | int | | “In multiplayer mode, the system should allow each player to enter their name. The number of names in the array should correspond to the number of players in the game.” |
| gameModeSelection | enum | | “The system should provide an option for users to select either "SinglePlayer" or "Multiplayer" game mode. The game mode selected will determine the” |
| **Result o Postcondition** | After selecting the game mode (SinglePlayer or Multiplayer) and specifying the number of players, the system will provide an interface for player interaction. In multiplayer mode, the system will collect and display the names of all participating players. The gameplay will commence according to the selected mode, with players able to interact with the game using the provided user interface elements. | | | |
| **Exit** | **Name** | | **Type** | **Format** |
| msg | | string | “Generated after a successful election to inform the user that the process has been successfully completed.” |
|

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier and name** | *[RF3-****Map Selection****]* | | | |
| **Summary** | * “Users can select from different maps, regions, or places, which involves the integration of map-related features such as map display, location search, and interaction with geospatial information. This requirement can be relevant for applications related to navigation, tourism, logistics, and various other areas where location selection is essential.” | | | |
| **Entries** | **Name** | **Type** | | **Condition** |
| mapSelection | Int | | “Users can select a map from a predefined list of available maps. The dropdown should be populated with map options, and the user's selection is a required input for the system to proceed.” |
| **Result or Postcondition** | After the user selects a map from the dropdown, the system will have the user's choice recorded as the selected map for the game session. The system is ready to proceed with the selected map for gameplay, and any relevant game settings or configurations are updated to reflect the chosen map. | | | |
| **Exit** | **Name** | | **Type** | **Format** |
| confirmationMsg | | String | Generated after a successful election to inform the user that the process has been successfully completed. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier and name** | *[RF4-****Player Management****]* | | | |
| **Summary** | * This requirement involves the management of player-related data and interactions within the application. It includes features for registering and logging in players, managing player information, updating scores, tracking player progress, and allowing players to manage their profiles, such as personal information and avatars. | | | |
| **Entries** | **Name** | **Type** | | **Condition** |
| playerInformation | String | | “The system should accept and manage player information, including player names, scores, game progress, and other relevant details. |
| playerProfileManage |  | | The system should allow players to manage their profiles, including updating personal information, changing passwords, or customizing avatars. |
| registerPlayer | String | | The system should allow users to register as players, requiring them to input their desired usernames, passwords, and other relevant details as needed. |
| playerLogin. | String | | The system should provide a login functionality where registered players can enter their usernames and passwords to access their accounts. |
| **Result or Postcondition** | After a player registers or logs in, the system stores their player information and allows them to access and manage their profiles. The system also tracks player progress and updates scores based on in-game performance. Player data remains accurate and up to date, ensuring that players have a seamless and personalized experience within the application. | | | |
| **Exit** | **Name** | | **Type** | **Format** |
| confirmMsg | | String | “Generated after a successful election to inform the user that the process has been successfully completed.” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Identifier and name** | *[RF5-****Player Score****]* | | | |
| **Summary** | * This requirement pertains to the management and tracking of player scores within the game. It includes recording and storing player scores, implementing a scoring algorithm, and potentially tracking and displaying high scores for a competitive element in the game. | | | |
| **Entries** | **Name** | **Type** | | **Condition** |
| playerScore | Int | | The system should accept and record player scores as they progress in the game. |
| scoreStorage | Array | | The system should store and manage player scores, associating them with player profiles and game progress. |
| **Result or Postcondition** | After playing the game, player scores are accurately calculated, stored, and associated with player profiles. The system updates scores based on in-game actions, achievements, or performance. Optionally, high scores are tracked and available for competitive comparison, enhancing the gaming experience. Player score data remains reliable and up to date. | | | |
| **Exit** | **Name** | | **Type** | **Format** |
| msg | | String | “Generated after a successful election to inform the user that the process has been successfully completed.” |

**Non-Functional Requirements:**

**Intuitive User Interface:**

- Users should be able to view a full-path of Stations located along the puzzle

- The interface should allow users to apply the algorithms anytime.

*Phase 2: Evidence the search results by including in your report sufficient elements that allow you to know various approaches to the problem as well as related theoretical and practical elements. Include the reference of each of the sources from which the information was obtained.*

**Why Floyd-Warshall's Algorithm is Suitable for Maze Solving**

Floyd-Warshall's algorithm is particularly well-suited for maze solving due to its ability to efficiently find the shortest path between any two points in the maze. This makes it a suitable choice for mazes with multiple paths or complex layouts.

**Advantages of Floyd-Warshall's Algorithm:**

**Efficient**: Floyd-Warshall's algorithm has a time complexity of O(n^3), where n is the number of cells in the maze. This makes it relatively efficient for solving mazes of moderate size.

**Versatility**: The algorithm can handle mazes with multiple paths or complex layouts.

**Memory-efficient:** Floyd-Warshall's algorithm requires only constant space, making it memory-efficient.

*References:*

***Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.***

***Nilsson, N. J. (2010). Principles of artificial intelligence (3rd ed.). Morgan Kaufmann.***

***Russell, S. J., & Norvig, P. (2010). Artificial intelligence: A modern approach (3rd ed.). Prentice Hall.***

*Phase 3: Based on both the information collected and some idea generation technique, creative solution alternatives are presented. The idea generation technique is indicated and briefly described. The creative alternatives are enriched with your own ideas.*

The generation of ideas came from meeting as a group to discuss the complexity and ease of adaptation to the context of the problem of the available algorithms.

The **Floyd-Warshall algorithm** is a dynamic programming algorithm used to find the shortest path between all pairs of vertices in a weighted graph. This algorithm is suitable for the context of a maze game problem because it allows finding the shortest path between any cell in the maze and the exit.

The other algorithms mentioned are not as suitable for this context. **The BFS** algorithm and the **DFS** algorithm are used to explore a graph sequentially, starting from a given vertex. These algorithms cannot find the shortest path between two arbitrary vertices of the graph.

**Dijkstra's algorithm** is used to find the shortest path between a given vertex and all other vertices of a weighted graph. This algorithm is more efficient than the Floyd-Warshall algorithm for finding the shortest path between a single pair of vertices. However, Dijkstra's algorithm cannot find the shortest path between all pairs of vertices in the graph.

**Prim's algorithm and Kruskal's algorithm** are used to find a minimum spanning tree in a weighted graph. These algorithms are not designed to find the shortest path between two arbitrary vertices of the graph.

In conclusion, the **Floyd-Warshall** algorithm is the best algorithm to use in the context of a maze game problem because it allows you to find the shortest path between any cell in the maze and the exit.

*Phase 4: Documentation of the Discarding of Non-Viable Ideas:*

**Maze with BFS Algorithm:**

**Labyrinth Complexity**: 6

**Interaction with Algorithms**: 8

**Visit Nodes:** 7

**Intuitive User Interface**: 6

**Error Handling:** 7

Total: 34

**Maze with DFS Algorithm:**

**Labyrinth Complexity:** 8

**Interaction with Algorithms**: 7

**Visit Nodes:** 6

**Intuitive User Interface**: 7

**Error Handling: 8**

Total: 36

**Maze with Prim Algorithm:**

**Labyrinth Complexity:** 6

**Interaction with Algorithms:** 9

**Visit Nodes:** 6

**Intuitive User Interface**: 7

**Error Handling:** 8

Total: 36

**Maze with Dijkstra Algorithm:**

**Labyrinth Complexity**: 7

**Interaction with Algorithms**: 8

**Visit Nodes:** 6

**Intuitive User Interface**: 7

**Error Handling:** 8

Total: 36

**Maze with Floyd-Warshall Algorithm:**

**Labyrinth Complexity:** 7

**Interaction with Algorithms:** 8

**Visit Nodes:** 7

**Intuitive User Interface:** 8

**Error Handling:** 8

Total: 38

Based on the evaluation of the different algorithms, **Floyd-Warshall** emerges as the most suitable choice for solving mazes with complex layouts due to its superior performance in labyrinth complexity, intuitive user interface, and error handling. While DFS and Prim algorithms offer comparable performance in certain aspects, their overall scores fall short of Floyd-Warshall's. BFS, on the other hand, while being simple and efficient, may not be as effective for mazes with intricate structures.

*Phase 5: Define criteria to evaluate the ideas. Explains what each criterion consists of and all the scales that an alternative evaluated with that criterion may have. Evaluate each idea based on said criteria and assign a result of that evaluation. Totalize the evaluation to know, based on the chosen criteria, which are the ideas that will be implemented.*

**Evaluation criteria:**

**Visit Nodes:** The algorithms' ability to allow the player to access stations in exchange for points is evaluated. Scores are assigned based on the player's efficiency in visiting strategic nodes.

**0:** Inability to visit nodes.

**1-3:** Visit nodes on a limited basis.

**4-7:** Good ability to visit nodes.

**8-10:** Excellent ability to visit nodes.

**Intuitive User Interface:** The user interface is evaluated in terms of its friendliness and ability to provide clear information about the game and the help available.

**0:** Confusing and unintelligible interface.

**1-3:** Unintuitive interface.

**4-7:** Quite intuitive interface**.**

**8-10:** Highly intuitive interface.

**Error Handling**: Evaluate how the game handles player errors, such as incorrect moves, and the effectiveness of providing opportunities to correct them**.**

**0:** Poor error handling.

**1-3**: Limited error handling**.**

**4-7:** Good error handling.

**8-10:** Excellent error handling.

**Maze Complexity**: Evaluate the difficulty and overall complexity of the designed maze.

0: Labyrinth too simple, without challenges.

**1-3:** Labyrinth not very challenging.

**4-7:** Challenging and balanced maze**.**

**8-10:** Very complex maze, but still enjoyable.

Interaction with Algorithms:

**Interaction with Algorithms:** Evaluate how algorithms affect gameplay and user experience.

**0:** confusing interaction.

**1-3:** Minimal interaction and little impact**.**

**4-7:** Beneficial and balanced interaction.

**8-10:** Strategic and enriching interaction.