Esteban Ariza, Johan Sebastián Giraldo, Mateo Valdés

Enlace de GitHub: https://github.com/mavaldot/AED-Final-Project

Final Project

Problem Definition:

Johan, Esteban and Mateo have traveled to Japan for a job interview on Nintendo. When they arrived at Nintendo's skyscraper, Mr. Sakurai said “You must prove your courage by creating the best mobile game I have ever seen. In this contest, Shigeru Miyamoto-san will choose the best videogame designer, based on their programming skills and imagination”. After that, PandiYa members started creating the best game ever.

Mateo, the mastermind, had the incredible idea of creating a video game that includes spatial gaming skills, Esteban started planning the gameplay and Johan the design itself. The main idea of the game is to choose, among many paths, the shortest path between two points, but this concept is not simple at all. The player has a limited amount of movements and some levels have special properties that make the game even harder. In addition, if you achieve a level with the minimum steps possible, you could get more stars (3 if you get a perfect score) to unlock the next levels. For the level designs, they decided to import the levels from a flat file with all the information of the challenge. Finally, they need to add funny stuff to keep the attention of the player.

1. Identifying the problem

Problem: A game must be created wherein a player must choose, among many paths, the shortest path between two points to achieve all the levels.

2. Research

Functional requirements:

Name	FR1: Load a level
Summary	The system allows creating a level by reading a plain text file that includes the number of vertices of the graph and their coordinates on the screen, the adjacency matrix, the vertex where start and the vertex where ends, the level number, number of stars required for unlocking the level and finally, the color of the character.
Input	The **path** of the plain text file with the level information
Output	The level was created successfully

Name	FR2: Unlock a level
Summary	The system allows unlocking a level. A level only can be unlocked if the user has the number of stars required for the level.
Input	-
Output	The level was unlocked successfully

Name	RF3: Calculate minimum movements required
Summary	The system allows calculating the minimum movements required to go from the start vertex to the ends vertex,
Input	-
Output	The minimum movements to go from the start vertex to the ends vertex

Name	RF4: Calculate maximum movements allowed

Summary	The system allows calculating the maximum movements allowed to go from the start vertex to the ends vertex. The way to calculate the number of movements allowed is adding up two to the number of movements required for the minimum path
Input	-
Output	The maximum movements

Name	FR5: Play a level
Summary	The system allows playing a level, the level is passed if the path lets go from the start vertex to the ends vertex with the movements allowed.
Input	A path from the start vertex to ends vertex
Output	The level is played successfully

Name	FR6: Calculate the number of stars earned in the level.
Summary	The system allows calculating the number of stars earned depending on how many movements were left over, if there are 2 movements left, earns 3 stars, if there are 1 movement left, earns 2 stars, if there are no one movements left, earns 1 star.
Input	-
Output	The number of stars earned

Matrix

Matrix is a simple data structure that stores data in an organized form in the form of rows and columns. It could be useful to manipulate data in an easier and more organized way. Some problems of the matrix, is the spatial complexity ($O(n^2)$) and is not efficient in some searching algorithms.

The matrix is very useful to represent 2D problems and is also used in graphs as Adjacency Matrix to represent the edges between two vertices.

Vector
Java, C++ and other programming languages use vectors or arrays to store data. It's also used in some game engines such as Unity and Unreal to make easier and fluid transitions, translations and scales.

Graph
A graph is a data structure that has a finite set of vertices and a finite set of edges, where a vertex or also called node, is an element of the graph and an edge is a vertex connection by pairs in the form (u,v).

Graphs can be used to model many real life situations such as the roads of a city, relationships of influence,

There are two commonly used representation of graphs:
1. Adjacency List
2. Adjacency Matrix

Tree
Trees, a particular type of graph that is organized hierarchically. The first vertex is called root, and it is connected with nodes that are also connected with other nodes, creating a structure similar to a tree. A tree is a very particular way of representing a graph, because it is easier to visualize the connections between the nodes.

References:

Wankhede, Rahul. Different Operations on Matrices. GeeksForGeeks.
https://www.geeksforgeeks.org/different-operation-matrices/

Msakibhr, Estenger. Vector in C++ STL. GeeksForGeeks.
https://www.geeksforgeeks.org/vector-in-cpp-stl/

Malhotra, Kanav. Graph and its representations. GeeksForGeeks.
https://www.geeksforgeeks.org/graph-and-its-representations/

Barnwal, Aashish. Graph and its representations. GeeksForGeeks.
https://www.geeksforgeeks.org/graph-and-its-representations/

Singh, Shikha. Binary Tree Data Structure. GeeksForGeeks.
https://www.geeksforgeeks.org/binary-tree-data-structure/

3. Creative solutions.

Brainstorming:

Alternative #1:
Have the user draw the points and paths in a piece of paper, and then ask him to find the shortest path and keep track of the score himself manually.

Alternative #2:
Have the user draw the points and paths in a piece of paper, and then ask him to find the shortest path and have a friend or acquaintance keep track of the score manually.

Alternative #3:
Use a tree to represent the points and the paths, with the tree nodes representing the points and the connections between the parent nodes and the children nodes representing the paths.

Alternative #4:
Use a vector or arraylist of size two to store every pair of points that are connected by a path.

Alternative #5:
Use a matrix as a game board of nodes, in which the user could travel between nodes as a labyrinth.

Alternative #6:
Use a graph to represent the points and the paths, with the nodes in the graph representing the points and the edges representing the paths between each two points.

Alternative #7:
Use a matrix of points as a game board, and if two points are in the same column or row then they are connected by a path.

Alternative #8:
Use a Doubly LinkedList of points, and if a point is linked to another point in the LinkedList then there is a path between those two points.

4. Transformation to preliminary designs

Alternative #1:
This alternative might be tedious or boring to the user, and if he is the one responsible for keeping his own score, then he might be dishonest or cheat by increasing his score. Therefore this alternative is discarded.

Alternative #2:

This alternative might also be tedious or boring to the user, and the user might be dishonest and try to persuade the friend or acquaintance to increase his score. Therefore this alternative is discarded.

Alternative #3:
Trees are a very popular and useful data structure, and they might even be useful in this case because their nodes can represent the points in this specific problem. Thus, we must consider this alternative.

Alternative #4:
This implementation seems easy and straightforward to implement in order to successfully solve our problem, so we must consider this alternative.

Alternative #5:
Alternative seems too complicated to implement effectively, so we must unfortunately discard this alternative.

Alternative #6:
This alternative seems to model this problem perfectly, since the points can be represented by nodes and the paths can be represented by edges, so we must consider this alternative.

Alternative #7:
Even though the program might take up a lot of space if the matrix gets too big, this alternative still gives us a valid solution and we must therefore consider it.

Alternative #8:
This alternative lacks versatility because if we use a Doubly LinkedList then a point can have a maximum of two paths, and our program might need more than two paths. Therefore, we must discard this alternative.

5. Evaluation and selection of the best solution

Criterion A: The problem model.

- [3] Represent the problem the best way possible.
- [2] The problem could be represented at least with this model.
- [1] The model is not appropriate for this problem.

Criterion B: Versatility in this problem.

- [3] This solution is very versatile and works in most situations.
- [2] This solution is moderately versatile and works in certain situations.
- [1] This data structure is not versatile at all and can only work in limited situations.

Criterion C: The difficulty of implementation.

- [3] The implementation is simple, even trivial

- [2] Average people can implement this solution.
- [1] Very difficult to actually implement

Criterion D: The speed of the solution.

- [3] The solution is the fastest discovered in history
- [2] The solution is relatively fast
- [1] The solution is extremely slow

Solution	**Criterion A**	**Criterion B**	**Criterion C**	**Criterion D**	**Total**
Tree	3	1	2	3	9
Vector	1	1	2	2	6
Graph	3	3	3	3	12
Matrix	2	2	2	2	8

Justification:

- Tree is a good way to represent the problem, however it lacks the versatility of a graph.
- Is very difficult to represent this problem as a vector, because of the limitations of this data structure.
- A graph is a good solution for the problem, because it is very intuitive and easy to represent in this particular context.
- The problem could be represented as a matrix, but there are a lot of better options for the implementation.

6. Preparing the report and specifications

ADT - Graph
Graph G = <N, E> • N is a collection of vertices $\{n_1, n_2, n_3, ..., n_n\} n_1, n_2, n_3, ..., n_n\}$ • E is a collection of edges { $e_1, e_2, e_3, ..., e_n$ } that connect two Vertices $e_1, e_2, e_3, ..., e_n$ } that connect two vertices
$\|N\| \geq 0, \ \|E\| \geq 0$
Primitive operations: • CreateGraph: → Graph : Creator • AddVertex: Graph x Vertex → Graph : Mutator

● RemoveVertex:	Graph x Vertex	→	Graph : Mutator
● AddEdge:	Graph x Vertex x Vertex	→	Graph : Mutator
● RemoveEdge:	Graph x Vertex x Vertex	→	Graph : Mutator
● GetVertices:	Graph	→	Vertices : Observer
● GetEdges:	Graph	→	Edges : Observer
● GetVertex:	Graph x Integer	→	Vertex : Observer
● GetEdge:	Graph x Vertex x Vertex	→	Edge : Observer
● BFS	Graph x Vertex	→	Graph : Observer
● DFS	Graph	→	Graph : Observer
● Prim	Graph x Vertex	→	Graph : Observer
● Kruskal	Graph	→	Graph : Observer
● Djikstra	Graph x Vertex	→	Graph : Observer
● FloydWarshall	Graph	→	Integer Matrix : Observer

CreateGraph()

"Creates a new graph"

{ pre: $true$ }
{ post: $\exists G \wedge N = \{\} \wedge E = \{\}$ }$\exists G \wedge N = \{\} \wedge E = \{\}$}

AddVertex(Graph, Vertex**)**

"Adds a Vertex to N, the graph's collection of Vertices"

{ pre: $\exists\, G$ }
{ post: $n \in N$ } $n \in N$ }

RemoveVertex(Graph, Vertex**)**

"Removes a Vertex from N, the graph's collection of Vertices"

{ pre: $\exists\, G \wedge N \neq \{\}$ }
{ post: $n \notin N$ }$n \notin N$ }

AddEdge(Graph, Vertex, Vertex**)**

"Adds an edge that connects the two Vertices"

{ pre: $\exists G \wedge n_1, n_2 \in N$ }$\exists G \wedge n_1, n_2 \in N$ }
{ post: $e_{n_1,n_2} \in E$ }$e_{n_1,n_2} \in E$ }

RemoveEdge(Graph**,** Vertex, Vertex**)**

“Removes the edge that connects the two Vertices”

{ pre: $\exists G \wedge n_1, n_2 \in N \wedge e_{n_1,n_2} \in E$ }$\exists G \wedge n_1, n_2 \in N \wedge e_{n_1,n_2} \in E$ }
{ post: $e_{n_1,n_2} \notin E$ }$e_{n_1,n_2} \notin E$ }

GetVertices(Graph**)**

“Returns N, the graph’s collection of Vertices”

{ pre: $\exists G$ }
{ post: N }

GetEdges(Graph**)**

“Returns E, the graph’s collection of edges”

{ pre: $\exists G$ }
{ post: E }

GetVertex(Graph, Integer**)**

“Returns n, which is a Vertex inside N, which is the graph’s collection of Vertices”

{ pre: $\exists G \wedge n \in N$}$\exists G \wedge n \in N$}
{ post: n }

GetEdge(Graph, Vertex, Vertex**)**

“Returns e, which is a Edge inside E, which is the graph’s collection of Edges”

{ pre: $\exists G \wedge (n1, n2) = e \; \varepsilon E$}
{ post: e }

BFS(Graph, Vertex**)**

“Searches the graph beginning from a source Vertex using Breadth First Search”

{ pre: $\exists G \in N$ }
{ post: BF Graph }

DFS(Graph, Vertex**)**

"Searches the graph beginning from a source Vertex using using Depth First Search"

{ pre: $\exists G \in N$ }
{ post: DF Graph }

Prim(Graph, Vertex**)**

"Finds the minimum spanning tree (MST) of the graph"

{ pre: $\exists G \wedge n \in N$ }
{ post: MST }

Kruskal(Graph**)**

"Finds the minimum spanning forest (MSF) of the graph"

{ pre: $\exists G \wedge n \in N$ }
{ post: MSF }

Djikstra(Graph, Vertex**)**

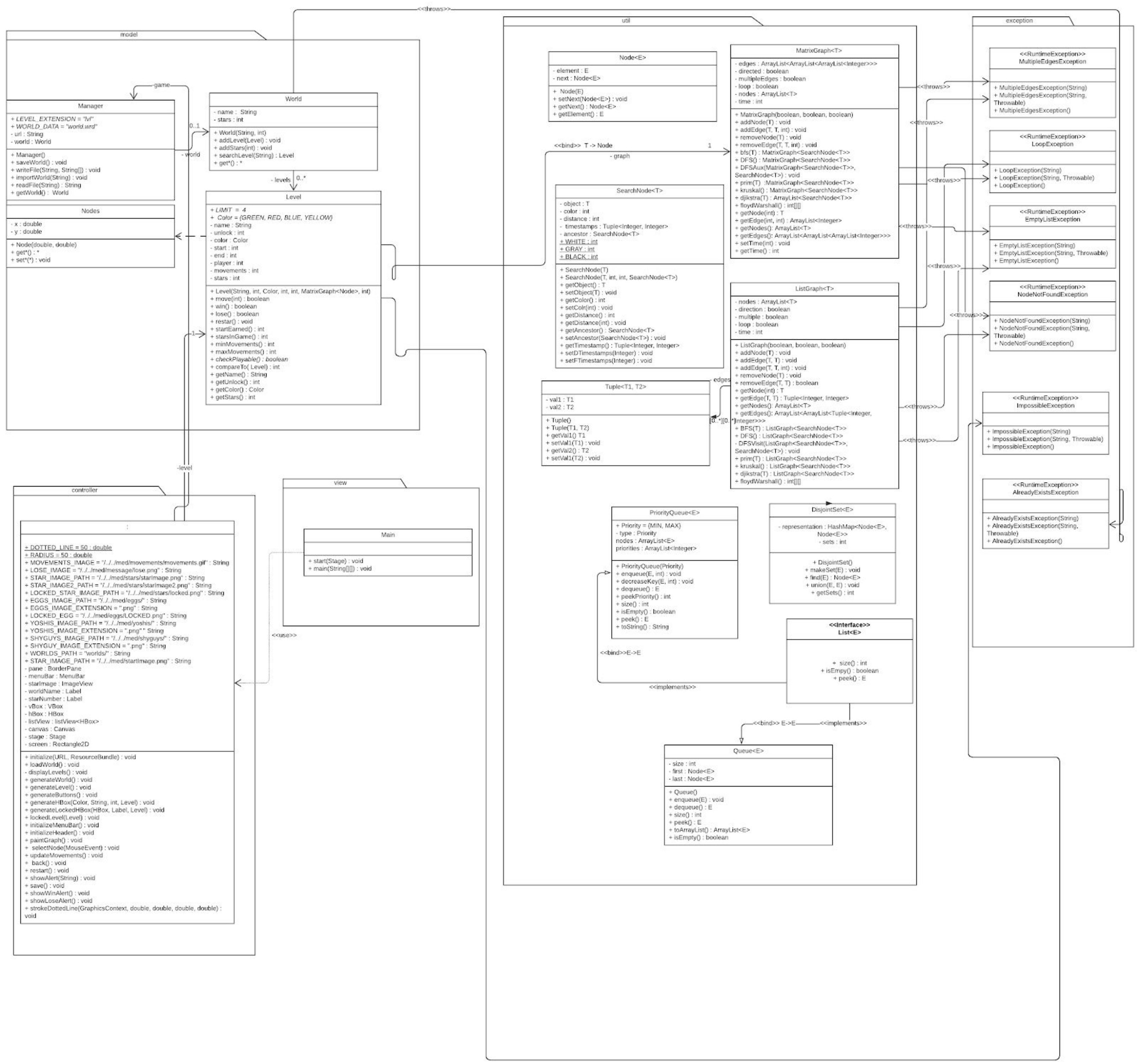
"Finds the shortest path between the Vertices in the graph and returns a new graph"

{ pre: $\exists G \wedge n \in N$ }
{ post: New graph }

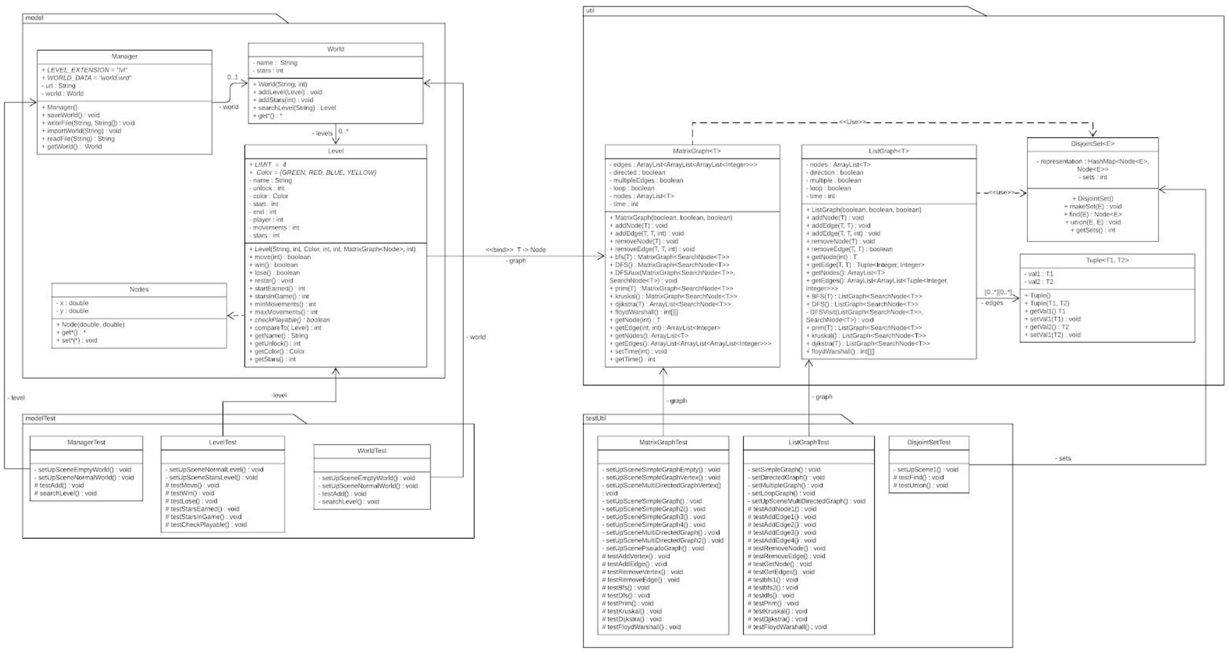
FloydWarshall(Graph**)**

"Returns a matrix with the shortest path between each pair of vertices"

{ pre: $\exists G$ }
{ post: Matrix }



Link para ver la imagen:
https://cdn.discordapp.com/attachments/588195711414304768/711796548828397589/YoshiClassDiagram.png


model
Manager
+ LEVEL_EXTENSION = "lvl"
+ WORLD_DATA = "world.wrd"
- url : String
- world : World
+ Manager()
+ saveWorld() : void
+ writeFile(String, String[]) : void
+ importWorld(String) : void
+ readFile(String) : String
+ getWorld() : World
0..1
- world
World
- name : String
- stars : int
+ World(String, int)
+ addLevel(Level) : void
+ addStars(int) : void
+ searchLevel(String) : Level
+ get*() : *
- levels
0..*
Level
+ LIMIT = 4
+ Color = {GREEN, RED, BLUE, YELLOW}
- name : String
- unlock : int
- color : Color
- start : int
- end : int
- player : int
- movements : int
- stars : int
+ Level(String, int, Color, int, int, MatrixGraph<Node>, int)
+ move(int) : boolean
+ win() : boolean
+ lose() : boolean
+ restar() : void
+ startEarned() : int
+ starsInGame() : int
+ minMovements() : int
+ maxMovements() : int
+ checkPlayable() : boolean
+ compareTo(Level) : int
+ getName() : String
+ getUnlock() : int
+ getColor() : Color
+ getStars() : int
Nodes
- x : double
- y : double
+ Node(double, double)
+ get*() : *
+ set*(*) : void
- level
-level
- world
modelTest
ManagerTest
- setUpSceneEmptyWorld() : void
- setUpSceneNormalWorld() : void
testAdd() : void
searchLevel() : void
LevelTest
- setUpSceneNormalLevel() : void
- setUpSceneStarsLevel() : void
testMove() : void
testWin() : void
testLose() : void
testStarsEarned() : void
testStarsInGame() : void
testCheckPlayable() : void
WorldTest
- setUpSceneEmptyWorld() : void
- setUpSceneNormalWorld() : void
- testAdd() : void
- searchLevel() : void
<<bind>> T -> Node
- graph
util
<<Use>>
MatrixGraph<T>
- edges : ArrayList<ArrayList<ArrayList<Integer>>>
- directed : boolean
- multipleEdges : boolean
- loop : boolean
- nodes : ArrayList<T>
- time : int
+ MatrixGraph(boolean, boolean, boolean)
+ addNode(T) : void
+ addEdge(T, T, int) : void
+ removeNode(T) : void
+ removeEdge(T, T, int) : void
+ bfs(T) : MatrixGraph<SearchNode<T>>
+ DFS() : MatrixGraph<SearchNode<T>>
+ DFSAux(MatrixGraph<SearchNode<T>>, SearchNode<T>) : void
+ prim(T) :MatrixGraph<SearchNode<T>>
+ kruskal() : MatrixGraph<SearchNode<T>>
+ djikstra(T) : ArrayList<SearchNode<T>>
+ floydWarshall() : int[][]
+ getNode(int) : T
+ getEdge(int, int) : ArrayList<Integer>
+ getNodes(): ArrayList<T>
+ getEdges(): ArrayList<ArrayList<ArrayList<Integer>>>
+ setTime(int) : void
+ getTime() : int
ListGraph<T>
- nodes : ArrayList<T>
- direction : boolean
- multiple : boolean
- loop : boolean
- time : int
+ ListGraph(boolean, boolean, boolean)
+ addNode(T) : void
+ addEdge(T, T) : void
+ addEdge(T, T, int) : void
+ removeNode(T) : void
+ removeEdge(T, T) : boolean
+ getNode(int) : T
+ getEdge(T, T) : Tuple<Integer, Integer>
+ getNodes(): ArrayList<T>
+ getEdges(): ArrayList<ArrayList<Tuple<Integer, Integer>>>
+ BFS(T) : ListGraph<SearchNode<T>>
+ DFS() : ListGraph<SearchNode<T>>
- DFSVisit(ListGraph<SearchNode<T>>, SearchNode<T>) : void
+ prim(T) : ListGraph<SearchNode<T>>
+ kruskal() : ListGraph<SearchNode<T>>
+ djikstra(T) : ListGraph<SearchNode<T>>
+ floydWarshall() : int[][]
<<use>>
DisjointSet<E>
- representation : HashMap<Node<E>, Node<E>>
- sets : int
+ DisjointSet()
+ makeSet(E) : void
+ find(E) : Node<E>
+ union(E, E) : void
+ getSets() : int
[0..*][0..*]
- edges
Tuple<T1, T2>
- val1 : T1
- val2 : T2
+ Tuple()
+ Tuple(T1, T2)
+ getVal1() T1
+ setVal1(T1) : void
+ getVal2() : T2
+ setVal1(T2) : void
- graph
- graph
testUtil
MatrixGraphTest
- setUpSceneSimpleGraphEmpty() : void
- setUpSceneSimpleGraphVertex() : void
- setUpSceneMultiDirectedGraphVertex() : void
- setUpSceneSimpleGraph() : void
- setUpSceneSimpleGraph2() : void
- setUpSceneSimpleGraph3() : void
- setUpSceneSimpleGraph4() : void
- setUpSceneMultiDirectedGraph() : void
- setUpSceneMultiDirectedGraph2() : void
- setUpScenePseudoGraph() : void
testAddVertex() : void
testAddEdge() : void
testRemoveVertex() : void
testRemoveEdge() : void
testBfs() : void
testDfs() : void
testPrim() : void
testKruskal() : void
testDijkstra() : void
testFloydWarshall() : void
ListGraphTest
- setSimpleGraph() : void
- setDirectedGraph() : void
- setMultipleGraph() : void
- setLoopGraph() : void
- setUpSceneMultiDirectedGraph() : void
testAddNode1() : void
testAddEdge1() : void
testAddEdge2() : void
testAddEdge3() : void
testAddEdge4() : void
testRemoveNode() : void
testRemoveEdge() : void
testGetNode() : void
testGetEdges() : void
testbfs1() : void
testbfs2() : void
testdfs() : void
testPrim() : void
testKruskal() : void
testDjikstra() : void
testFloydWarshall() : void
DisjointSetTest
- setUpScene1() : void
testFind() : void
testUnion() : void
- sets

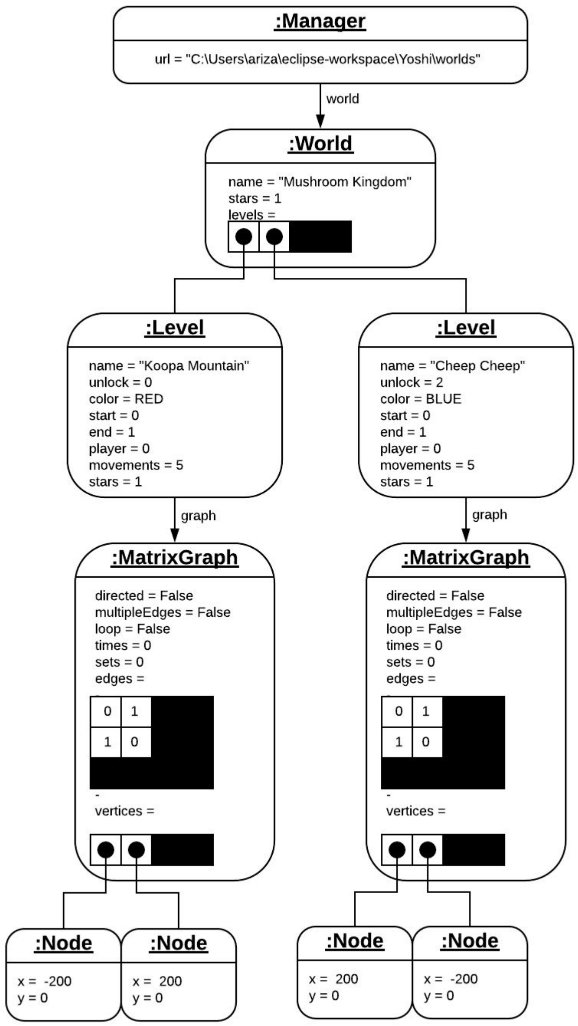


Image Link:
https://cdn.discordapp.com/attachments/588195711414304768/711718453165162527/Blank_Diagram_3.png

Class	**Method**	**Scene**	**Input**	**Result**
MatrixGraph	addVertex	An empty simple graph of Strings.	Add vertices “Johan”, “Esteban” & “Mateo”	The vertices are inside the graph in the correct positions.
MatrixGraph	addEdge	A simple graph of Strings with nodes “Johan”, “Esteban”,	Add edges “Johan”-”Esteban”, “Esteban”-”Mate	The vertices are inside the graph in the correct positions with all

		“Mateo”.	o”.	the possible restrictions.
		A multi directed graph of Strings with nodes “Johan”, “Esteban”, “Mateo”.	Add edges “Johan”→”Esteban”, “Esteban”→”Mateo”, “Mateo”→“Johan” (3 times).	The vertices are inside the graph in the correct positions with all the possible restrictions.
MatrixGraph	removeVertex	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”.	Remove vertex “Esteban”.	The vertex and his matrix references are not in the graph. All the possible restrictions are fine.
MatrixGraph	removeEdge	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”.	Remove edge “Johan”-”Esteban”.	The edge is not in the graph. All the possible restrictions are fine.

		An multi directed graph of Strings with nodes “Johan”, “Esteban”, “Mateo” and edges “Johan”→”Esteban”, “Johan”→”Johan”, “Esteban”→”Mateo”, “Mateo”→“Johan” (3 times).	Remove edges “Johan”→”Johan”, “Mateo”→“Johan”.	The edges are not in the graph. All the possible restrictions are fine.
MatrixGraph	bfs	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”.	Generate the BF tree in node “Johan”.	The BF tree is successfully generated as it should be.
			Generate the BF tree in node “Mateo”.	The BF tree is successfully generated as it should be.
		An multi directed graph of Strings with nodes “Johan”, “Esteban”, “Mateo” and edges “Johan”→”Esteban”, “Johan”→”Johan”, “Esteban”→”Mateo”, “Mateo”→“Johan” (3 times).	Generate the BF tree in node “Esteban”.	The BF tree is successfully generated as it should be.
			Remove the vertex “Mateo”. Then, it generates the BF tree in the node “Esteban”.	The BF tree is successfully generated as it should be.
MatrixGraph	dfs	n simple graph of Strings with nodes “Johan”, “Esteban”,	generates the DF forest.	The DF forest is successfully generated as it should be.

		“Mateo”, “Juan” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”.		
		An multi directed graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”→”Esteban”, “Johan”→”Johan”, “Esteban”→”Mateo” (2 times), “Mateo”→“Johan”.	generates the DF forest.	The DF forest is successfully generated as it should be.
MatrixGraph	prim	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”, “Mateo”-”Juan”, “Juan”-”Johan”, “Johan”-”Mateo”, “Esteban”-”Juan”.	Execute prim.	The generated tree is correct.
		A pseudo graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”-”Esteban”,	Execute prim.	The generated tree is correct.

		“Johan”-”Johan”, “Esteban”-”Mateo” (3 times), “Johan”-”Mateo”,		
MatrixGraph	kruskal	A simple graph of Strings with nodes “a”, “b”, “c”, “d” and edges “a”-”b”, “a”-”c”, “c”-”b”, “b”-”d”.	Execute kruskal.	The generated forest is correct.
		A pseudo graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”-”Esteban”, “Johan”-”Johan”, “Esteban”-”Mateo” (3 times), “Johan”-”Mateo”,	Execute kruskal.	The generated forest is correct.
MatrixGraph	dijkstra	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”, “Mateo”-”Juan”, “Juan”-”Johan”, “Johan”-”Mateo”, “Esteban”-”Juan”.	Execute dijkstra.	The distances are correct.
		An multi directed graph of Strings with nodes “Johan”, “Esteban”,	Execute dijkstra.	The distances are correct.

		“Mateo”, “Juan” and edges “Johan”→”Esteban”, “Johan”→”Johan”, “Esteban”→”Mateo” (2 times), “Mateo”→“Johan”.		
MatrixGraph	floydWarshall	A simple graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”-”Esteban”, “Esteban”-”Mateo”, “Mateo”-”Juan”, “Juan”-”Johan”, “Johan”-”Mateo”, “Esteban”-”Juan”.	Execute floyd warshall.	The matrix of distances is correct.
		An multi directed graph of Strings with nodes “Johan”, “Esteban”, “Mateo”, “Juan” and edges “Johan”→”Esteban”, “Johan”→”Johan”, “Esteban”→”Mateo” (2 times), “Mateo”→“Johan”.	Execute floyd warshall.	The matrix of distances is correct.

Class	Method	Scene	Input	Result
ListGraph	addNode	A simple graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	-	The nodes have been added in order and without errors
ListGraph	addEdge	A simple graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan”	The edges are added without any errors the way they are supposed to
ListGraph	addEdge	A directed graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create these edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan”	The directed edges are added without any errors the way they are supposed to
ListGraph	addEdge	A directed graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create these edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan, “Restrepo” - “Johan”	The multiple edges are added without any errors the way they are supposed to
ListGraph	addEdge	A graph with loops with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create these edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan, “Ariza” - “Ariza”	The edges (including the loop) are added without any errors the way they are supposed to
ListGraph	removeNode	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Remove the nodes “Ariza” and “Mateo”	The nodes have been removed without errors the way they are supposed to
ListGraph	removeEdge	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Remove the edge “Johan” - “Mateo”	The edge has been removed without errors the way it is supposed to

ListGraph	getNode	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	-	There is no problem getting each node individually
ListGraph	getEdges	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan”	The is no problem getting each edge individually
ListGraph	BFS	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the edges: “Ariza” - “Johan”, “Johan” - “Mateo” and “Restrepo” - “Johan”	The function creates a BF Tree in an appropriate manner without any errors
ListGraph	BFS	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the weighted edges: ● “Ariza” - “Ariza” (weight: 3) ● “Ariza” - “Johan” (weight: 2) ● “Ariza” - “Johan” (weight: 1) ● “Johan” - “Mateo” (weight: 1) ● “Johan” - “Restrepo” (weight: 2) ● “Mateo” - “Restrepo” (weight: 2)	The function creates a BF Tree in an appropriate manner without any errors
ListGraph	DFS	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the weighted edges: ● “Ariza” - “Johan” (weight: 2) ● “Johan” - “Mateo” (weight: 1) ● “Ariza” - “Ariza” (weight: 3) ● “Ariza” - “Johan” (weight: 2) ● “Ariza” - “Johan” (weight: 1) ● “Johan” - “Mateo” (weight: 1)	The function creates a DF Tree in an appropriate manner without any errors
ListGraph	Prim	A graph with the names “Ariza”,	Create the weighted edges: ● “Mateo” - “Ariza”	The Minimum Spanning Tree is

		“Johan”, “Mateo” and “Restrepo”	(weight: 2) ● “Mateo” - “Ariza” (weight: 1) ● “Ariza” - “Restrepo” (weight: 3) ● “Ariza” - “Restrepo” (weight: 6) ● “Mateo” - “Restrepo” (weight: 5) ● “Mateo” - “Johan” (weight: 4) ● “Restrepo” - “Johan” (weight: 2)	created in an appropriate manner without any errors
ListGraph	Kruskal	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the weighted edges: ● “Mateo” - “Ariza” (weight: 10) ● “Johan” - “Ariza” (weight: 5) ● “Ariza” - “Restrepo” (weight: 3) ● “Mateo” - “Johan” (weight: 7) ● “Mateo” - “Ariza” (weight: 20) ● “Johan” - “Ariza” (weight: 50) ● “Ariza” - “Restrepo” (weight: 33) ● “Mateo” - “Johan” (weight: 72)	The Minimum Spanning Tree is created in an appropriate manner without any errors
ListGraph	Djikstra	A graph with the names “Ariza”, “Johan”, “Mateo” and “Restrepo”	Create the weighted edges: ● “Mateo” - “Ariza” (weight: 2) ● “Restrepo” - “Ariza” (weight: 3) ● “Ariza” - “Johan” (weight: 5) ● “Mateo” - “Johan” (weight: 2) ● “Mateo” - “Restrepo” (weight: 9)	A graph is returned containing information about the shortest paths between the nodes
ListGraph	FloydWarshall	A graph with the names “Ariza”, “Johan”, “Mateo”	Create the weighted edges: ● “Ariza” - “Mateo” (weight: -2)	The matrix is created with the appropriate

		and “Restrepo”	“Mateo” - “Restrepo” (weight: 2) “Johan” - “Mateo” (weight: 3) “Johan” - “Ariza” (weight: 4) “Restrepo” - “Johan” (weight: -1)	values and without any errors

Clas s	Method	Scene	Input	Result
Level	move	A directed graph named “graph” is created with three nodes: - Node 1: x = 1, y = 1 - Node 2: x = 2, y = 2 - Node 3: x = 3, y = 3 The directed graph also has three edges: - Edge 1: Node 1 - Node 2 (weight: 1) - Edge 2: Node 2 - Node 3 (weight: 2) - Edge 3: Node 3 - Node 2 (weight: 1) A level with created with the following attributes: - Name: “Johan” - Unlock: 2 - Color: Red - Start: 0 - End: 2 - Graph: The graph - Stars: 0	The player in the level is moved repeatedly until there are no more moves left	The player moves appropriately without errors until he has no more moves left. Once he has no more moves left, he is not allowed to move again.
Level	win	A directed graph named “graph” is created with three nodes: - Node 1: x = 1, y = 1 - Node 2: x = 2, y = 2 - Node 3: x = 3, y = 3 The directed graph also has three edges: - Edge 1: Node 1 -	The player moves until the objective	When the player reaches the objective he wins

		Node 2 (weight: 1) ● Edge 2: Node 2 - Node 3 (weight: 2) ● Edge 3: Node 3 - Node 2 (weight: 1) A level with created with the following attributes: ● Name: “Johan” ● Unlock: 2 ● Color: Red ● Start: 0 ● End: 2 ● Graph: The graph ● Stars: 0		
Level	lose	A directed graph named “graph” is created with three nodes: ● Node 1: x = 1, y = 1 ● Node 2: x = 2, y = 2 ● Node 3: x = 3, y = 3 The directed graph also has three edges: ● Edge 1: Node 1 - Node 2 (weight: 1) ● Edge 2: Node 2 - Node 3 (weight: 2) ● Edge 3: Node 3 - Node 2 (weight: 1) A level with created with the following attributes: ● Name: “Johan” ● Unlock: 2 ● Color: Red ● Start: 0 ● End: 2 ● Graph: The graph ● Stars: 0	The player moves until he has no more moves left	When the player has no more moves left he loses
Level	starsEarned	A directed graph named “graph” is created with three nodes: ● Node 1: x = 0, y = 0 ● Node 2: x = 1, y = 1 ● Node 3: x = 2, y = 2 ● Node 4: x = 3, y = 3 ● Node 5: x = 4, y = 4	The player moves and collects stars	The players earns the collected stars appropriately without any errors

		The directed graph also has three edges: ● Edge 1: Node 1 - Node 2 (weight: 1) ● Edge 2: Node 1 - Node 3 (weight: 1) ● Edge 3: Node 1 - Node 4 (weight: 1) ● Edge 4: Node 2 - Node 5 (weight: 2) ● Edge 5: Node 3 - Node 5 (weight: 3) ● Edge 6: Node 4 - Node 5 (weight: 6) A level with created with the following attributes: ● Name: “Johan” ● Unlock: 2 ● Color: Red ● Start: 0 ● End: 4 ● Graph: The graph ● Stars: 0		
Level	starsInGame	A directed graph named “graph” is created with three nodes: ● Node 1: x = 0, y = 0 ● Node 2: x = 1, y = 1 ● Node 3: x = 2, y = 2 ● Node 4: x = 3, y = 3 ● Node 5: x = 4, y = 4 The directed graph also has three edges: ● Edge 1: Node 1 - Node 2 (weight: 1) ● Edge 2: Node 1 - Node 3 (weight: 1) ● Edge 3: Node 1 - Node 4 (weight: 1) ● Edge 4: Node 2 - Node 5 (weight: 2) ● Edge 5: Node 3 - Node 5 (weight: 3) ● Edge 6: Node 4 - Node 5 (weight: 6) A level with created with	The player moves and collects stars	The stars in the game are the appropriate number and there are no errors

		the following attributes: ● Name: “Johan” ● Unlock: 2 ● Color: Red ● Start: 0 ● End: 4 ● Graph: The graph ● Stars: 0		
Level	checkPlayable	A graph is created with two vertices: ● Vertex 1: x =1, y = 1 ● Vertex 2: x = 2, y = 2	Edges are added to the graph	The graph is playable once the edges are added

Class	**Method**	**Scene**	**Input**	**Result**
Manager	importWorld	Create a world with 3 levels. The characteristics of the levels are: Level #1: ● Name: “Johan” ● Unlock: 0 ● Color: Green ● Start: 0 ● End: 4 ● Stars: 0 Level #2: ● Name: “Mateo” ● Unlock: 2 ● Color: Red ● Start: 0 ● End: 3 ● Stars: 0 Level #3: ● Name: “Esteban” ● Unlock: 4 ● Color: Yellow ● Start: 0 ● End: 2 ● Stars: 0	The world is loaded	The world, the levels and all the level’s components are loaded appropriately without any errors

Class	Method	Scene	Input	Result
World	add	Create an empty world with the following characteristics: ● Name: World ● Stars: 0	Levels are added to the world	The levels are added successfully to the world and their values are appropriate
World	searchLevel	Create a world with the following characteristics: ● Name: World ● Stars: 0 It contains 2 levels named “Johan” and “Esteban”	The levels “Esteban”, “Johan” and “Mateo” are searched	The levels “Esteban” and “Johan” are found, but the level “Mateo” is not found because it does not exist in the world

Class	Method	Scene	Input	Result
DisjointSet	find	Crate a disjoint set and make 4 sets: ● Set #1: {a} ● Set #2: {b} ● Set #3: {c} ● Set #4: {d}	-	All of the sets are found appropriately without errors
DisjointSet	union	Crate a disjoint set and make 4 sets: ● Set #1: {a} ● Set #2: {b} ● Set #3: {c} ● Set #4: {d}	-	The sets can be united without any errors or strange behaviors