**Integrative Task #2 Computation and Discrete Structures 1**

**By: Juan Pablo Parra, Stick Martinez and Daniel Gonzalez**

**Specification of requirements:**

**SOFTWARE ENGINEERING PROBLEM SPECIFICATION TABLE**

|  |  |
| --- | --- |
| CLIENT | Owner of the task and reminder system |
| USER | Any person using the system |
| FUNCTIONAL REQUIREMENTS | **RF1**: Board Representation with the vertexes.  **RF2**: Movement of players.  **RF3**: Counting the points of the players.  **RF4**: Calculate shortest path.  **RF5:** Mark the road with less weight. |
| CONTEXT OF THE PROBLEM | We need to make a system for a game that's like a maze, but with some changes. Two players will compete in a 1 vs. 1 match, moving around a grid that looks like a box game. The catch is that each player can only go to nearby spots. We're using a graph to model the game, where each square on the board is like a point in the graph. To make things interesting, we're not making all moves have the same difficulty. Usually, moves in games are equally challenging, but we're mixing it up. This way, the game is more fun, and the strategies we use in the game will work well. |
| NON-FUNCTIONAL REQUIREMENTS | RNF : Easy-to-understand concrete user interface by using Java FX |

| **Name or identifier** | **RF1** | | |
| --- | --- | --- | --- |
| **Summary** | The program must be able to represent the game board with its predetermined vertices with their edges. Besides that, it should show the different weights of the edges to provide the correct tools to the player. | | |
| **Entries** | Input name | Data type | Selection or repetition condition |
| **Result or postcondition** | The appropriate number of vertices is created, there must be a minimum of 50. | | |
| **Outputs** | Output name | Data type | Selection or repetition condition |
| Vertex | Button | if the result was correct all modeled vertices appear as buttons |

| **Name or identifier** | **RF2** | | |
| --- | --- | --- | --- |
| **Summary** | The program must be able to represent that a player has moved from one square to another by pressing the vertex. It should be noted that the player can only move to adjacent vertices. Near the vertex it should show which player is standing on that square. | | |
| **Entries** | Input name | Data type | Selection or repetition condition |
| ClicButton | Button | when clicked select the vertex you want to go to |
| ButtonGo | Button | Once selected, click on go. |
| **Result or postcondition** | The button where the player is standing is shaded. | | |
| **Outputs** | Output name | Data type | Selection or repetition condition |
| WhatPlayer | String | it says which player is standing. |

| **Name or identifier** | **RF3** | | |
| --- | --- | --- | --- |
| **Summary** | The program must be able to calculate the points of each player. The points are nothing more than the weight of each edge along which the players moved. Thus giving the score of each player and the winner is the one who arrives with the lowest score or weighting. | | |
| **Entries** | Input name | Data type | Selection or repetition condition |
| OtroButton | Button | the vertex to which you want to move |
| ButtonGo | Button | Once selected, click on go. |
|  | SumaPonderacion | int | the weight of the edge |
| **Result or postcondition** | Adds up the weighting between edges | | |
| **Outputs** | Output name | Data type | Selection or repetition condition |
| Points | int | the points of the player |

| **Name or identifier** | **RF4** | | |
| --- | --- | --- | --- |
| **Summary** | The program must be able to calculate the shortest path from where the player is currently standing to the goal. This will be modeled as a power in the game that will be given every 5 rounds. It should be noted that this will be done with the Dijkstra algorithm. | | |
| **Entries** | Input name | Data type | Selection or repetition condition |
| keyVertex | K | the key of the vertex |
| **Result or postcondition** | Calculate the shortest path from q to the target | | |
| **Outputs** | Output name | Data type | Selection or repetition condition |
| ItStore | boolean | if true it was stored correctly, if false it was not stored correctly |

| **Name or identifier** | **RF5** | | |
| --- | --- | --- | --- |
| **Summary** | The program must be able to mark the edge or path that has the least weighting where the player is standing. This will be modeled as a power that will be given to the player every 2 rounds. It should be noted that this can be done thanks to Prim's algorithm. | | |
| **Entries** | Input name | Data type | Selection or repetition condition |
| **Result or postcondition** | Marks the path with less weight | | |
| **Outputs** | Output name | Data type | Selection or repetition condition |
| ItStore | boolean | if true it was stored correctly, if false it was not stored correctly |