The engineering method

Contenido

[Context of the problem 1](#_Toc83732864)

[Identifying the problem 1](#_Toc83732865)

[Identification of needs and symptoms 1](#_Toc83732866)

[Definition of the problem 1](#_Toc83732867)

[Data collection 1](#_Toc83732868)

[Stack 1](#_Toc83732869)

[Queue 2](#_Toc83732870)

[Hash function 2](#_Toc83732871)

[Hash table 2](#_Toc83732872)

[Open Addressing 2](#_Toc83732873)

[Sorting algorithms 2](#_Toc83732874)

[Search for creative solutions 2](#_Toc83732875)

[Transition of the formulation of ideas to preliminary design ideas 2](#_Toc83732876)

[Alternative 1: 2](#_Toc83732877)

[Alternative 2 3](#_Toc83732878)

[Evaluation and selection of the best solution 3](#_Toc83732879)

[Criteria 3](#_Toc83732880)

[Evaluation 4](#_Toc83732881)

[Selection 4](#_Toc83732882)

[Preparation of reports and specifications 4](#_Toc83732883)

[Specification of the problem 4](#_Toc83732884)

[Requirements 4](#_Toc83732885)

[Nonfunctional requirements 4](#_Toc83732886)

[Considerations 5](#_Toc83732887)

[Implementation of the design 5](#_Toc83732888)

[Biography 5](#_Toc83732889)

# Context of the problem

A foreign millionaire has decided to invest in the city of Cali, he has invasion a videogame store that will work in an innovative way. Because of this, he has contacted us to develop a program that will allow “caleños” to learn more about how this new shop will work by simulation the process of buying a game in said store.

# Identifying the problem

## Identification of needs and symptoms

* The program must simulate all 4 sections of the stores buying process
  + Section 1: entrance of the client to the store
  + Section 2: search for the games
  + Section 3: Getting the physical copies of the game
  + Section 4: Buying the games
* For section 2, there must be 2 different sorting algorithms
* Throughout the whole process, the time each costumer takes must be considered
* The order in which the games are paid for must be considered
* The queue for the cash register (Section 4) must be considered and should depend in the number of available cash registers
* The program must use Queues, Stacks and Hash tables
* The output must specify:
  + Which client left the store first
  + The total price of his purchase
  + The order in which his games were packaged

## Definition of the problem

The eccentric millionaire wants the development of a program that simulates the stages a costumer will go through in his shop, so the people of Cali can get to know his new store.

# Data collection

Information that will be needed for the development of this project

## Stack

Stack is a data structure that follows the order FILO, which means, first in last out (this can also be expressed as LIFO, last in first out). This means, in this data structure, the first object added can only be accessed once all the other objects have been removed. It also means, we can only add elements to the end of the structure. As the name implies, we can picture this structure as a stack of books, where we can only pic the top book without making a mess. (<https://www.geeksforgeeks.org/stack-data-structure-introduction-program/>)

## Queue

Queue is a data structure that follows the order of FIFO, which means, first in first out. This means, only the first object can be accessed, and new objects can only be added to the end of the structure. As the name implies, we can picture this has a queue of a restaurant, where the first person in the queue is the first to get let into de establishment. (<https://www.geeksforgeeks.org/queue-data-structure/>)

## Hash function

A function that usually converts very big values and uses them as keys for a hash table. This function should be fast and should uniformly distribute keys <https://www.geeksforgeeks.org/hashing-set-1-introduction/>)

## Hash table

Stores pointers corresponding to values. These values are accessed by using the hash function to get the key, and get de value in said key position (<https://www.geeksforgeeks.org/hashing-set-1-introduction/>)

## Open Addressing

A hash table that contains all the values of the world. It does not support collisions. (<https://www.geeksforgeeks.org/hashing-set-1-introduction/>)

## Sorting algorithms

Sorting algorithms are used to order a list of elements according to the comparison of the elements. (<https://www.geeksforgeeks.org/sorting-algorithms/>)

# Search for creative solutions

* Alternative 1: The user decides enters all the information of the store, including cash registers, clients, games and stands, and the program takes care of the rest
* Alternative 2: The user acts as all the costumers and can only decide which games they buy of the set games and stands that will be delivered by the program.
* Alternative 3: The program has multiple users, an administrator that manages the games, stands, and cash registers. The other users would be the costumers, and would enter the games they want

# Transition of the formulation of ideas to preliminary design ideas

The alternative 3 is discarded as we currently don’t know how to make a program multiple users can access at the same time, nor can we manage the complications this would create.

## Alternative 1:

* Functions in a way that resembles the example that has been given
* Multiple store configurations can be created for multiple cases
* The program would not resemble how a store works, as the user enters all the information of said store
* Can cause errors if the user makes a mistake while entering all the data of the store
* Would give a fast result as all the information needed is given at once
* Would return the outputs requested

## Alternative 2

* Would resemble more how the store works
* It will not work as the example given
* There would have to be a way to re stock games
* Can’t handle multiple cases
* Not all the information is given at once
* The inputs of the user are less likely to cause error, has he doesn’t have to give as much information
* Can’t modify how many cash registers there are

# Evaluation and selection of the best solution

## Criteria

* Criteria A: The precision of the solution
  + [2] Exact
  + [1] Approximated
* Criteria B: Resemblance to the example given
  + [3] It resembles the example
  + [2] it deviates from the example
  + [1] it does not resemble the example
* Criteria C: Simulation of the store
  + [3] complete
  + [2] Misses some steps
  + [1] does not simulate the store
* Criteria D: Ease of use
  + [2] easy
  + [1] requires some explaining

## Evaluation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Criteria A | Criteria B | Criteria C | Criteria D | Total |
| Alternative 1: User enters all the information | Exact  2 | Resembles the example  3 | Complete  3 | Requires some explaining  1 | 9 |
| Alternative 2: User acts only as the clients | Exact  2 | Deviates from the example  2 | Misses some step  2 | Ease of use  2 | 8 |

## 

## Selection

Using the evaluation, we can see that the first alternative is the most well suited for the program, but not by much. Because of this, we will use the first alternative, but we will take in consideration the other alternative if need be.

# Preparation of reports and specifications

## Specification of the problem

Problem: Simulate the store

Input: Number of cases, number of cash registers, number of stands, contents of each standby the levels/floors of each stand, number of clients, the code/ID of the client and the games he would like to buy.

Output: The order in which clients paid for there games, and how much each paid

## Requirements

* The program must simulate the events that happen in the store
* The program must know which games are still available and which have run out of stock
* The program must simulate which clients are in the cash registers and which are still in the queue
* The program must calculate the order in which the clients leave the store
* The program must calculate the total price the client has paid for all his games, considering that, if a game is out of stock, in this not charge said game
* The program must consider the stacks created in the shop cart
* The program must process the n quantity of clients
* The program must process an n quantity of stands with an m quantity of levels/floors
* The program must calculate the order in which the games where packaged

## Nonfunctional requirements

* The graphic user interface must be easy to understand
* The program must be organized
* The program must use at least 2 sorting algorithms with a time complexity of O(n2)
* The program must use Stacks, queues, and hash tables

## Considerations

Things that should be considered

1. The stock of a game can run out, in which case in cannot be considerer when the client goes to pay
2. The time a client takes in the store is the order in which they get to the cash register
   1. The time depends in the other they came into de store (the first client in has a time of 1min, the second a time of 2min etc) plus the number of games they must pick off stands
3. A cash register takes 1 step by game
4. A cash register takes 1 step to get a new client after it’s finished with another costumer

The class diagram is in the folder doc, and is named TI1.png

# Implementation of the design

Main,Store,Client,Hashtable,Stand

List of tasks to implement:

1. Create new stand
2. Creates a Store
3. Create stand
4. Search empty
5. Creates a Stand
6. Hash
7. Create clients list
8. Add client
9. Order clients list
10. Order in stands
11. Add elements to queue
12. Stack games
13. Search game
14. Sort time
15. Order clients by time
16. Creates a Client
17. Cash registers
18. Check for array of null

Subroutine specification |Construction

|  |  |
| --- | --- |
| Name | createNewStand |
| Description | By reading the part of the case that involves stands, the method separates the name, amount of floors and the games to send it to another method that creates the stand. |
| Parameters | None |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | Store |
| Description | The method creates an Object “Store”. |
| Parameters | -numCashRegister: Int, the amount of cash registers. -numOfStands: Int, the amount of stands |
| Return | Object “Store” |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | createStand |
| Description | The method saves a new stand on the stands array |
| Parameters | -name: String, the name of the stand  -level: int, the amount of floors of the stand  -values: String[] the games of the stand |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | searchEmpty |
| Description | The method search for an empty spot on an array and returns the position |
| Parameters | -array: Object[], any array |
| Return | -spot: int, the empty position |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | Stand |
| Description | The method creates an Object “Stand”. |
| Parameters | -name: String, the name of the stand  -level: int, the amount of floors of the stand  -values: String[] the games of the stand |
| Return | Object “Stand” |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | hash |
| Description | The method takes each value of a game and puts the stand games in a Hashtable |
| Parameters | None |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | createClientList |
| Description | The method declarates an array of Clients |
| Parameters | None |
| Return | None |
|  |  |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | addClients |
| Description | Adds the Clients to the array of Clients |
| Parameters | clientData: String, The data of the client that is going to be added |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | orderClientLists |
| Description | The method organizes the games of the Client |
| Parameters | None |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | orderInStands |
| Description | The method takes a list of games an organizes it following the order of the Stands |
| Parameters | -games: String[], the games |
| Return | -games: String[], the games organized |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | addElementsToQueue |
| Description | The method converts an array in a Queue |
| Parameters | -games: String[], the array |
| Return | -gameQueue: Queue <Integer>, the array transformed in a queue |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | stackGames |
| Description |  |
| Parameters |  |
| Return |  |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | searchGame |
| Description | The method organizes the games of a Client, returning a new list for the Client’s games that were sold out |
| Parameters | -client: Client, the client whom the process is going to be done |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | sortTime |
| Description | This method calculates the time that the client is going to take to finish the line |
| Parameters | -client: Client, the client whom the process is going to be done  -pos: int, the position of the client |
| Return | None |

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | orderClientsByTime |
| Description | This method organizes the Clients following the time that the Client is going to take to buy the games |
| Parameters | None |
| Return | None |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | Client |
| Description | The method creates an Object “Client”. |
| Parameters | -values: String[] the games of the stand |
| Return | Object “Client”. |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | cashRegisters |
| Description | The method makes all the pay process simulation with the clients and returns all the info of the Clients shop |
| Parameters | None |
| Return | -print: String, all the info of the Clients shop(id, total pay, games) |

Texto

Descripción generada automáticamente

|  |  |
| --- | --- |
| Name | checkForArrayOfNull |
| Description | This method verifies if a Clients array is empty |
| Parameters | -test: Clients[], the array that is going to be checked |
| Return | -empty: Boolean, returns if the array is empty or not |

Texto

Descripción generada automáticamente

# Biography