# Tarea Integradora 2 – Computación y estructuras discretas I

# Método de la ingeniería

**1.**    Definición del problema

La universidad Icesi necesita crear un juego basado en la lógica de Snakes and Ladders, la temática elegida para desarrollarlo será Rick and Morty, el juego debe contar con 2 jugadores en simultaneo, cada uno con un nickname diferente. Los jugadores deben recolectar la mayor cantidad de semillas. En el juego se debe poder establecer la cantidad de los portales y la cantidad de semillas. El juego finalizará cuando se hayan recolectado todas las semillas y el jugador que posea más semillas tendrá un puntaje más alto, por lo tanto, será el ganador.

**2**.    Investigación

* **Stack:** es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo LIFO. la pila es un contenedor de nodos.
* **Queue:** se utiliza para almacenar elementos como cualquier otra estructura de datos, pero de una manera particular. En palabras simples, podemos decir que la cola es un tipo de estructura de datos en el lenguaje de programación Java que almacena elementos del mismo tipo.
* **Priority queue:** es un tipo de dato abstracto similar a una cola en la que los elementos tienen adicionalmente, una prioridad asignada. En una cola de prioridades un elemento con mayor prioridad será desencolado antes que un elemento de menor prioridad. Si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de cola.
* **Grafo:** Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en inglés) que pueden ser orientados o no. Los grafos son estructuras discretas que constan de vértices y aristas que conectan entre si esos vértices. Por lo tanto, un grafo G costa de dos partes: 1) Un conjunto V = V (G) cuyos elementos se denominan vértices, puntos o nodos de G. 2) Un conjunto E = E(G) de pares de vértices distintos denominados aristas de G
* **Lista enlazada:** es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior.
* **Tabla Hash:** es una estructura de datos que implementa el tipo de dato abstracto llamado Diccionario. Esta asocia *llaves* o *claves* con *valores*. Las tablas hash almacenan la información en posiciones pseudoaleatorias, así que el acceso ordenado a su contenido es bastante lento.
* **Arreglo:** son**agrupaciones de campos de memoria en los cuales podemos almacenar valores de forma «ordenada»**, estos valores almacenados pueden ser de tipo primitivo o de objetos (clases), para manipular los datos (agregar, eliminar, buscar) se accede a ellos por medio de su índice, el cual va desde 0 (cero) hasta su valor -1 (n-1)

**3.**    **Especificación de requerimientos**

R\_F1. El programa debe preguntar los nicknames de los 2 jugadores.

R\_F2. El programa debe permitir 2 jugadores en simultaneo.

R\_F3. El programa debe permitir al usuario definir la cantidad de semillas que desea en el juego.

R\_F4. El programa debe permitir al usuario definir la cantidad de portales que desea en el juego.

R\_F5. El programa debe sumar el puntaje de un jugador cada vez que recoja una semilla.

R\_F6. El programa debe teletransportar a un jugador hacia la pareja del portal donde se encuentre ubicado.

R\_F7. El programa debe prohibir la creación de más de una pareja de portales representada con exactamente la misma letra(Mayus importan).

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F1. Preguntar los nicknames de los 2 jugadores . | | |
| Summary | Cuando se este iniciando el juego, se deben pedir los 2 nicknames. | | |
| Input | Entry name | Data type | Selection or repetition condition |
| Nickname | String | String ¡= String |
| General activities necessary to obtain the results | El usuario debe digitar los nicknames | | |
| Outcome | Nickname guardado | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
| Nickname | String | Nickname == String |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F2. Ingresar 2 jugadores | | |
| Summary | El programa debe permitir que 2 jugadores puedan jugar en el mismo dispositivo cada uno con su respectivo turno y su respectivo puntales. | | |
| Input | Entry name | Data type | Selection or repetition condition |
| Player | Jugador | N/A |
| General activities necessary to obtain the results | Crear los dos objetos de clase jugador | | |
| Outcome | Jugadores creados | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F3. Definir cantidad de semillas | | |
| Summary | El programa debe permitir que los jugadores establezcan la cantidad mínima de semillas | | |
| Input | Entry name | Data type | Selection or repetition condition |
| AmountOfseeds | Int | AmountOfSeeds <= cantidad de casillas & > 0 |
| General activities necessary to obtain the results | Verificar que la cantidad de semillas sea menor o igual a la cantidad de casillas y mayor a 0 | | |
| Outcome | Cantidad de semillas guardadas | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F4. Definir cantidad de portales | | |
| Summary | El programa debe permitir que los jugadores establezcan la cantidad mínima de portales | | |
| Input | Entry name | Data type | Selection or repetition condition |
| AmountOfportals | Int | AmountOfportals < mitad casillas -1 & > 0 |
| General activities necessary to obtain the results | Verificar que la cantidad de portales sea mayor a 0 y menor que la mitad de las casillas | | |
| Outcome | Cantidad de portales guardados | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F5. Sumar puntaje | | |
| Summary | El programa debe estar en la capacidad de sumarle puntaje al jugador cada vez que este recoja una semilla en el juego | | |
| Input | Entry name | Data type | Selection or repetition condition |
| Score | int | N/A |
| General activities necessary to obtain the results | Ingresar la opción para poder acceder al monitor. | | |
| Outcome | Se le suma puntaje al jugador después de haber recolectado la semilla | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F6. Teletransportacion de jugador | | |
| Summary | El programa debe permitir que los jugadores sean teletransportados hacia la pareja del portal en el que se encuentran | | |
| Input | Entry name | Data type | Selection or repetition condition |
| pair | String | Pair == portal |
| General activities necessary to obtain the results | Verificar que el portal donde saldrá el jugador tenga asignada la misma identificación que el portal en el que se encuentra | | |
| Outcome | Jugador teletransportado | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | R\_F7. Prohibir crear mas de una pareja de portales con la misma identificación | | |
| Summary | El programa debe prohibir que se puedan crear mas de 1 pareja de portales identificados por la misma letra, cada pareja de portales tendrá una identificación distinta | | |
| Input | Entry name | Data type | Selection or repetition condition |
| idOfportal | Array of char |  |
| General activities necessary to obtain the results | Verificar que la identificación de una pareja de portales sea unica | | |
| Outcome | Pareja de portales con una identificacion unica | | |
| Outputs | Entry name | Data type | Selection or repetition condition |
|  |  |  |

**4.**    Brainstorm

1. utilizando stack como estructura de datos se crean las parejas de portales en orden, así verificamos que cada pareja tenga una identificación única, además se pueden crear las semillas utilizando una queue para asegurarnos que sean recolectadas por los jugadores en el mismo orden que fueron creadas.

2. Utilizando un grafo no dirigido se podría facilitar el movimiento de los jugadores entre las casillas del tablero. Así mismo, la utilización de grafos nos permite implementar una herramienta de ayuda al jugador que le indique la ruta más corta a tomar para recolectar una semilla.

3. usando listas doblemente enlazadas puede lograrse un fácil movimiento entre casillas por parte de los jugadores. Sin embargo, el nuevo modo de juego con la movilidad extendida sería más complicado de implementar. De la misma forma, el algoritmo de ayuda de la ruta más corta (Dijkstra) también sería más complejo de implementar.

4. Utilizando arreglos se almacena la cantidad de portales con el fin de poder navegar entre ellos a medida que el jugador lo desee, en un arreglo distinto también se almacenan las semillas para efectos de facilidad a la hora de recorrer el tablero, las semillas podrían almacenarse en el arreglo de forma intermitente en las posiciones del arreglo, esto tratando de simular una posición con semilla y otra posición nula.

5. utilizando matrices de adyacencia para crear los nuevos vértices del grafo para que se creen aristas automáticamente con todos los vértices adyacentes. De manera que todos los vértices adyacentes estarán conectados bajo el mismo peso.

6. Implementando un grafo dirigido se podría recorrer el tablero en su totalidad en una única dirección sin posibilidad de retroceder, permitiendo así que el jugador vaya siempre a donde está el vértice siguiente del vértice donde se encuentra, así se aseguraría de recolectar las semillas yendo vértice por vértice.

4.1 Descarte de ideas

Luego de haber discutido las 6 posibles soluciones, analizarlas y considerar planes de trabajo, concluimos que las soluciones descartadas serán las soluciones 1, 3 y 4. Viendo la estructura de datos utilizada en la solución 1, nos parece que stack no es adecuado para la creación de las parejas de portales, porque al momento de implementar el grafo, tocaría desapilar cada pareja de portales y verificar tanto su id, como su peso. En la solución 3, se propuso implementar una lista doblemente enlazada, pero sabemos que esta no apta para el desplazamiento de jugadores que se necesita, sino más bien para almacenamiento de id. Finalmente hablando de la solución 4 se propone almacenar la cantidad de portales y semillas en 2 arreglos distintos, igual que en la solución 3, un arreglo no es apto para simular un tablero y mucho menos recolectar objetos añadidos, además tampoco tendría sentido almacenar objetos cada uno con una casilla de diferencia.

4.2 Ideas seleccionadas

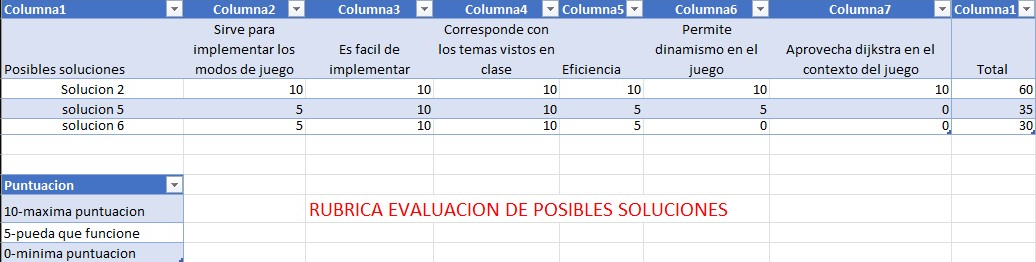
Las ideas con las que nos quedamos son las ideas 2, 5 y 6.

2. Utilizando un grafo no dirigido se podría facilitar el movimiento de los jugadores entre las casillas del tablero, pues sabemos que en el juego se debe poder avanzar, ir a la derecha, ir a la izquierda y se debe poder retroceder. Así mismo, la utilización de grafos nos permite implementar una herramienta de ayuda al jugador que le indique la ruta más corta a tomar para recolectar una semilla que en este caso sería el algoritmo Dijkstra visto en clase. También se podría implementar DFS con la idea de retar al jugador a que no pase 2 veces por la misma casilla.

5. utilizando matrices de adyacencia para crear los nuevos vértices del grafo para que se creen aristas automáticamente con todos los vértices adyacentes. De manera que todos los vértices adyacentes estarán conectados bajo el mismo peso. Esto nos permite tener un juego más parejo entre jugadores debido a que moverse hacia cualquier casilla siempre tendrá el mismo costo. Esta implementación facilitaría el uso de DFS para no pasar 2 veces por la misma casilla, pero Dijkstra sería más complejo de implementar.

6. Implementando un grafo dirigido se podría recorrer el tablero en su totalidad en una única dirección sin posibilidad de retroceder, permitiendo así que el jugador vaya siempre a donde está el vértice siguiente del vértice donde se encuentra, así se aseguraría de recolectar las semillas yendo vértice por vértice. Esta solución al igual que la anterior sería mas optima para DFS pero el algoritmo de Dijkstra no se vería aprovechado, logrando así un juego más lineal porque solo se movería en una sola dirección.

4.3 Rubrica de evaluación soluciones:



**5**.    Escoger la mejor solución

Llegamos a la conclusión de que la mejor solución es la 2 porque, teniendo en cuenta la optimización de rendimiento que ofrecen los grafos sería mucho más cómodo el desplazamiento de los jugadores por el tablero, teniendo en cuenta que tal como se enunció en la solución, estos deben poder moverse hacia adelante, atrás, arriba y abajo, además de que la recolección de las semillas sería más flexible. En conclusión, tal como se indica en la solución esta buscaría siempre la ruta más corta logrando así una mayor facilidad a la hora de desplazamiento y recolección.

# Abstract Data Type- ADT:

|  |
| --- |
| **ADT <Graph>** |
| Graph = {<Vertex, Edge, Weight>, Vertex = {}, Edge = {} |
| {inv: } |
| Primitive operation:   * create: <> à Graph * addVertex: <Vertex> à Boolean * removeVertex: <Vertex> à Boolean * addEdge: <Vertex1, vertex2, weight> à Boolean * searchVertex: <Vertex> à Boolean |