Engineering Method
Application to the Solution of a Problem
Game "UNO"

Problematic Context

The group of professors of the course Computation and Discrete Structures I at Icesi University has proposed a project to each of its students which consists of the development of the game "Uno", a card game in which players try to run out of cards in their hand. The idea is to create a program that includes its own data structure and includes topics worked on in class.

Solution development

It requires the creation of a documentation and subsequently the creation of a version of the game "UNO" that complies with features to distribute cards to players, implement the logic to allow players to play cards and draw new cards if necessary, validate that players' moves comply with the rules of the game and update the game state accordingly, verify if a player has run out of cards in hand to determine the winner of the game, and perform extensive testing of the game to ensure its proper functioning.

All of this will be done in a sequence of steps to provide ideas, clarify the gameplay and choose the best implementation option.

Step 1. Problem Identification

For this step the needs that must be present for the correct development of the program are recognized, it contains the identification and analysis of those needs.

Identification of needs and symptoms

- Users require a software system that allows them to play the card game "Uno" digitally.
- There is no Java implementation of the "Uno" game on the Internet or platforms that uses the required data structures to manage the game efficiently.
- The solution must ensure a fair distribution of cards, proper handling of the game rules, and a user-friendly and easy-to-read display for the user.

Problem Definition

A Java program needs to be developed that implements the card game "Uno" using stacks, queues, hash tables and priority queues to manage different aspects of the game. In addition, the solution must comply with the rules of the "Uno" game, including card distribution, player turn management, rule checking, and determining the winner of the game.

Requirements specification

CLIENT	ICESI University CED-Block
USER	Players who want to participate in the game
FUNCTIONAL REQUIREMENTS	**RF1** Start of Game **RF2** Distribute Cards **RF3** Game Turns **RF4** Special Cards **RF5** End of Game
CONTEXT OF THE PROBLEM	The development of a "version" of the “UNO” card game is expected. The challenge is to develop a version that simulates the physical game experience, implementing all the rules and variants of the game, including special cards and turn switching between players. It must be interactive, easy to use and support multiple players.
NON FUNCTIONAL REQUIREMENTS	**RNF1** Usability **RNF2** Performance **RNF3** Scalability **RNF4** Maintainability **RNF5** Portability

NAME OR IDENTIFIER	**RF1 Start of Game**		
SUMMARY	The system allows starting the “UNO” game, asking the user for the number of players (2-5) and their names.		
INPUTS	**Input name**	**Data type**	**Conditions and values**
	numPlayers	int	*Must be a number between 2 and 5 representing the minimum and maximum number of players.*
	PlayerNames	List<String>	*The list should contain the same number of names as the numbers chosen before.*
POSTCONDITIONS	*The game begins with the number of players entered, each identified by name, and all cards properly distributed to start the game.*		
OUTPUTS	**Output name**	**Data type**	**Format**
	N/A	N/A	*N/A*

NAME OR IDENTIFIER	RF2 Distribute Cards		
SUMMARY	The system distributes 7 cards to each player at the start of the game.		
INPUTS	Input name	Data type	Conditions and values
	N/A	N/A	*N/A*
POSTCONDITIONS	*Each player receives 7 cards from the deck of cards, ensuring that the game can begin with all hands correctly distributed.*		
OUTPUTS	Output name	Data type	Format
	N/A	N/A	*N/A*

NAME OR IDENTIFIER	RF3 Game Turns		
SUMMARY	The system manages the turns of play between players, allowing them to play cards from their hand or draw from the deck.		
INPUTS	Input name	Data type	Conditions and values
	cardIndex	int	*It must be between 0 and the number of cards the player currently has in his deck.*
POSTCONDITIONS	*Players can perform actions during their turns, such as playing a valid card from their hand, drawing a card from the deck if they do not have a valid card, or passing the turn if they do not wish to play any card.*		
OUTPUTS	Output name	Data type	Format
	N/A	N/A	*N/A*

NAME OR IDENTIFIER	RF4 Special Cards		
SUMMARY	The system must manage the effect of the special cards of the game UNO, such as Skip, Reverse, +2, and Color Change.		
INPUTS	**Input name**	**Data type**	**Conditions and values**
	color	Int	*It should be a number between and 1 and 4 which will correspond to the four colors available to change*
POSTCONDITIONS	*Special cards have specific effects on the game and/or the player. For example, Skip can make the next player miss his turn; Reverse can change the direction of the game; +2 can make the next player draw two additional cards; Color Change allows the player to change the color of the card in play.*		
OUTPUTS	**Output name**	**Data type**	**Format**
	message	String	*Message describing the effect of the special card played.*

NAME OR IDENTIFIER	RF5 End of Game		
SUMMARY	The system detects and handles the end of the game when a player runs out of cards in hand.		
INPUTS	**Input name**	**Data type**	**Conditions and values**
	N/A	N/A	*N/A*
POSTCONDITIONS	*The game ends when one of the players runs out of cards in his hand. The player who runs out of cards is declared the winner.*		
OUTPUTS	**Output name**	**Data type**	**Format**
	winnerPLayer	String	*Message of congratulations to the winner of the game.*

Step 2. Information Collection

In order to have total clarity about the operation of the game and the implementation of the data structures, a search for information is made and ideas about the best way to run the game are expressed. The information has been consulted in reliable sources that allow to have certainty about what is exposed below.

Cards of the game "Uno".

Each card in the game has a color, it can be red, blue, green or yellow and is accompanied by a number from 0 to 9. In addition to the numbered cards, there are special cards, such as "Color Change", "Draw 2", "Reverse" and "Skip". In total, the standard deck will contain 108 cards, which are distributed as follows:

- 80 numbered cards (0-9) in the colors red, blue, green and yellow (20 cards of each color).
- 4 "Color Change" cards (2 of each color).
- 8 "Draw 2" cards (2 of each color).
- 8 "Reverse" cards (2 of each color).
- 8 "Skip" cards (2 of each color).

Card Distribution

7 cards will be distributes to each player from a standard 108-card deck at the start of the game, the deck contains all the card types already mentioned.

Start and Player's Turn

At the start of the game, a card is placed face up in the center as the initial card. Then, each player, in his turn, must play a card that matches in color, number or symbol with the top card in the discard pile. In case a player cannot play any card, he must draw a card from the discard pile.

Objective of the Game

The object and end of the game is to run out of cards in the hand. The first player to run out of cards wins the game.

Special Cards

Special cards have particular effects, such as changing the color of the game, causing the next player to draw additional cards, reversing the order of play, or causing the next player to lose his turn.

Data Structures Used

- Stacks
- Queues
- Hash Tables
- Priority Queues

Step 3. Search for Creative Solutions

In this step, we thought of some possible options that could provide an efficient solution to the project that would be both creative and simple. The options focus on the implementation of data structures for the whole process of building and managing the game:

Alternative 1

Arrays:

- For the discard deck and play deck we use an array with 108 positions for each of the cards, to draw the cards we can remove the first or the last one and thus do this process in order. The discard deck arrangement is becoming empty while the playing deck begins to fill up.

Priority queues:

- For the player's hand a priority queue is implemented so that the player can choose the card he wants to play, when the player chooses a card to this one the priority is increased so that it is the first one and can leave the player's hand of cards.

Hash tables:

- To store information of the cards: The idea is to create a hash table with 5 memory spaces where by color the cards are stored. In this case, collisions are treated in such a way that a node following the first element of the table is created in the same memory space.

CircularList:

- To determine the order of the game, it is handled as a circular list where each element has an associated position represented in the node, so the node with the highest position would be at the top of the list and to advance in the game, reference would be made to the next node and so on. In the case of reversing the order, the pointer of the "next" of each node is changed so that it now points to the one that would actually be its previous one.

Alternative 2

Stacks:

- The piles are used in the game deck so that each of the cards being played are stacked in this deck so that only the first card is taken into account to know what logic the game should follow, and the discard deck is organized with all the cards so that it is only possible to draw from the top of the deck.

Queues:

- Implemented to simulate the distribution of cards from the discard deck pile to the player’s deck. Also implemented to draw cards in order to separate only the cards that the player must take.

Hash tables:

- To store card information: a hash table is created with a memory space approximating the nearest prime number of one third of the 108 cards, i.e. 37. This is done in order to handle collisions, when there is already a memory space occupied and you want to add another card object a linked list is created in the memory space where they matched.

Priority queues:

- To determine the order of the game: The priority queue is handled as a simple list where each node has a priority to which it is associated at the time of being added, the last player to be registered will be the one that will start the round and to make a change the priority

of all players is increased allowing a space to remain in position 1, so that this becomes the place of the player who just played in the round.

Alternative 3

Stacks:

- Piles will be used to represent different aspects of the game "Uno". First, they will be used to constitute the discard pile, which will contain the 108 cards used in the game. In addition, a pile will be used to represent the game deck, where players will place the cards, they play during the game. Finally, each player will have their own card pile, which will store the cards they have in their hand.

Queues:

- This will be used when giving or distributing cards to a player. Cards will be dealt to players from a queue, and cards to be drawn will also be added from a queue to the player's deck.

LinkedList:

- A doubly linked list will be used to represent the order of players during the game. This data structure will allow for easy insertion and deletion of players, as well as navigation back and forth in the list to determine the next player in turn.

Hash tables:

- A hash table will be implemented to store additional information about the "Uno" game cards. This table will have 5 memory spaces referring to each color and will be used to associate each card with its numerical value, its type (normal or special) and any other relevant information. The hash table will provide quick access to this information, which will facilitate the implementation of the game rules and the management of the cards during the game.

Alternative 4

Array:

- One-dimensional arrays will be used to represent the playing deck and the discard deck. These arrays will allow quick access to the cards and easy manipulation of the cards during the game.

LinkedList:

- Simple linked lists will be used to represent player hands and play order. These lists will provide flexibility in card management, allowing dynamic insertion and deletion of items.

Two-dimensional arrays:

- Two-dimensional arrays could be used to represent the cards in the game, where one dimension would represent the color and the other the number or type of card. This structure would provide a logical organization of cards according to their attributes, which would facilitate the search and manipulation of specific cards.

Step 4. Transition from Ideas to Preliminary Designs

The first thing we do in this step is to discard the ideas that are not feasible. In this sense we discard alternative 1 and alternative 4 for the following reasons:

Alternative 1

First, this alternative proposes some feasible options to some extent, such as the use of priority queues and hash tables, in which we find the first problem, since if the table is of five spaces

many collisions will be generated, thus generating a list of cards in each space by color. In addition, we found the implementation of arrays to represent the playing deck and the discard deck which will inevitably cause difficulties in the management of the cards. We conclude that adding or removing cards from the decks frequently can be problematic because the arrays are of a static nature.

Alternative 4

On the other hand, this alternative presents us with LinkedList structures, arrays and two-dimensional arrays. The simple LinkedList would represent the player cards and the game order, which would result in complications searching and manipulating the elements within the list, since it would be necessary to run it sequentially to access an element. The array would be used to represent the game deck and the discard deck which, in the same model as in alternative one, could generate difficulties in the dynamic management of cards. Finally, the two-dimensional arrays would be used to represent the game cards which would require additional management to ensure the consistency of the cards in the game and their correspondence with other data structures used.

Careful review of alternatives 2 and 3 leads us to the following:

Alternative 2

For alternative two, the use of stacks, queues, hash tables, and priority queues provides versatility in the structure, i.e., it can be easily adapted to possible modifications or extensions in game One in the future. This is because these data structures are suitable for representing the game deck, discard deck, card distribution, and game order, making it easy to implement and modify the game.

Alternative 3

As for alternative three, the use of stacks, queues, doubly linked lists and hash tables also offers great flexibility and adaptability to changes in game "Uno". The combination of these structures allows an easy representation of the game deck, discard deck, players' hands and game order, which facilitates the dynamic management of the cards and the manipulation of the information associated with each card.

Step 5. Evaluation and Selection of the Best Solution

Criteria

At this point the key criteria are defined for the selection of the best alternative, which should best fit the requirements stated above and, bearing in mind that it should be an efficient and complete solution. For the selection, each criterion has been given a numerical value containing the weight or importance of such criterion in the program solution.

- *Criterion A:* Efficient. A solution with better efficiency than the others considered is preferred.
    - [3] Provides a highly efficient implementation, maximizing the performance and speed of the game.
    - [2] Provides an efficient implementation, but there may be areas where efficiency can be improved.

- [1] Has significant efficiency issues, which may negatively affect game performance.

- *Criterion B:* Entirety. A solution that addresses all project requirements is preferred. The entirety can be:
  - [3] Solves all
  - [2] Solves half of them
  - [1] Solves one

-*Criterion C*: Integrity. Implements all structures required for the project. can be:
  - [4] Implements all
  - [3] Implements 3 structures
  - [2] Implements 2 structures
  - [1] Implements only one
  - [0] Does not implement any structure

-*Criterion D:* Facility. A solution that in a simpler way correctly solves the problem is preferred.
  - [3] The solution is simple
  - [2] It has very few complications
  - [1] It is complicated to implement

-*Criterion E*: Flexibility. A solution that allows easy adaptation to possible changes or extensions in the "Uno" set in the future is preferred.
  - [3] Provides great flexibility and adaptability to changes.
  - [2] Provides some flexibility, but may require significant modifications to adapt to changes.
  - [1] Has little flexibility and may be difficult to modify or expand in the future.

Evaluation

Evaluating the above criteria on the alternatives that remain, we obtain the following table:

	Criterion A	**Criterion B**	**Criterion C**	**Criterion D**	**Criterion E**	**Total**
Alternative 2	2	3	4	3	3	15
Alternative 3	3	3	3	2	3	14

Selection

Based on the above evaluation, Alternative 2 should be selected, since it obtained the highest score according to the defined criteria. It should be taken into account that the criterion in which the alternative was worse evaluated than the other alternative should be handled appropriately.

References

Universidad ICESI. (2024-1). Tarea integradora1_2024-1.docx.

Crespo, A. (2012, April 2). Java course: priority queuing. RedesZone. https://www.redeszone.net/2012/04/02/curso-de-java-colas-de-prioridad/.

Cuervo, V. (2009, October 6). Creating a stack in Java. Line of Code. https://lineadecodigo.com/java/crear-una-pila-en-java/

Meza, F. (2013, August 2). Using a Java Hashtable. Code Line. https://lineadecodigo.com/java/usar-una-hashtable-java/

Nogués, A. (2020, April 16). These are the 13 new ways to play cards at UNO proposed by Mattel in full quarantine. Diario Sur. https://www.diariosur.es/planes/nuevas-formas-jugar-20200416124042-nt.html

Stacks in Java (n/d). CÓDIGO Libre. Retrieved April 13, 2024, from http://codigolibre.weebly.com/blog/pilas-en-java.

Sánchez, B. C. (2021, November 8). How to play UNO: how many people, rules and who wins. Www.mundodeportivo.com/uncomo. https://www.mundodeportivo.com/uncomo/ocio/articulo/como-jugar-al-uno-cuantas-personas-reglas-y-quien-gana-51668.html.

Santoyo, S. (2020, April 17). 13 new ways to play UNO and have fun at home. Wikiduca. https://www.wikiduca.com/blog/maneras-de-jugar-a-uno

Uno, J. (2021, September 15). New Uno game. Game Uno; gonzalo. https://juegouno.online/juego-uno-nuevo/?amp=1