Engineering Method

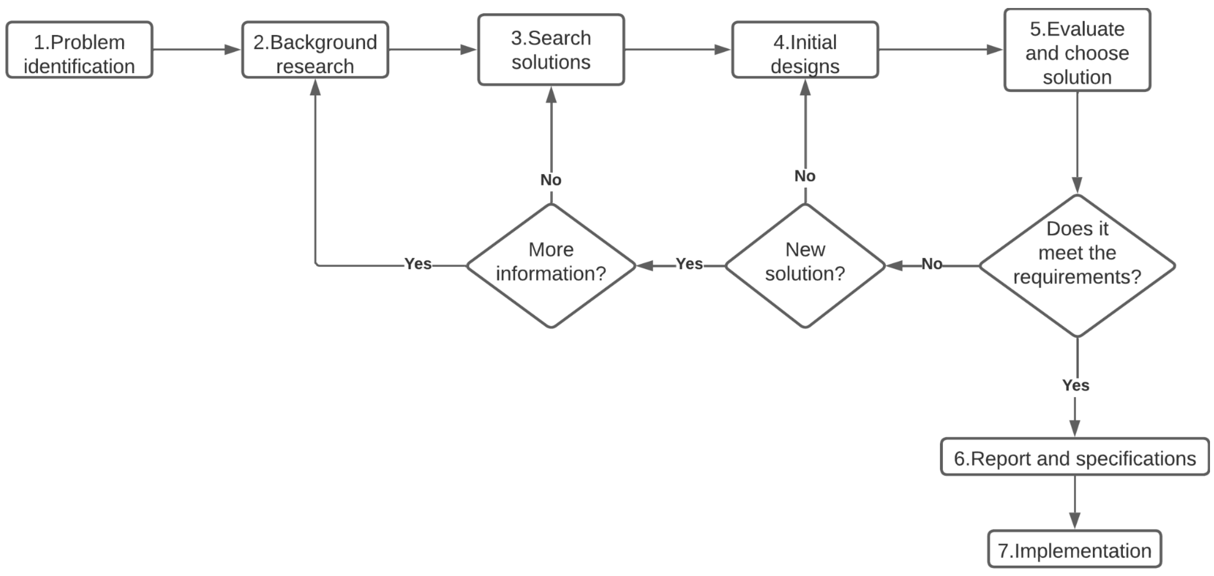
John Maurice Hortua
Santiago Arboleda Velasco
Samuel Viviescas Carrillo

Problem Context

Students enrolled in the Algorithms and Discrete Structures course at Icesi University are facing challenges in understanding and applying graph theory concepts. The difficulty arises from the abstract nature of these concepts, making their application less intuitive. Additionally, students are confronted with a task where they are required to design and implement a game with a minimum of 50 vertices and 50 edges. The interesting aspect is that students have the opportunity to propose the game themselves.

Solution development

In order to solve this problem, the engineering method was chosen for developing the solution, following a systematic approach that aligns with the stated problem. The following flowchart was defined, and we will follow its steps in the process of the development of the solution.



Step 1. Problem identification

<u>Needs assessments</u>

- The solution must ensure the use of 2 algorithms, whether they are Graph Traversals (BFS, DFS), Minimum Weight Paths (Dijkstra, Floyd-Warshall), or Minimum Spanning Tree (MST) (Prim, Kruskal).

- The solution to the problem must ensure the automatic switch to at least 2 different versions of algorithm implementations.

- The solution to the problem must ensure a user-friendly graphical interface.

- The solution should provide assistance and instructions to the user.

Problem definition

Students in the Algorithms and Discrete Structures course at Icesi University must design and implement a game with a minimum of 50 vertices and 50 edges, with the added twist that students can propose the game themselves.

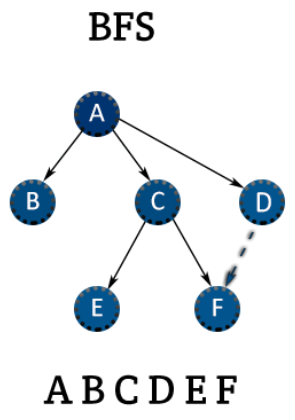
Step 2. Background research

Definitions

BFS

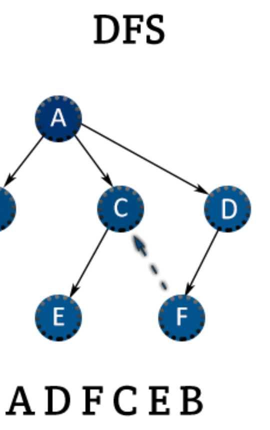
Is a method for graphing data, search trees, and traversing structures. The method visits & marks all critical nodes in a network in an exact breadthwise manner. This algorithm chooses a single node (the beginning or origin point) in a network and visits all nodes near the chosen node.

After visiting and marking the beginning node, the algorithm goes on to the next uninhabited nodes and analyzes them. All nodes are indicated once they have been visited. These cycles continue until all nodes in the graph have been visited and marked correctly. BFS is an abbreviation for Breadth-first search.



DFS

Is a depth-first search strategy that may be used to locate or explore graphs or trees. Before backtracking, the algorithm begins at the tree's root and explores each path. When an iteration reaches a dead end, a stack information structure is utilized to remember, identify the next vertex, and begin a search. DFS's complete form is a depth-first search.

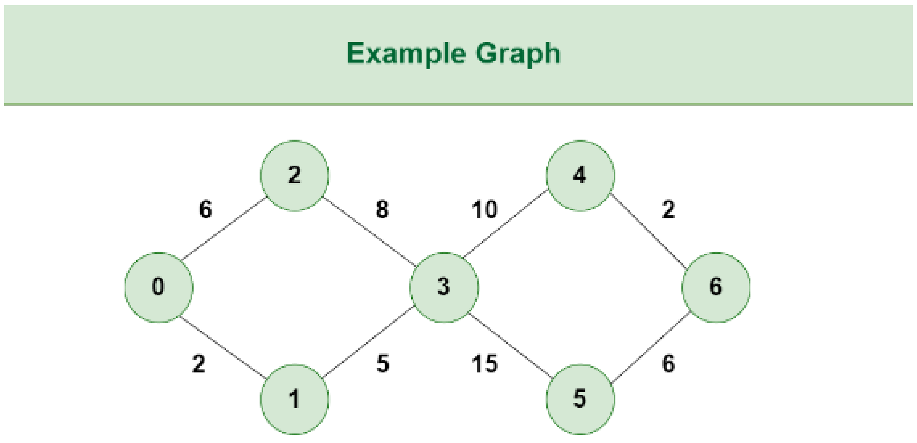


Dijkstra

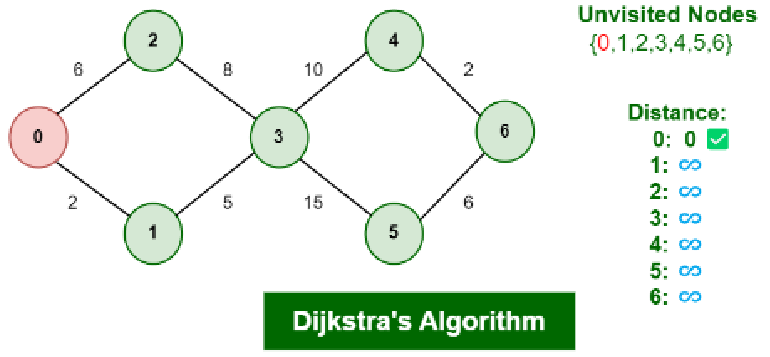
Is a popular algorithm for solving many single-source shortest path problems having non-negative edge weight in the graphs i.e., it is to find the shortest distance between two vertices on a graph. It was conceived by Dutch computer scientist Edsger W. Dijkstra in 1956.

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source.

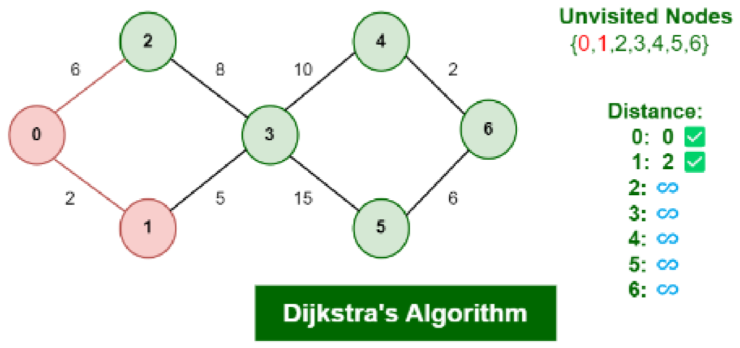
It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited.



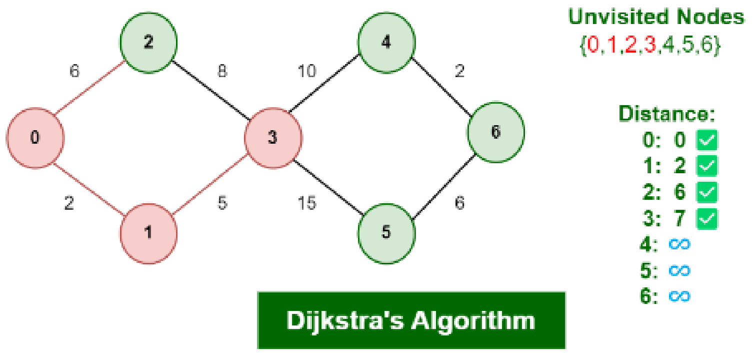
STEP 1 Start from Node 0 and mark Node 0 as Visited and check for adjacent nodes

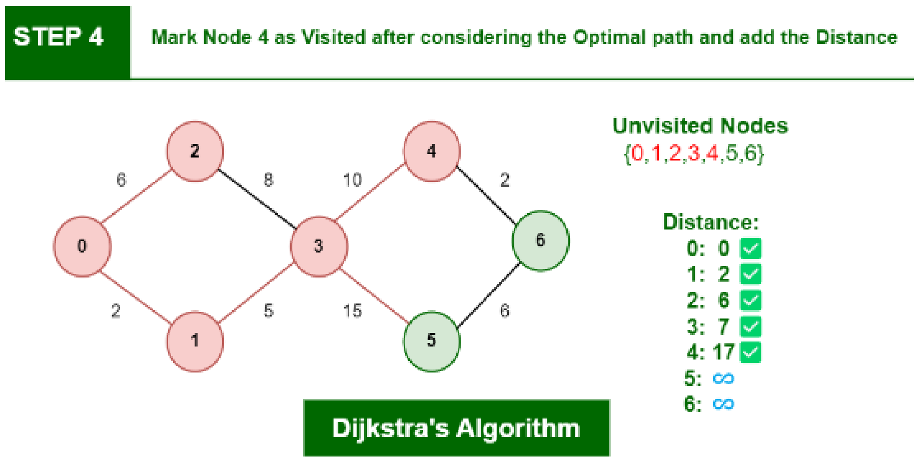


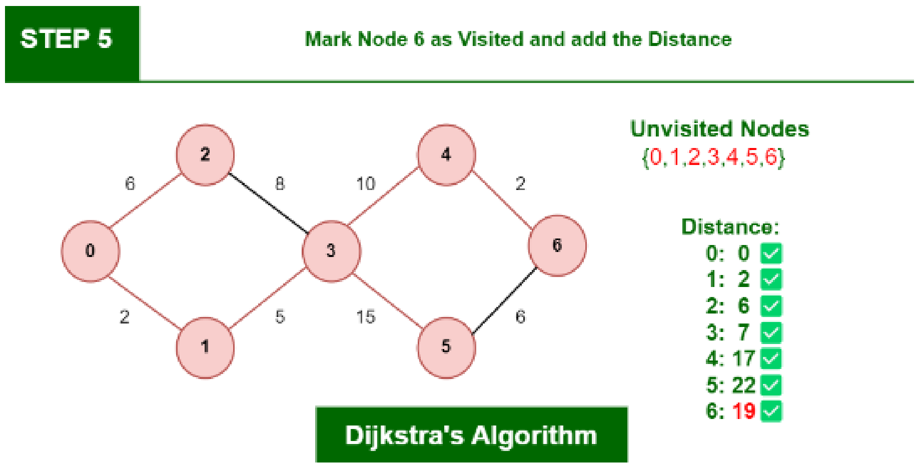
STEP 2 Mark Node 1 as Visited and add the Distance



STEP 3 Mark Node 3 as Visited after considering the Optimal path and add the Distance

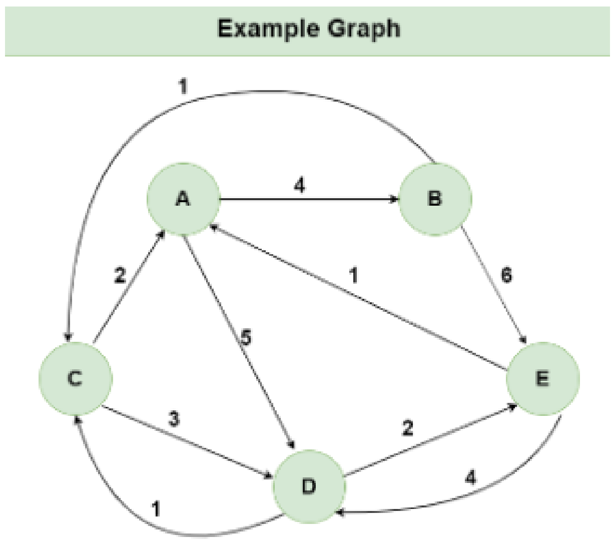






Floyd-Warshall

Is a fundamental algorithm in computer science and graph theory. It is used to find the shortest paths between all pairs of nodes in a weighted graph. This algorithm is highly efficient and can handle graphs with both positive and negative edge weights, making it a versatile tool for solving a wide range of network and connectivity problems.



Step1: Initializing Distance[ ][ ] using the Input Graph

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

Step 2: Using Node A as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][A] + Distance[A][j])

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	?	?	?	?
C	2	?	?	?	?
D	∞	?	?	?	?
E	1	?	?	?	?

$\longrightarrow$

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	∞	4	0

Step 3: Using Node B as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][B] + Distance[B][j])

	A	B	C	D	E
A	?	4	?	?	?
B	∞	0	1	∞	6
C	?	6	?	?	?
D	?	∞	?	?	?
E	?	5	?	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	6	4	0

Step 4: Using Node C as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][C] + Distance[C][j])

	A	B	C	D	E
A	?	?	5	?	?
B	?	?	1	?	?
C	2	6	0	3	12
D	?	?	1	?	?
E	?	?	6	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0

Step 5: Using Node D as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][D] + Distance[D][j])

	A	B	C	D	E
A	?	?	?	5	?
B	?	?	?	4	?
C	?	?	?	3	?
D	3	7	1	0	2
E	?	?	?	4	?

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 6: Using Node E as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][E] + Distance[E][j])

	A	B	C	D	E
A	?	?	?	?	7
B	?	?	?	?	6
C	?	?	?	?	5
D	?	?	?	?	2
E	1	5	5	4	0

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

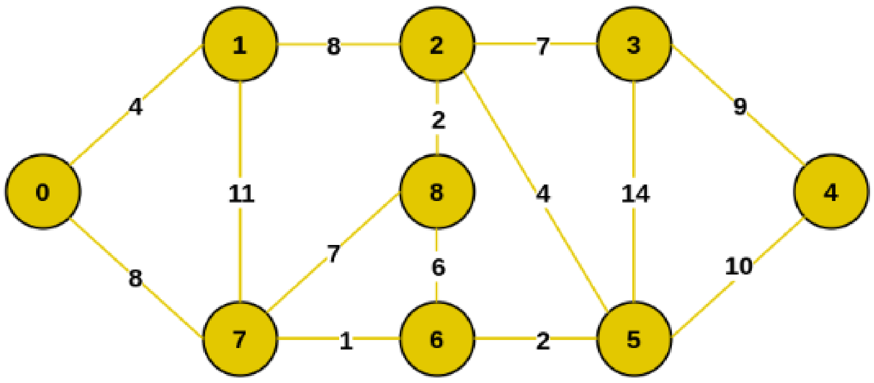
Step 7: Return Distance[ ][ ] matrix as the result

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

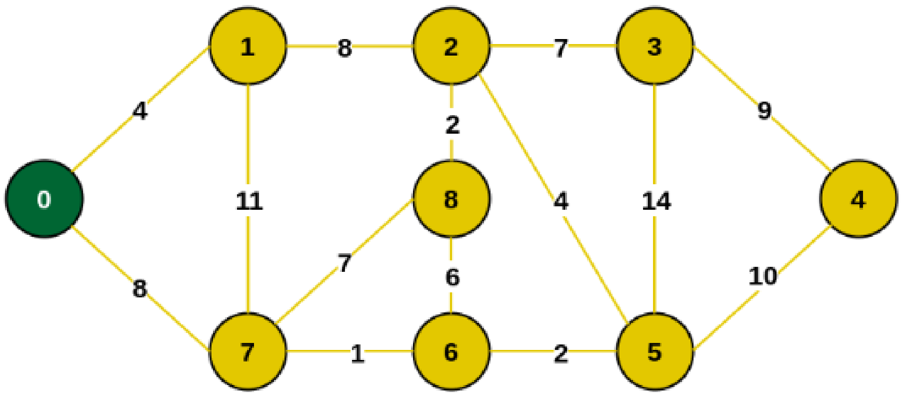
Prim

This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

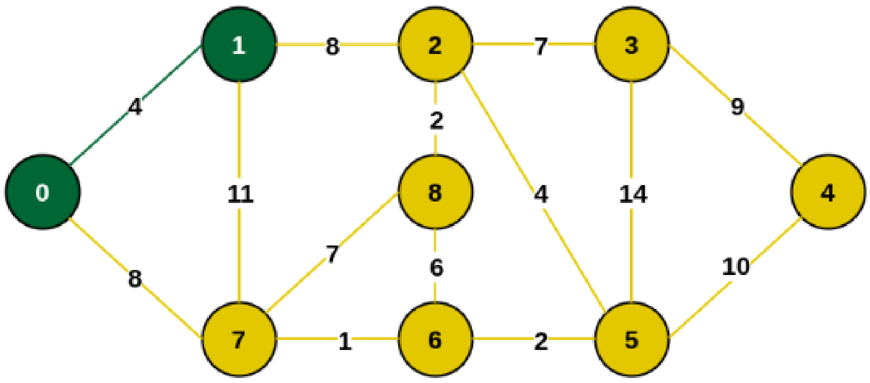
The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.



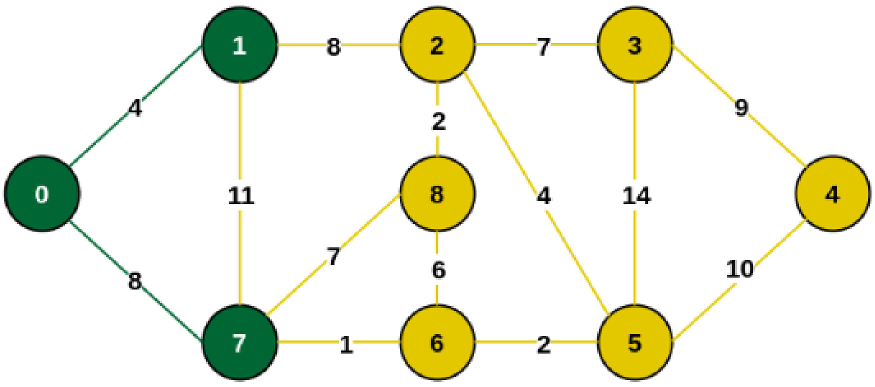
Example of a Graph



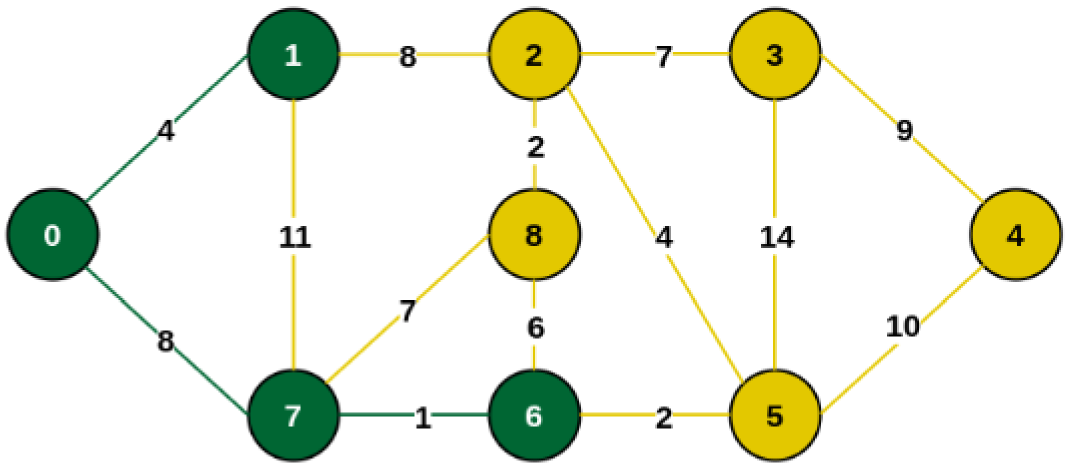
Select an arbitrary starting vertex. Here we have selected 0



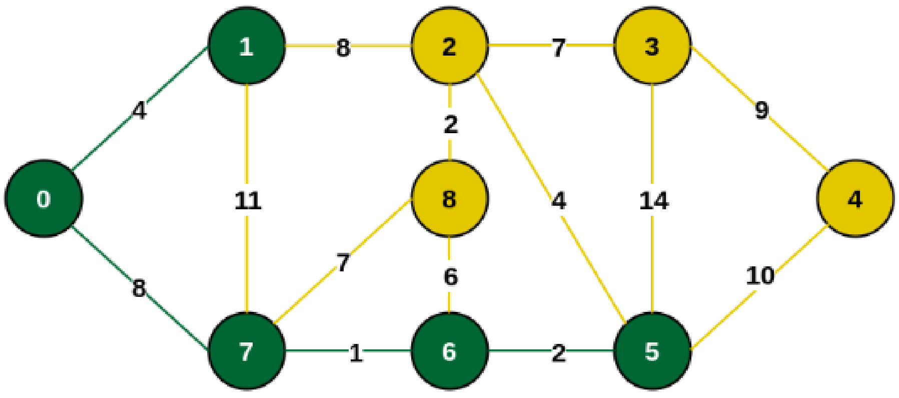
Minimum weighted edge from MST to other vertices is 0-1 with weight 4



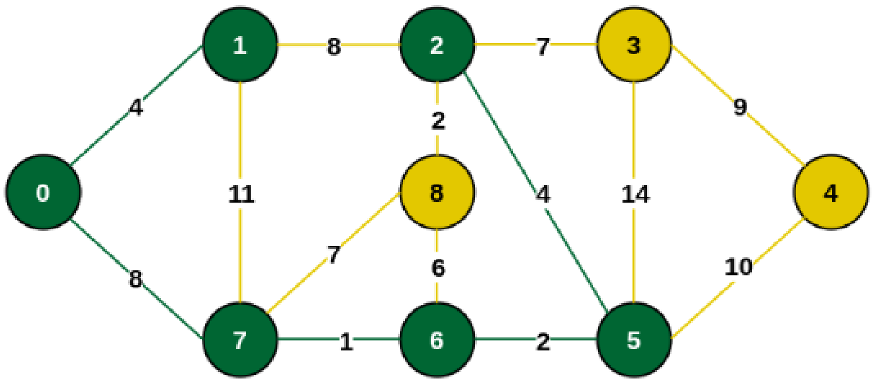
Minimum weighted edge from MST to other vertices is 0-7 with weight 8



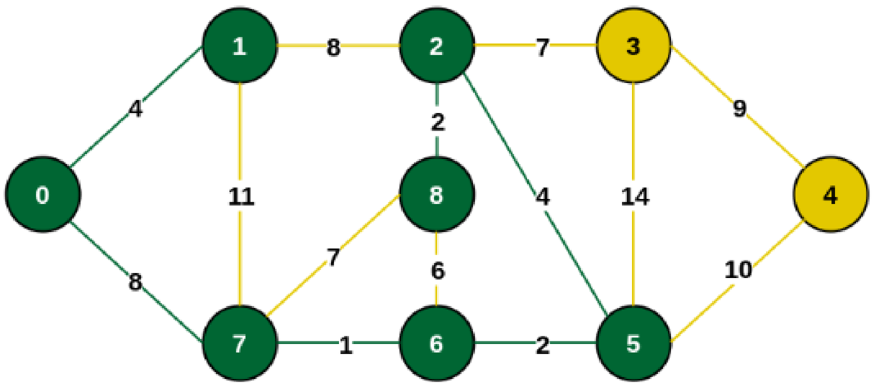
Minimum weighted edge from MST to other vertices is 7-6 with weight 1



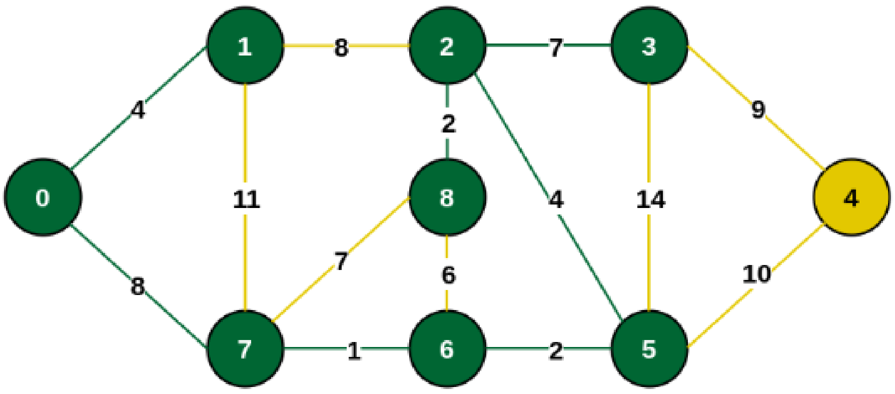
Minimum weighted edge from MST to other vertices is 6-5 with weight 2



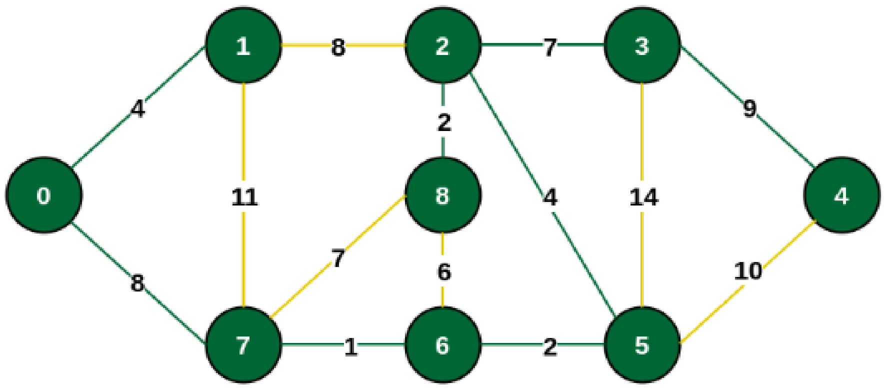
Minimum weighted edge from MST to other vertices is 5-2 with weight 4



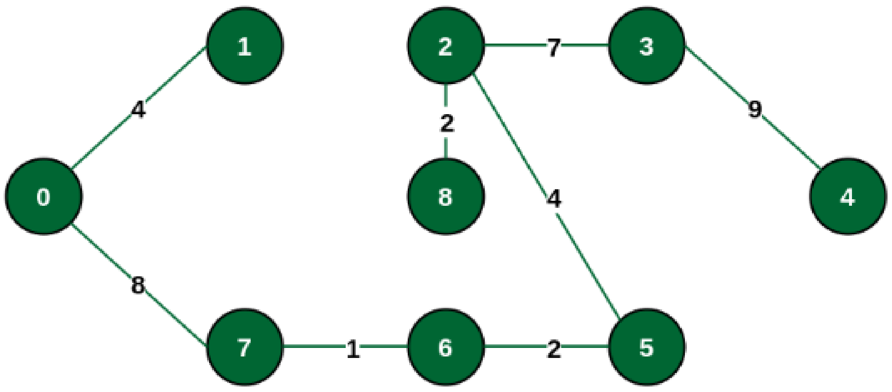
Minimum weighted edge from MST to other vertices is 2-8 with weight 2



Minimum weighted edge from MST to other vertices is 2-3 with weight 7



Minimum weighted edge from MST to other vertices is 3-4 with weight 9

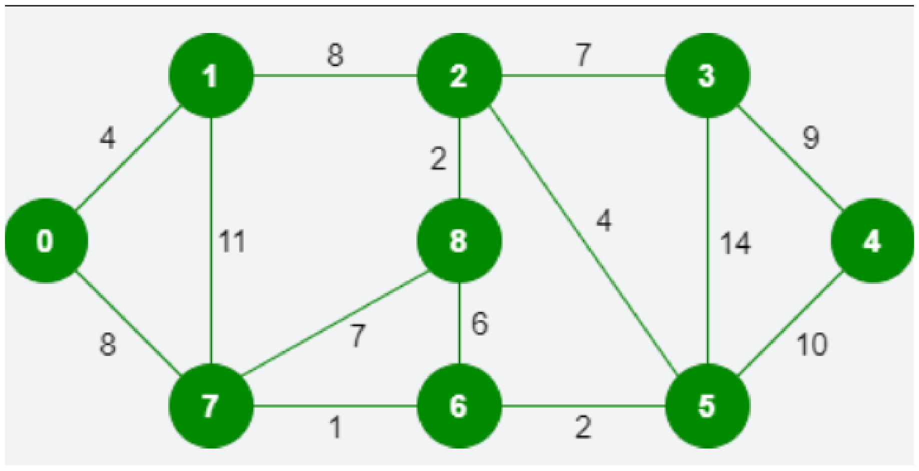


The final structure of MST

Kruskal

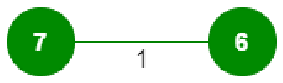
In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution.

Input Graph:



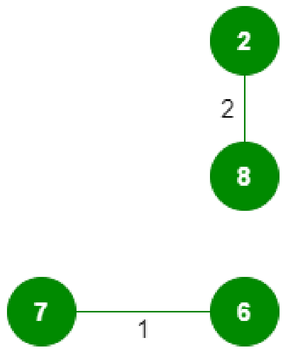
Step 1: Pick edge 7-6. No cycle is formed, include it.

Step 1 Add edge 7-6 in the MST



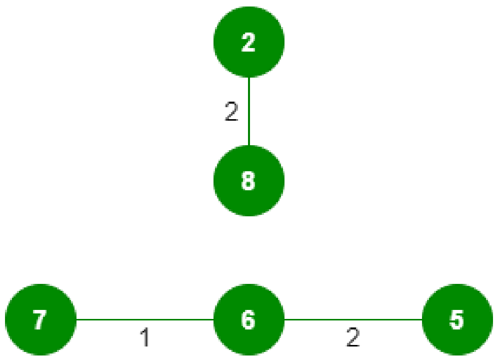
Step 2: Pick edge 8-2. No cycle is formed, include it.

Step 2 Add edge 8-2 in the MST

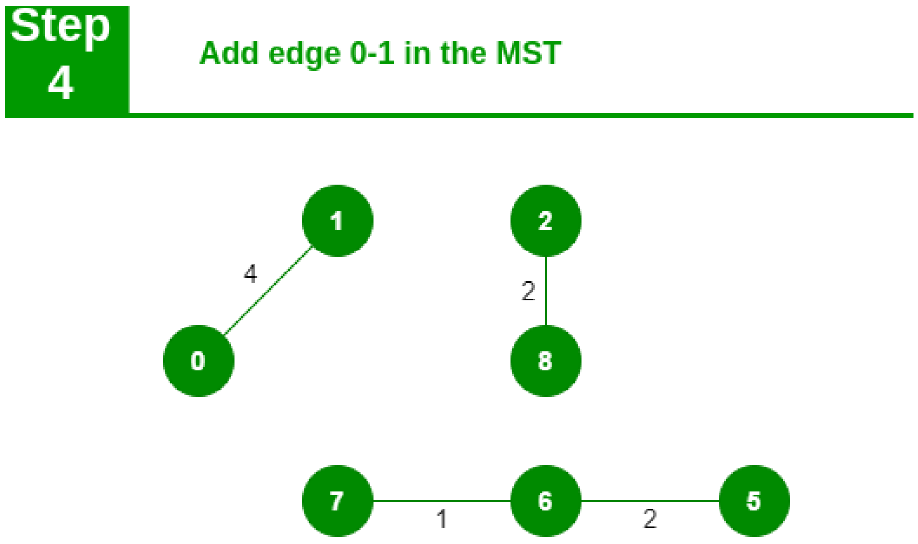


Step 3: Pick edge 6-5. No cycle is formed, include it.

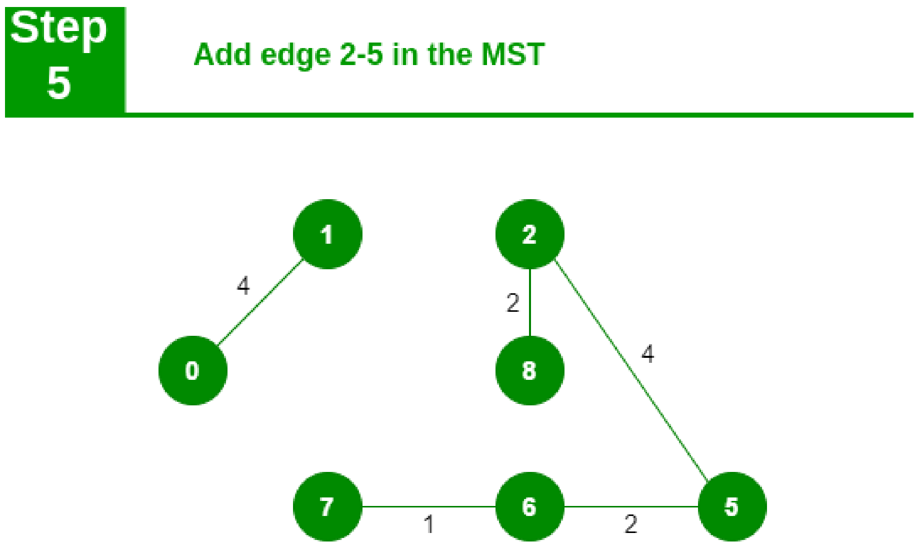
Step 3 Add edge 6-5 in the MST



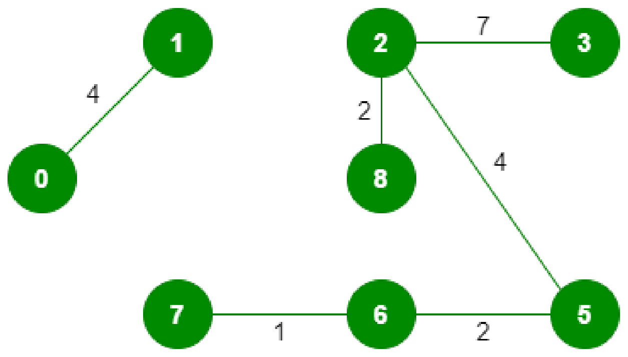
Step 4: Pick edge 0-1. No cycle is formed, include it.



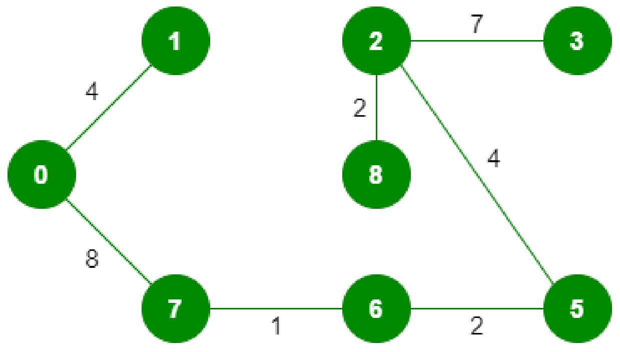
Step 5: Pick edge 2-5. No cycle is formed, include it.



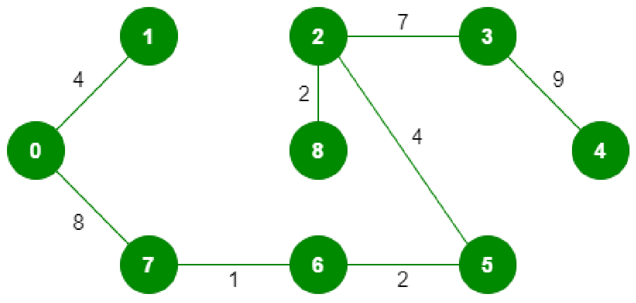
Step 6: Pick edge 8-6. Since including this edge results in the cycle, discard it.
Pick edge 2-3: No cycle is formed, include it.



Step 7: Pick edge 7-8. Since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed, include it.

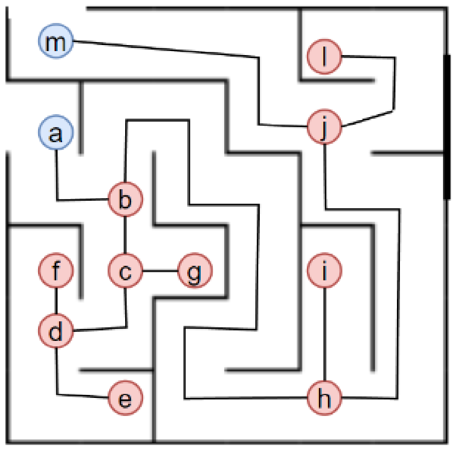


Step 8: Pick edge 1-2. Since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed, include it.



Step 3. Search solutions

Alternative 1: Maze with Graphs:



Conceptual Design:

In this maze game, the labyrinth is represented as a graph where nodes correspond to intersections and edges represent paths. The goal is for players to navigate from the entrance to the exit by traversing the graph. Each intersection (node) presents a decision point, and players must choose the correct path to advance.

Implementation:

Graph Representation:
Design a graph where each intersection is a node, and each path between intersections is an edge. The graph can be implemented as an adjacency list or matrix.

Node Content:
Each node may contain information such as clues, obstacles, or challenges. Players must make decisions at each intersection based on this information.

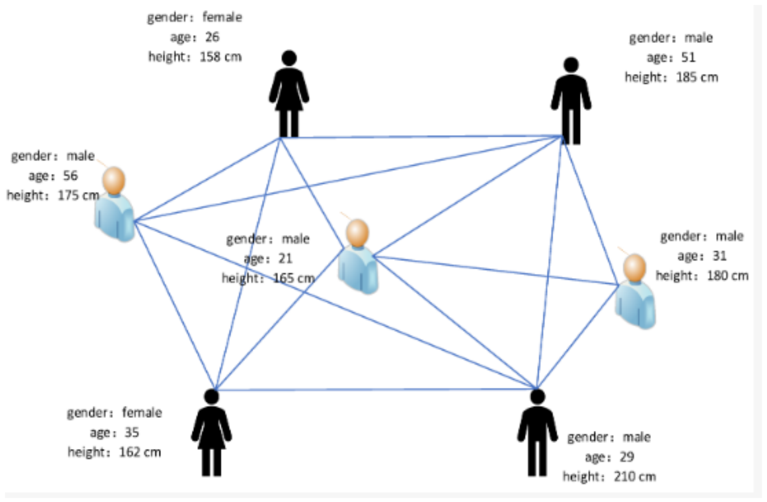
Player Movement:
Players move through the maze by selecting edges that connect nodes. Incorrect choices may lead to dead ends or setbacks, while correct choices bring them closer to the exit.

Challenges and Puzzles:
Introduce challenges at certain nodes, such as puzzles that must be solved to proceed. These puzzles can be related to graph theory concepts, adding an educational element.

Game Progression:
As players progress, the maze complexity can increase. Shortcuts, hidden paths, or teleportation nodes can be introduced to keep the game engaging.

Goal:
The ultimate goal is for players to reach the exit node. The game can have multiple levels with increasing difficulty, each represented by a different maze graph

<u>Alternative 2: Social Network Simulator:</u>



Conceptual Design:

In this simulator, the social network is modeled as a graph, where nodes represent individuals and edges denote relationships. Players engage in activities to influence and shape the dynamics of the network.

Implementation:

Graph Representation:
Represent individuals as nodes and relationships (friendships, alliances, etc.) as edges. Attributes like popularity, influence, and interests can be associated with each node.

Player Actions:
Players can perform actions such as making new connections, influencing others, or participating in events. Each action affects the social graph, altering relationships and attributes.

Goals and Objectives:

Assign goals to players, such as becoming the most influential individual, creating the largest social group, or initiating and resolving conflicts within the network.

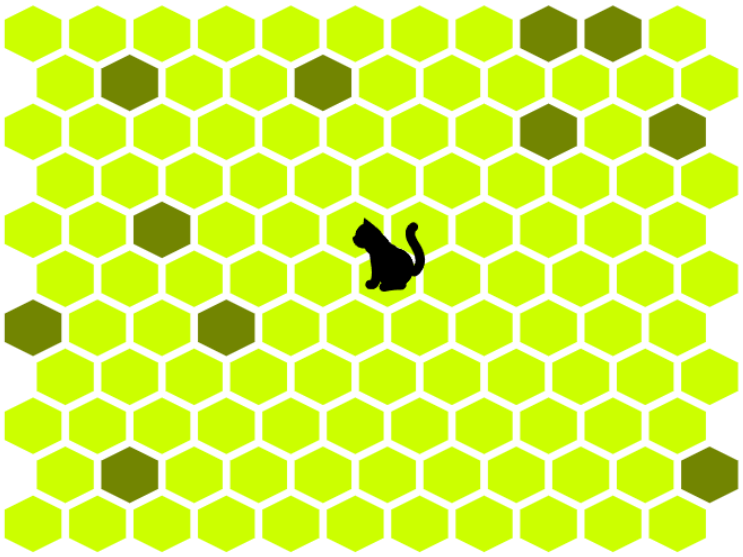
Dynamic Events:
Introduce dynamic events that impact the entire network, such as trending topics, rivalries, or external influences. Players must adapt their strategies to these changes.

Social Challenges:
Implement challenges that players can solve using their understanding of the social graph. For example, resolving conflicts, mediating disputes, or fostering positive interactions.

Feedback System:
Provide feedback to players on the consequences of their actions. The social graph evolves based on their decisions, affecting the overall dynamics of the simulated network.

Simulation Depth:
Increase the complexity by simulating various social aspects like trust, loyalty, and evolving interests. This adds depth to the gameplay and encourages strategic thinking.

<u>Alternative 3: Trap the cat:</u>



Conceptual Design:

In Trap the Cat, the player aims to strategically encircle a cat that moves across a grid-based board. The challenge lies in selecting adjacent cells to gradually limit the cat's movement until it's trapped.

Implementation:

Grid Representation:
Design a grid-based board where each cell represents a possible position for the cat. The cat can move freely across navigable cells.

Cat Movement:
The cat moves from one cell to an adjacent, navigable cell during its turn. The movement can follow a specific pattern.

Player Actions:
The player can select cells to render impassable for the cat. These chosen cells gradually restrict the cat's movement options. Player strategize to predict the cat's next move.

Trap Formation:
Player must create a perimeter around the cat. The objective is to limit the cat's accessible cells until it has no valid moves left, leading to its capture.

Winning Conditions:
The game can have different winning conditions, such as capturing the cat within a certain number of turns or surviving a set number of rounds without the cat escaping.

Step 4. Transition from Ideas to Initial designs

All the games defined earlier could be modeled using graphs, where each of them has a different objective and functioning. Upon conducting a detailed review of all of the alternatives, we have the following aspects:

Alternative 1:
- The maze utilizes a graph where nodes represent intersections, and edges represent paths.
- Players navigate through the maze by traversing nodes and edges. The graph structure enables the creation of complex mazes with decision points.
- The graph structure facilitates the incorporation of diverse content at each node, contributing to the maze's complexity and providing a dynamic gameplay experience.
- The graph dictates the possible movements, allowing for varied paths and decision-making at each intersection.

Alternative 2:
- The social network is modeled as a graph where nodes represent individuals, and edges denote relationships.
- The graph structure facilitates the simulation of social interactions, allowing players to influence and shape the dynamics of the network.
- Actions directly impact the graph by modifying relationships and attributes associated with nodes, showcasing the dynamic nature of social connections.
- The graph adapts to external events, affecting relationships and attributes across nodes, creating a realistic and evolving social environment.

Alternative 3:
- The game is played on a grid where each cell represents a possible position for the cat.
- The grid serves as a discrete graph, where cells are nodes, and adjacency defines possible cat movements.
- The graph structure dictates the cat's movement options, allowing for dynamic and strategic gameplay.
- Player actions modify the graph by blocking specific paths, gradually limiting the cat's available movements and leading to its eventual capture.

Step 5. Evaluate and choose solution

Criteria:
We are going to define a series of criteria with which we will evaluate the solution alternatives. Based on the results obtained, we will choose the one that best satisfies the needs of the problem presented.

Evaluation:

- Criterion A. Solution Accuracy. The alternative provides a solution.
    - [2] Exact
    - [1] Approximate
- Criterion B. Use of graphs.
    - [2] Yes
    - [1] No
- Criterion C. Programming Skills Development.
    - [2] High
    - [1] Low
- Criterion D. Enjoyable game.
    - [3] High
    - [2] Medium
    - [1] Low

Alternatives	Criterion A	Criterion B	Criterion C	Criterion D	Total
Alternative 1	Exact 2	Yes 2	High 2	Low 1	7
Alternative 2	Exact 2	Yes 2	High 2	Medium 2	8
Alternative 3	Exact 2	Yes 2	High 2	High 3	9

Solution selection:

According to the previous evaluation, Alternative 3 should be selected as it obtained the highest score based on the defined criteria. Since we're developing a game, it is going to be the most entertaining game of them all, which is trap the cat.

Step 6. Reports and specifications

Problem Specification

ADTs

ADT Graph
Graph = (V, E) V = a set whose elements are the vertices of the graph E = a set whose elements are the edge
inv: $\forall \mathcal{V} \in V$: $\mathcal{V}$ is a unique vertex $\forall e \in E$: $e = (\mathcal{V}1, \mathcal{V}2)$ where $\mathcal{V}1, \mathcal{V}2 \in V \land \mathcal{V}1 \neq \mathcal{V}2$ $\forall (\mathcal{V}1, \mathcal{V}2) \in E, (\mathcal{V}2, \mathcal{V}1) \in E$
Basic operations: - **getVertices** Graph -> Vertex - **getAdjacentVertices** Graph -> Vertex - **setPolygon** Graph

Step 7. Implementation

Solution implementation in Java:

List of task to implement:

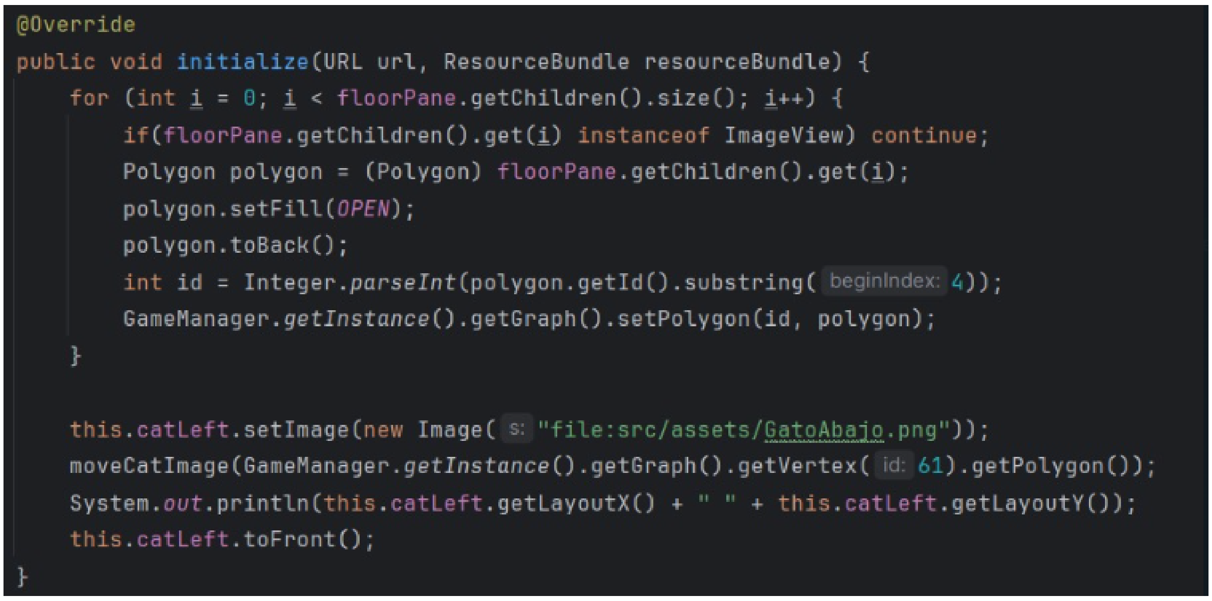
- Create board
- Cat movement
- Player actions

-

Subroutine specifications

Name:	Create board
Description:	allows to create a board
Input:	none
Output	Board, board with polygons

Implementation

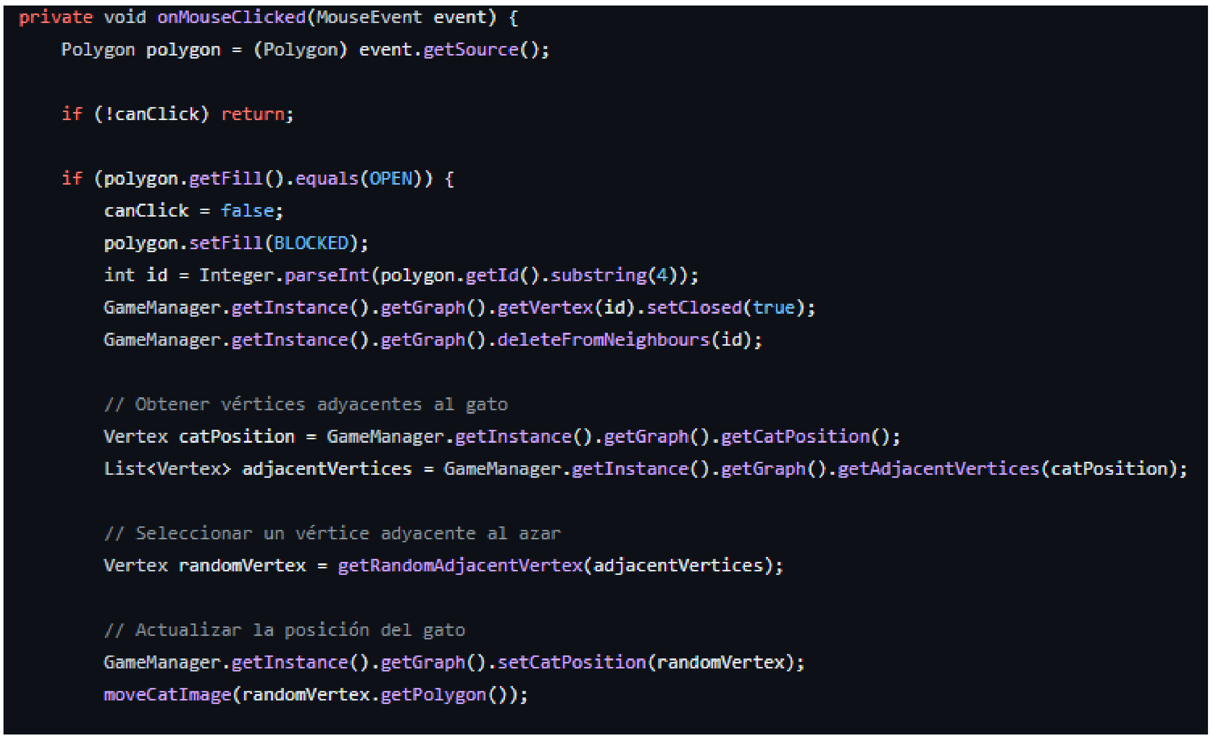


```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    for (int i = 0; i < floorPane.getChildren().size(); i++) {
        if(floorPane.getChildren().get(i) instanceof ImageView) continue;
        Polygon polygon = (Polygon) floorPane.getChildren().get(i);
        polygon.setFill(OPEN);
        polygon.toBack();
        int id = Integer.parseInt(polygon.getId().substring( beginIndex: 4));
        GameManager.getInstance().getGraph().setPolygon(id, polygon);
    }

    this.catLeft.setImage(new Image( s: "file:src/assets/GatoAbajo.png"));
    moveCatImage(GameManager.getInstance().getGraph().getVertex( id: 61).getPolygon());
    System.out.println(this.catLeft.getLayoutX() + " " + this.catLeft.getLayoutY());
    this.catLeft.toFront();
}
```

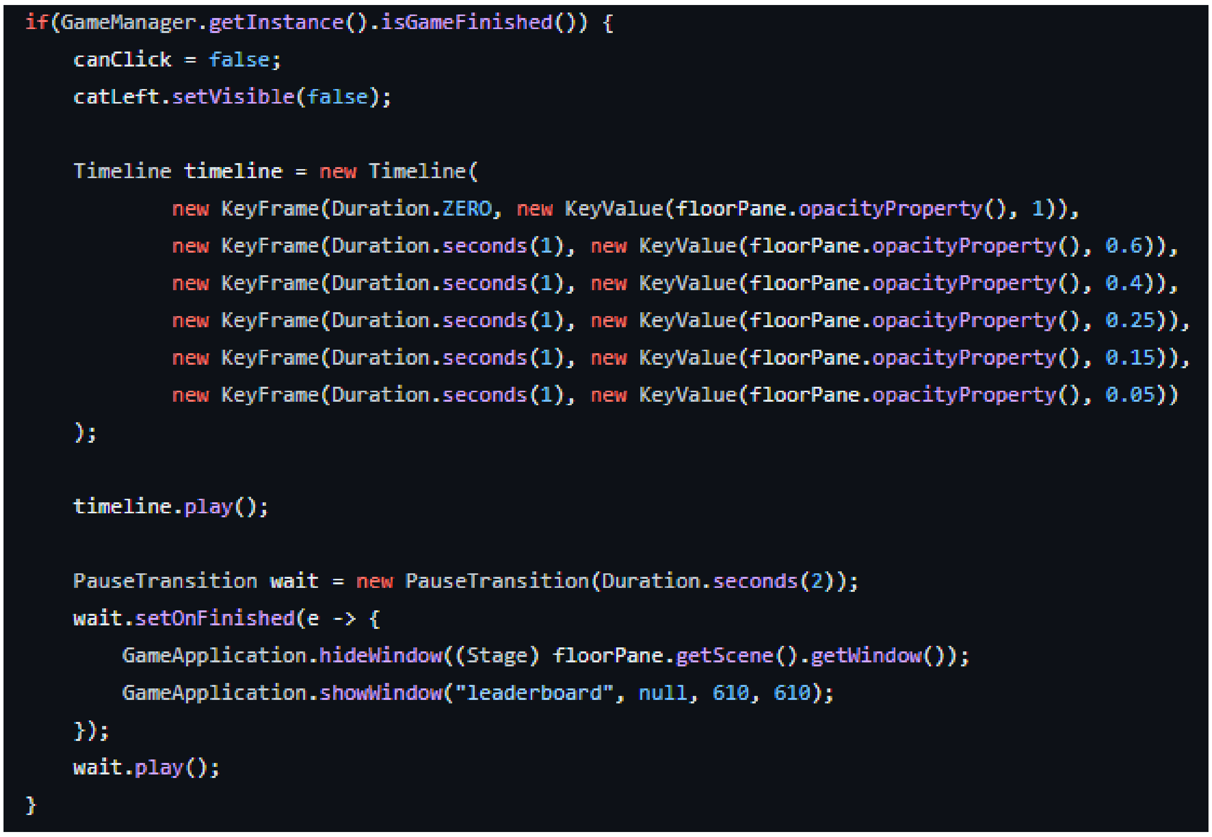
Name:	Cat movement
Description:	allows the cat to move around the board
Input:	none
Output	movement

Name:	Player action
Description:	allows the player to interact with the board
Input:	player click
Output	Board change, a polygon changes color and is no longer available for the cat to jump

Implementation cat movement and player action



```java
private void onMouseClicked(MouseEvent event) {
    Polygon polygon = (Polygon) event.getSource();

    if (!canClick) return;

    if (polygon.getFill().equals(OPEN)) {
        canClick = false;
        polygon.setFill(BLOCKED);
        int id = Integer.parseInt(polygon.getId().substring(4));
        GameManager.getInstance().getGraph().getVertex(id).setClosed(true);
        GameManager.getInstance().getGraph().deleteFromNeighbours(id);

        // Obtener vértices adyacentes al gato
        Vertex catPosition = GameManager.getInstance().getGraph().getCatPosition();
        List<Vertex> adjacentVertices = GameManager.getInstance().getGraph().getAdjacentVertices(catPosition);

        // Seleccionar un vértice adyacente al azar
        Vertex randomVertex = getRandomAdjacentVertex(adjacentVertices);

        // Actualizar la posición del gato
        GameManager.getInstance().getGraph().setCatPosition(randomVertex);
        moveCatImage(randomVertex.getPolygon());
```



```java
if(GameManager.getInstance().isGameFinished()) {
    canClick = false;
    catLeft.setVisible(false);

    Timeline timeline = new Timeline(
            new KeyFrame(Duration.ZERO, new KeyValue(floorPane.opacityProperty(), 1)),
            new KeyFrame(Duration.seconds(1), new KeyValue(floorPane.opacityProperty(), 0.6)),
            new KeyFrame(Duration.seconds(1), new KeyValue(floorPane.opacityProperty(), 0.4)),
            new KeyFrame(Duration.seconds(1), new KeyValue(floorPane.opacityProperty(), 0.25)),
            new KeyFrame(Duration.seconds(1), new KeyValue(floorPane.opacityProperty(), 0.15)),
            new KeyFrame(Duration.seconds(1), new KeyValue(floorPane.opacityProperty(), 0.05))
    );

    timeline.play();

    PauseTransition wait = new PauseTransition(Duration.seconds(2));
    wait.setOnFinished(e -> {
        GameApplication.hideWindow((Stage) floorPane.getScene().getWindow());
        GameApplication.showWindow("leaderboard", null, 610, 610);
    });
    wait.play();
}
```

References

Difference Between BFS And DFS. (n.d.). Retrieved from https://unacademy.com/content/gate-cse-it/difference-between-bfs-and-dfs/

Floyd Warshall Algorithm - GeeksforGeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

Kruskal's Minimum Spanning Tree (MST) Algorithm - GeeksforGeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Prim's Algorithm for Minimum Spanning Tree (MST) - GeeksforGeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm - GeeksforGeeks. (n.d.). Retrieved from https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/