

INTEGRATIVE TASK 2



DAVIDE FLAMINI CAZARAN - A00381665 - SYSTEMS ENGINEERING
NICOLÁS CUÉLLAR MOLINA - A00394970 - SYSTEMS ENGINEERING
ANDRES CABEZAS GUERRERO - A00394772 - SYSTEMS ENGINEERING

ICESI UNIVERSITY
FACULTY OF ENGINEERING
COMPUTING AND DISCRETE STRUCTURES I

PROBLEM SPECIFICATION TABLE

CLIENT	Territory Conquest Game
USER	Player
FUNCTIONAL REQUIREMENTS	<ul style="list-style-type: none">● R1: Add territories to the game● R2: Establish connections between adjacent territories● R3: Allow players to conquer adjacent territories● R4: Calculate the score of each player based on the territories conquered● R5: Use graph traversal algorithms (BFS or DFS) to determine the areas conquered by each player
CONTEXT OF THE PROBLEM	<p>The creation of a strategy game based on the conquest of territories is required. Additionally, there must be an efficient system for managing the territory conquest process in the game. Players must conquer adjacent territories, represented as vertices in a graph, where the edges represent the connections between the territories. The objective is to determine the areas conquered by each player and calculate the corresponding score.</p>
NON-FUNCTIONAL REQUIREMENTS	<ul style="list-style-type: none">- RN1: Representation of Graphs, by means of own and implemented codes, the problem must be solved.- RN2: Intuitive user interface- RN3: The project must be uploaded to the GitHub platform and must have changes that allow the evolution of the project to be tracked.

REQUIREMENT SPECIFICATIONS

Name or identifier	R1: Add Territory		
Summary	The system should allow the addition of a territory to the game, along with its corresponding connections.		
Inputs	input name	Datatype	Selection or repetition condition
	territoryName	String	N/A
	adjacentTerritories	Array/List of Strings	N/A
General activities necessary to obtain the results	<ol style="list-style-type: none">1. Prompt the user to enter the name of the territory to be added.2. Store the entered territory name in the variable territoryName.3. Prompt the user to enter the names of the adjacent territories.4. Store the entered adjacent territories in the variable adjacentTerritories.5. Create a new territory object using the provided information.6. Add the new territory to the graph and establish the corresponding connections with the adjacent territories.		
Result or postcondition	The new territory is added to the game with its connections to the adjacent territories.		
Outputs	output name	Datatype	Selection or repetition condition
	confirmation	String	N/A

Name or identifier	R2: Establish connections between adjacent territories		
Summary	The system must allow establishing connections with weight between adjacent territories in the territory conquest game, where the weight represents the military power or strength of the connection.		
Inputs	input name	Datatype	Selection or repetition condition
	territorySource	String	N/A
	territoryDestination	String	N/A
	edgeWeigth	int	N/A
General activities necessary to obtain the results	<ol style="list-style-type: none"> 1. Check if the origin and destination territories exist in the game. 2. Verify if the territories of origin and destination are different. 3. Check if the territories are adjacent to each other. 4. Check if there is already a connection between the territories. 5. Establish the connection with weight between the territories. 		
Result or postcondition	The connection with weight between the adjacent territories is established, where the weight represents the military power or strength of the connection.		
Outputs	output name	Datatype	Selection or repetition condition
	N/A	void	N/A

Name or identifier	R3: Allow players to conquer adjacent territories		
Summary	The system must allow players to conquer Adjacent territories in the strategy game based on the conquest of territories. This implies that a player can select their own territory and attack an adjacent territory controlled by another player, with the aim of conquering it.		
Inputs	input name	Datatype	Selection or repetition condition
	attackingTerritory	String	N/A
	defendingTerritory	String	N/A
General activities necessary to obtain the results	<ol style="list-style-type: none"> 1. Verify the validity of the territory of origin selected by the player. 2. Verify the validity of the target territory selected by the player. 3. Check if the source territory and the target territory are connected by an edge in the graph. 4. Calculate the probability of success of the attack based on the military power of the territories involved. 5. Update game settings in case the attack is successful. 6. Do not make any changes to the game settings if the attack fails. 		
Result or postcondition	The player's conquered territories are updated in the game settings and a confirmation message if the attack was successful.		
Outputs	output name	Datatype	Selection or repetition condition
	confirmation	String	N/A

Name or identifier	R4: Calculate the score of each player based on the territories conquered		
Summary	In this requirement, it is sought to calculate the score of each player based on the territories conquered in the strategy game based on the conquest of territories.		
Inputs	input name	Datatype	Selection or repetition condition
	playerTerritories	ArrayList	N/A
	edgeWeights	ArrayList	N/A
General activities necessary to obtain the results	<ol style="list-style-type: none"> 1. Calculate the size of the area conquered by the player based on the conquered territories. 2. Calculate the player's score based on the size of the conquered area. 3. Consider the weight of the edges that connect the territories within the conquered area to adjust the score, if necessary. 		
Result or postcondition	Each player's score is calculated and updated in the game settings by displaying each player's score.		
Outputs	output name	Datatype	Selection or repetition condition
	playersScore	String	N/A

Name or identifier	R5: Use graph traversal algorithms (BFS or DFS) to determine the areas conquered by each player		
Summary	In this requirement, it is sought to use graph traversal algorithms, such as BFS (Breadth-First Search) or DFS (Depth-First Search), to determine the areas conquered by each player in the strategy game based on the conquest of territories.		
Inputs	input name	Datatype	Selection or repetition condition
	player	Player	N/A
	playerTerritory	Territory	N/A
General activities necessary to obtain the results	<ol style="list-style-type: none"> 1. Select a territory controlled by the player as the starting point. 2. Initialize an empty set or list to store the territories conquered by the player. 3. Take a walk over the graph using BFS and DFS, starting from the starting territory. 4. During the tour, mark each territory visited as part of the area conquered by the player and add it to the set or list of conquered territories. 5. Repeat this process for each territory controlled by the player. 6. Save the set or list of territories conquered by the player. 		
Result or postcondition	The areas conquered by each player are determined using graph traversal algorithms.		
Outputs	output name	Datatype	Selection or repetition condition
	confirmation	String	

ENGINEERING DESIGN PROCESS

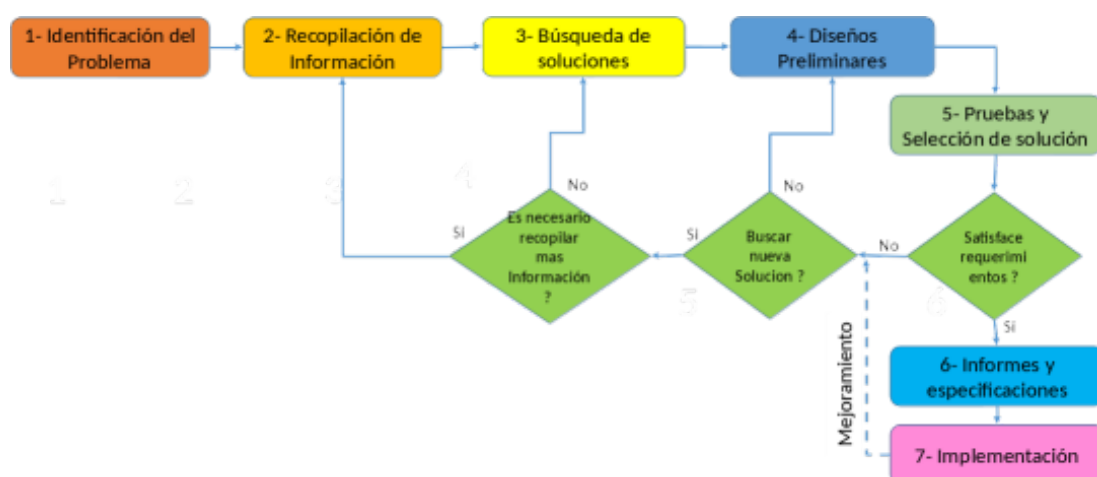
Context of the problem:

The problem in question is the creation of a strategy game based on the conquest of territories. Players must conquer adjacent territories, represented as vertices in a graph, where the edges represent the connections between the territories. The objective is to determine the areas conquered by each player and calculate the corresponding score.

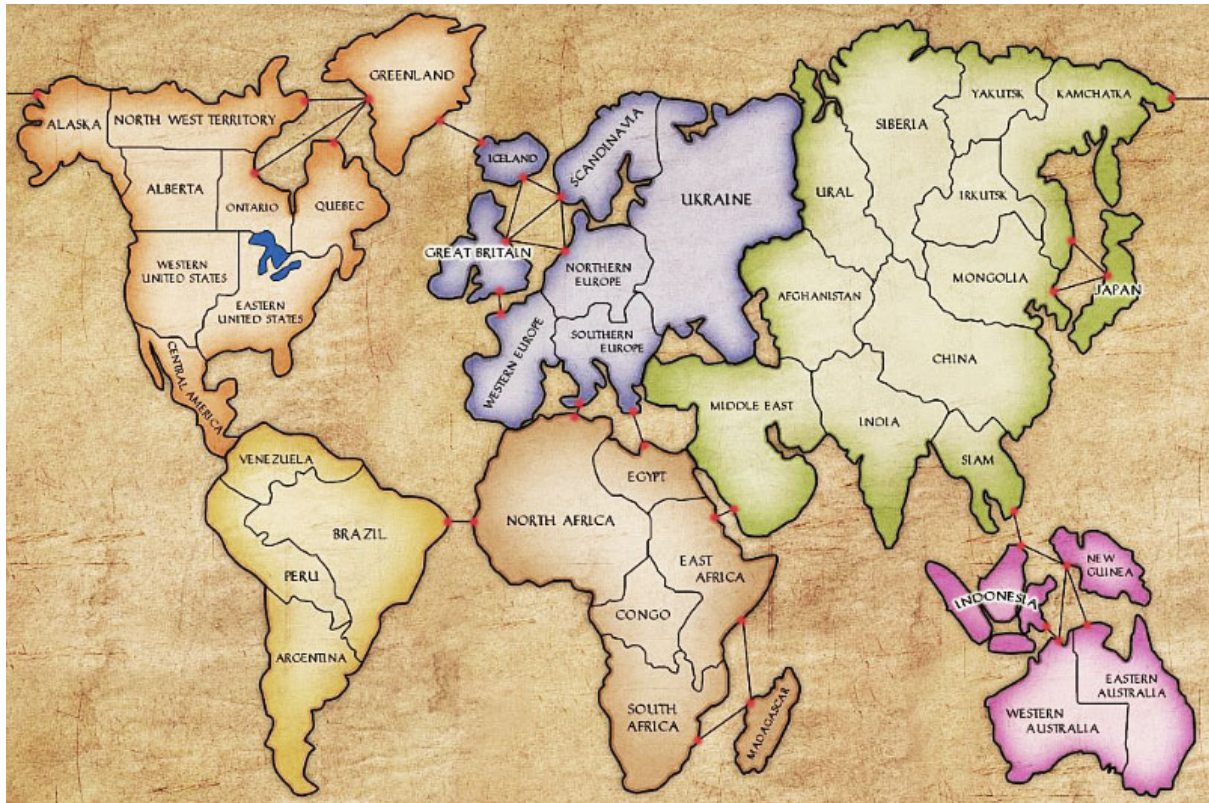
Solution development:

Given the context and nature of the problem posed, we have decided to use the Engineering Method for the development of an effective and efficient solution. This systematic approach will allow us to perform a thorough analysis and understanding of the problem situation, identify the necessary requirements, and set clear and achievable goals for the solution.

Based on the description of the Engineering Method provided in the book "Introduction to Engineering" by Paul Wright, we have defined the following flowchart, which we will follow in the development of the solution:



Step 1: Problem Identification:



Problem definition:

The problem of the strategy game based on the conquest of territories centers on the way in which the players can conquer adjacent territories in the context of the graph. In this game, the territories are represented as vertices in the graph, the edges represent the connections between the territories, and the weights of these represent the troops available from a territory in that region.

The edges of the graph are essential, since they define the adjacency relations between the territories. Each edge connects two vertices and represents that those territories are adjacent to each other, which means that they can be conquered one after the other in the game. So, the existence of an edge between two vertices indicates the possibility that a player can conquer one of the territories with the troops of that edge and then advance towards the adjacent territory through that connection.

The vertices represent the individual territories in the game. Each vertex of the count corresponds to a specific territory that players can conquer. The way the vertices are connected by the edges determines which territories are considered adjacent and thus can be conquered by the player.

The problem lies in finding an optimal strategy so that the players conquer adjacent territories in the graph, taking advantage of the connections represented by the edges. Graph traversal algorithms such as BFS and DFS are used to determine the areas conquered by each player and calculate the corresponding score. These algorithms allow players to efficiently explore the graph and make the best decisions.

Symptoms and Needs:

- Lack of an efficient system for managing the process of conquest of territories in the game.
- Absence of a data structure that allows the territories and the connections between them to be adequately represented.
- Lack of algorithms for routes over graphs to determine the areas conquered by each player.

Causes:

- Difficulty for players to strategically conquer adjacent territories.
- Lack of balance in the game, since the scores of each player cannot be calculated correctly.
- Loss of interest and satisfaction on the part of the players when facing a challenging game but with problems in its operation.
- Potential conflicts between players due to lack of clear and fair territory conquest mechanics.
- Limitation in the gaming experience, as players cannot take full advantage of the strategic and competitive features of the game.

Step 2: Information Gathering:

Once the problem has been identified and the needs properly defined, the engineer begins to gather the information and data necessary to solve it. In the case of the current project, the problem revolves around the creation of a strategy game based on the conquest of territories using graphs. The objective is to determine the areas conquered by each player and calculate the corresponding scores.

In this initial phase, comprehensive information on graph-based games, strategies for territory conquest, and graph data structures was collected. Gathering information is crucial to understanding current practices in game development, as well as identifying best practices for designing and implementing conquest mechanisms.

As a first section, various graph data structures suitable to represent the territories and their connections will be explored. Common data structures such as adjacency lists or adjacency matrices will be studied, along with their advantages and disadvantages.

Adjacency Lists: An adjacency list is a data structure that is commonly used to represent graphs. It consists of a list of vertices, where each vertex has associated a list of its adjacent vertices. This representation is efficient in terms of space when the graph is sparse and allows fast access to the adjacent vertices of each vertex.

Bibliographic reference:

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

Adjacency Matrices: An adjacency matrix is a two-dimensional data structure that represents a graph. The matrix has dimensions $n \times n$, where n is the number of vertices in the graph. Each entry in the array represents a connection between two vertices, and a Boolean or numeric value is used to indicate the existence or weight of the edge. This representation is efficient in terms of fast edge access, but may require more space compared to adjacency

lists, especially for large or dense graphs.

Bibliographic reference:

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.

Secondly, it will talk about games on the market similar to the one that you want to develop in order to have a solid foundation for the development of the solution to the problem.

An example of an existing game that bears similarities to the project under development is the popular board game "Risk." In Risk, the game board is made up of a map divided into regions, where each region represents a territory. These regions are connected to each other through shared borders, which establishes a graph structure in the game.

As in the development project, in Risk players must conquer adjacent territories to expand their domain. This is similar to the idea of the vertices of the graph representing the territories in the project. Players in Risk make strategic decisions about which territories to attack and how to deploy their armies, considering the connections between territories and the possibility of advancing into neighboring territories.

Also, in the board game Risk, the concept of conquering entire regions is related to the idea of areas conquered by each player in the project. Players must plan and execute strategies to fully control a region, which results in the conquest of all related territories within the graph.

The inclusion of this example, the board game "Risk", provides a relevant and practical analogy for understanding how adjacent territories and the connections between them can be represented by graphs in the context of the project under development. Likewise, it can serve as a source of inspiration for the implementation of the mechanics of conquest of territories and calculation of conquered areas using the graph structure.

Bibliographic reference:

Hasbro. (1959). Risk [Board game]. Hasbro Gaming.

Likewise, it will be important to investigate graph traversal algorithms to calculate the areas conquered by each player. Depth Search (DFS) and Breadth Search (BFS) are examples of algorithms that can be used to efficiently explore connected territories.

Depth Search (DFS): Depth search algorithm is a technique used to explore or traverse a graph. Start at an initial node, and from there, explore as deeply as possible along each branch before backtracking. It uses a stack to keep track of the nodes that need to be visited. DFS is useful for finding paths, detecting cycles, and performing searches in unweighted graphs.

Bibliographic reference:

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

Breadth-Finding Search (BFS): The breadth-finding algorithm is another technique used to traverse or explore a graph. It starts at an initial node and explores all of that node's neighbors before moving on to neighbors of neighbors. It uses a queue to keep track of the nodes that need to be visited. BFS is useful for finding the shortest path between two nodes, discovering connected components, and performing searches in unweighted graphs.

Bibliographic reference:

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.

In conclusion, by collecting information from various sources such as research articles, books, and online resources, you will establish a strong knowledge base. This knowledge will be used to propose adequate and efficient solutions to the problem at hand, including the representation of territories as vertices and the use of graph algorithms to calculate

conquered areas and scores.

Step 3: Creative solutions search: In this phase, the creation of creative solution ideas for the problem is intended:

It is important to generate a preliminary list of solutions to have a more accurate approximation to the problem. This phase is fundamental to solve the problem effectively, and it is also important for steps 4 and 5 of the engineering method, where less feasible ideas are discarded, and the best solution is selected. The technique used for the project was brainstorming, which is based on the free and spontaneous generation of ideas without judging them, and in the subsequent selection and combination of those considered most relevant and useful. The team members were asked to propose solution ideas based on their research and knowledge of the problem, and the resulting solution ideas are as follows:

- Random Starting Positions: Implement a mechanism where the starting positions of the territories assigned to the players are random in each game. This adds unpredictability and strategic diversity to each game, making it more challenging and entertaining.
- Special Abilities: Introduce special abilities or upgrades that players can acquire during the game. These abilities can provide unique advantages such as faster troop movement, increased defense, or the ability to conquer multiple territories in a single turn. This adds a layer of complexity and strategic decision making to the game.

- **Diplomatic Negotiations:** Allow players to negotiate alliances, treaties, and trade agreements with each other. This introduces a diplomatic aspect to the game, where players can form temporary alliances to conquer territories together or negotiate a peaceful coexistence to focus on other strategic objectives.
- **Territory Improvements:** Allow players to improve conquered territories by investing resources. These upgrades could provide defensive bonuses, increases in resource production, or additional troop recruiting capabilities. This incentivizes players to focus on specific territories and strategically manage their resources.
- **Event Cards:** Introduce a deck of event cards that can be drawn during the game. These cards can have various effects, such as changing the borders between territories, triggering natural disasters, or providing temporary buffs or setbacks for players. This adds unpredictability and strategic adaptability to the game.
- **Multiple Victory Conditions:** Instead of a single victory condition (for example, conquer all territories), introduce multiple ways to win the game. For example, players can achieve victory by conquering a certain number of regions, forming alliances with other players, or completing specific objectives. This diversifies the game and allows for different strategic approaches.

It must be taken into account that the previous list of solutions was proposed based on the

characteristics that the solution to the problem could have, because the game structure is already defined (it will not be changed in the development of the solution), therefore, in the solution proposals an attempt was made to propose a feature that would differentiate the game and give it a degree of originality, which is one of the main objectives of the project.

Step 4: Transition from idea formulation to preliminary designs:

In this phase, ideas that are not feasible will be discarded, and promising ideas will be shaped and modified to form feasible drafts and designs.

First, we will start with the ideas that will be discarded, which are as follows:

- Diplomatic Negotiations: If the main objective of the game is to focus on military strategy and the conquest of territories, diplomatic negotiations could divert attention from that central focus. If you're looking for a game more oriented towards direct competition and tactical decision-making on the battlefield, discarding diplomatic negotiations can keep the game focused on those aspects.

- Event Cards: If you want to keep the game more predictable and strategic, event cards can introduce too much uncertainty and unpredictability into the game. If you want players to plan and execute long-term strategies, event cards could generate random outcomes that make it difficult to make strategic decisions and long-term planning. Dismissing this idea will help keep the game more focused on strategic planning and execution.

- Territory Improvements: If the game is primarily focused on territory conquest and direct interaction between players, territory improvements can add an additional layer of complexity and resource management that can divert attention from the

main objective. If you're looking to keep the game more streamlined and oriented towards direct military strategy, discarding the improvements in territories can simplify the game mechanics and focus it on the conquest of territories itself.

In conclusion, when analyzing the different solution ideas proposed for the problem of the conquest of territories in the strategy game based on graphs, three of them have been discarded: diplomatic negotiations, event cards and improvements in the territories. These decisions are based on the need to maintain the strategic focus of the game, either through direct competition in the conquest of territories or long-term strategic planning. By discarding these ideas, it seeks to maintain a balance in the game, avoid possible imbalances between players and guarantee an experience focused on making tactical decisions and executing military strategies. With this selection of solutions, it will be possible to advance in the next stage of the engineering process to choose the best solution and continue the development of the game.

Next, we will discuss the ideas that are not discarded, specifying them and making their respective analytical specification models:

random starting positions:

This idea proposes to assign the starting positions of the players randomly on the game map. At the start of the game, each player would be given a random territory to begin their conquest. The advantage of this mechanic is that it introduces an element of surprise and variety to each game, requiring players to adapt their strategies based on the initial circumstances. However, a potential downside is that randomization can lead to initial imbalances, as some players might start out in more favorable territory than others. This can affect the competitiveness and equality of opportunity between the players.

Analytical model:

1. Definition of variables:

- N: Total number of players
- Map: Representation of the game map
- Players: List of players
- Territories: List of available territories
- Assignments: Random assignments of starting territories for players

2. Generation of random assignments:

- Create a list of available territories.
- For each player in the player list:
 - Randomly choose a territory from the list of available territories.
 - Assign that territory to the player.
 - Remove that territory from the list of available territories.

3. Verification of initial imbalances:

- Calculate a balance metric (for example, add the characteristics of the territories assigned to each player).
- Calculate the mean and standard deviation of the equilibrium metric for the initial assignments.
- If the standard deviation exceeds a predefined threshold, it is considered that there are initial imbalances.

4. Management of imbalances:

- If there are initial imbalances:

- Make adjustments to territory assignments to balance the game (for example, swap territories between players).

- Provide bonuses to disadvantaged players to offset the initial imbalance.

- If there are no initial imbalances, continue with the initial assignments.

5. Start of the game:

- Show the game map with the starting positions of the players.

- Start the game.

6. Development of the game:

- Players carry out their actions according to the rules of the game.

- Continue the game until an end condition is met (for example, full conquest, time up, etc.).

7. Game over:

- Determine the final result of the game.

- Show the results (points, classification, etc.).

- Finish the game.

This analytical model provides a general structure to implement the proposed idea.

Generating random assignments, checking for initial imbalances, and handling imbalances are important steps in ensuring competitiveness and equal opportunity among players. The

development and the end of the game it will depend of the specific rules and mechanics of the game in question.

Special abilities:

The idea of special abilities is to assign each player unique or special abilities that give them strategic advantages during the game. Each player would have a different ability that could be used at strategic moments to influence the development of the game. These abilities can vary, from increasing the defense of one's own territories, obtaining additional resources, sabotaging the opponent's movements, among other possibilities. Special abilities add diversity and customization to the game, allowing players to explore different strategies and tactics based on their unique abilities. However, it is important to balance these skills to avoid any of them being too powerful or unbalanced compared to the others, which could affect gameplay and competitiveness.

Analytical Model:

1. Definition of variables:

- N: Total number of players
- Players: List of players
- Skills: List of special skills available
- Assignments: Special skill assignments for players

2. Generation of special ability assignments:

- Create a list of special abilities available.
- For each player in the player list:
 - Randomly choose a special skill from the list of available skills.
 - Assign that special ability to the player.
 - Remove that special skill from the list of available skills.

3. Balance of special abilities:

- Evaluate each special ability and its impact on the game.

- Assign a level of power or advantage to each special ability.
- Verify that there is no significant disproportion in power levels between the special abilities assigned to the players.
- If an imbalance is detected, make adjustments to power levels or assigned abilities to balance the game.

4. Start of the game:

- Show the special abilities assigned to each player.
- Start the game.

5. Development of the game:

- Players use their special abilities at strategic moments during the game.
- Each player must abide by the rules and restrictions associated with their special ability.
- Special abilities can influence the development of the game, such as increasing defense, obtaining additional resources or sabotaging the opponent's movements.

6. Balance during the game:

- Monitor the impact of special abilities during the game.
- If any special ability becomes too powerful or unbalanced, consider adjustments to keep the game competitive and fun.
- Make changes to special abilities, restrictions or associated rules if necessary.

7. Game over:

- Determine the final result of the game, considering strategic actions and the use of special abilities.
- Show the results and reward the players according to their performance.
- Finish the game.

This analytical model provides a general framework for implementing the idea of special abilities. Generating random assignments, balancing skills, and in-game monitoring are important steps in ensuring a balanced and fun gameplay experience. Each special ability can provide unique strategic advantages, allowing players to explore different tactical approaches. However, it is essential to balance the special abilities to avoid any of them being too powerful and unbalanced, which could negatively affect the gameplay experience.

and the competitiveness between players.

Multiple victory conditions:

Instead of having a single win condition, this idea proposes setting different conditions that players must meet to win the game. For example, apart from the conquest of territories, conditions such as accumulating a certain amount of resources, completing specific objectives or achieving an alliance with other players could be included. The implementation of multiple victory conditions adds strategic depth to the game, as players must consider different approaches and tactics to reach their goal. However, it is important to ensure that the conditions are balanced and avoiding to help too much of a type of strategy or a particular player, to maintain competition and challenge among the participants.

Analytical Model:

1. Definition of variables:

- N: Total number of players
- Players: List of players
- Conditions: List of available victory conditions
- Objectives: Specific objectives associated with victory conditions
- Progress: Each player's progress towards victory conditions

2. Establishing Victory Conditions:

- Create a list of available victory conditions.
- For each victory condition, define the associated specific objectives.
- Assign a level of difficulty or requirements for each objective.

3. Start of the game:

- Assign victory conditions to players.
- Show the conditions and objectives assigned to each player.

- Start the game.

4. Development of the game:

- Players carry out their actions and strategies to meet the objectives and the assigned victory conditions.
- Objectives can include the conquest of territories, accumulation of resources, completing specific missions or establishing alliances with other players.

5. Tracking progress towards victory conditions:

- Record the progress of each player towards the assigned objectives.
- Update the progress based on the actions performed by the players during the game.

6. Balancing Victory Conditions:

- Evaluate the conditions and objectives assigned to each player.
- Verify that there is no significant disproportion in difficulty or requirements between the victory conditions.
- Make adjustments to the requirements or assigned conditions if an imbalance is detected to maintain competition and challenge among players.

7. Verification of victory conditions:

- Check if any player has fulfilled all the assigned victory conditions.
- If a player fulfills all his victory conditions, he is declared the winner of the game.

8. Game over:

- Show the results and reward the winning player.
- Finish the game.

This analytical model provides a general framework for implementing the idea of multiple victory conditions. The conditions and objectives assigned to each player allow for greater strategic depth, as players must consider different approaches and tactics to meet their

specific objectives. Balancing victory conditions is essential to avoid overly favoring a particular strategy type or player, thus maintaining competition and challenge among participants.

Step 5: Evaluation and Selection of the Best Solution:

Finally, in this writing, the selection of the best solution will be carried out. For this purpose, a series of criteria were established that will be scored from 1 to 5 (where 5 is the highest score considered excellent and 1 is the lowest score considered regular) and will allow choosing the best solution proposal. The criteria are as follows:

- Efficiency: Which of the proposed solutions is more efficient in terms of time and resources?
- Ease of implementation: Which of the solutions is easier to implement and maintain? Is research necessary to solve the problem?
- Scalability: Capacity of the strategy game system based on the conquest of territories to handle an increase in the number of players, territories and connections.
- Usability: Is the solution easy to use for end-users? Is it easy to understand and use for users?
- Security: Does the solution guarantee the privacy and security of the users?
- Cost: What is the cost of implementing and maintaining the solution? Is it financially viable?

The methodology used to evaluate the criteria based on the solution proposals was to have the team meet and agree on a score (1-5) for each criterion. The results were as follows:

Random starting positions:

<i>Criteria</i>	<i>Score (1-5)</i>
Efficiency	5
Ease of implementation	4
Scalability	5
Usability	5
Security	5
Cost	5
Total	29

Special abilities:

<i>Criteria</i>	<i>Score (1-5)</i>
Efficiency	3
Ease of Implementation	3
Scalability	4
Usability	5
Security	4
Cost	4
Total	23

Multiple victory conditions:

<i>Criteria</i>	<i>Score (1-5)</i>
Efficiency	3
Ease of Implementation	4
Scalability	5
Usability	5
Security	4
Cost	4
Total	24

Based on the scores obtained, the winning criteria obtained a total of 29, the highest score among the ideas evaluated. This indicates that it best meets the established criteria and is considered the most suitable option for the project in terms of efficiency, ease of implementation, scalability, usability, security and cost.

In conclusion, after evaluating different ideas based on several criteria, the idea of random starting positions has been chosen as the optimal solution for the strategy game based on the conquest of territories. This idea scored highest for efficiency, scalability, usability, security, and cost, making it the most appropriate choice for the project.

In this solution, the game will be implemented using the Java programming language and making use of graph-based data structures. The idea of random starting positions involves randomly assigning starting territories to each player at the start of the game. This adds an element of surprise and strategy, as players don't know which territories are theirs and must adapt their strategy based on their assigned locations. The use of graphs will allow representing the connections between territories, which will facilitate the calculation of conquered areas and the determination of victory conditions. Furthermore, development in Java will provide flexibility and ease of implementation.

In summary, the idea of random starting positions has been selected due to its high score on the evaluated criteria and its ability to provide an exciting and challenging experience. With the use of graphs and the implementation in Java, it is expected to create a strategy game based on the conquest of territories that is efficient, scalable and easy to play.

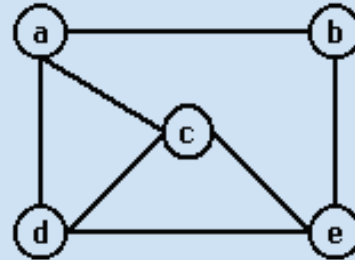
TAD DEFINITION

TAD <Graph>

$V = \{ a, b, c, d, e \}$

$E = \{ (a, b), (a, c), (a, d), (b, e), (c, d), (c, e), (d, e) \}$

Graph =



Invariant:

Let $G=(V, E)$ be a graph, where V is the set of vertices and E is the set of edges.

- For every vertex v in V , there exists a unique identifier $ID(v)$ that represents it.
- For every pair of vertices u and v in V , there exists a path from u to v through a sequence of edges in E .
- For every edge e in E , the start and end vertices of e are different, that is, $e = (u, v)$ where $u \neq v$.
- For every edge e in E , the start and end vertices of e belong to V , that is, $e = (u, v)$ where $u, v \in V$.
- For every edge e in E , there is a weight $w(e)$ associated to e , $y w(e) \geq 0$.
- For every edge $e = (u, v)$ in E , there exists an edge $e' = (v, u)$ in E , which implies an arrival connection from u to v and an arrival connection from v to u .

Main operations:

CreateGraph(): N/A \rightarrow Graph

AddVertex(): Graph x Element \rightarrow Graph

RemoveVertex(): Graph x Element \rightarrow Element

AddEdge(): Graph x ElementOrigin x ElementFinish x Weight \rightarrow Graph

RemoveEdge(): Graph x ElementOrigin x ElementFinish \rightarrow Edge

GetAdjacent(): Graph x Element \rightarrow Elements

GetWeightEdge(): Graph x ElementOrigin x ElementFinish \rightarrow Integer

GetVertexs(): Graph \rightarrow Elements

IsVertex(): Graph x Element \rightarrow boolean

IsEdge(): Graph x ElementOrigin x ElementFinish \rightarrow boolean

Specifications:

CreateGraph(): Construction Operation

“Create a new empty Graph”

{pre: True}

{pos: New Graph}}

AddVertex(): Modifying Operation

“Add a vertex to a Graph”

{pre: Graph \neq nil }

{pos: Elemento \in Graph}}

RemoveVertex(): Modifying Operation

“Deletes a vertex from a Graph”

{pre: Graph \neq nil \wedge Elemento \in Graph }

{pos: Elemento \notin Graph}}

AddEdge(): Modifying Operation

“Add an Edge between one registered vertex and another”

{pre: Graph \neq nil \wedge Element Origin \in Graph \wedge ElementoFinish \in Graph \wedge Weight ≥ 0 }

{pos: {ElementOrigin, ElementFinish} \in Graph}}

RemoveEdge(): Modifying Operation

“Deletes an Edge, but not the vertices that compose it”

{pre: Graph \neq nil \wedge Element Origin \in Graph \wedge ElementoFinish \in Graph \wedge

{ElementOrigin, ElementFinish} \in Graph}}

{pos: {ElementOrigin, ElementFinish} \notin Graph \wedge ElementOrigin \in Graph \wedge ElementFinish \in Graph}

GetAdjacent(): Parser Operation

“Get the list of Vertexs adjacent of a given Vertex”

{pre: Graph \neq nil \wedge Element \in Graph}

{pos: Elements}

GetWeightEdge(): Parser Operation

“Get the weight of a Edge registered”

{pre: Graph \neq nil \wedge Element Origin \in Graph \wedge ElementFinish \in Graph \wedge

{ElementOrigin, ElementFinish} \in Graph}

{pos: weight}

GetVertexs(): Parser Operation

“Get list of vertexs of Graph”

{pre: Graph \neq nil}

{pos: Elements}

IsVertex(): Parser Operation

“Check if the vertex exists”

{pre: Graph \neq nil}

{pos: True if Element \in Graph, False otherwise}

IsEdge(): Parser Operation

“Check if the Edge exists”

{pre: Graph \neq nil}

{pos: True yes ElementOrigin \in Graph \wedge ElementFinish \in Graph \wedge

{ElementOrigin, ElementFinish} \in Graph, False otherwise}