Gabriel Delgado, Juan Ovalle, Juan Acevedo, Jonathan Velosa y David Velásquez

No. de Equipo Trabajo: 1

I. INTRODUCCIÓN

El siguiente documento detalla los avances que se han hecho en el proyecto del juego UNO para la primera entrega. Se describe como se va a llevar a cabo el desarrollo del proyecto, descripciones de la funcionalidad e interfaz de usuario, y un análisis comparativo para definir que estructuras de datos se acomodan mejor a las necesidades del proyecto.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

El problema que se quiere resolver es como crear un juego estilo UNO utilizando las implementaciones de diferentes estructuras de datos secuenciales. El objetivo es brindar al usuario un entendimiento de las estructuras de datos mediante una experiencia interactiva y fácil de comprender.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

El usuario del producto final será el propio jugador del UNO. Únicamente puede leer datos y actualizar datos, más no crear ni borrar. Adicionalmente, solo pueden hacer una consulta parcial de los datos ya que lo único que ven son sus cartas y la que el computador ponga en la baraja de descarte.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

El primer prototipo va a tener cuatro funcionalidades para el manejo de datos: mezclar la baraja de las 108 cartas, repartir siete cartas a cada jugador, elegir una carta del jugador y poner la carta en la cola de descarte, y vaciar la cola de descarte cuando esta se llena

Mezcla de Baraja

- *Descripción:* Genera un arreglo estático de 108 elementos para guardar las cartas y se mezclan
- *Acciones iniciadoras y comportamiento esperado*: Cuando se inicie el juego, esta función se ejecuta y el arreglo de las 108 cartas se crea y se mezcla.
- *Requerimientos funcionales*:
    - Cuando se inicie el juego, el arreglo de 108 cartas se genera
    - Se deben mezclar las cartas con un algoritmo que de forma aleatoria las reorganice

Repartir cartas

- *Descripción:* Se sacan siete cartas de la baraja principal y se le dan al jugador.
- *Acciones iniciadoras y comportamiento esperado*: Una vez se finaliza la mezcla de la baraja, inmediatamente se generan los mazos de los jugadores, el humano y la máquina.
- *Requerimientos funcionales*:
    - La lógica debe elegir al azar siete cartas de la baraja principal
    - Una vez se elige una carta, esta debe quedar deshabilitada en la baraja principal para evitar duplicados.
    - A medida que entran las cartas, se organizan de acuerdo con sus propiedades (Color, valor, poder, etc.)

Elegir y poner carta

- *Descripción:* El jugador elige que carta quiere poner en su turno
- *Acciones iniciadoras y comportamiento esperado*: Una vez se han repartido las cartas, se ejecuta el primer turno del jugador y puede poner una carta en la cola de descarte. Luego, sigue la máquina y otra vez le toca al jugador.
- *Requerimientos funcionales*:
    - Se debe considerar el tipo de carta que hay en la parte superior de la cola de descarte para determinar si el jugador puede poner su carta.
    - Si el jugador puede poner su carta, esta se debe eliminar de la baraja del jugador o la máquina.
    - Dependiendo de la carta que se pone, cambia el estado actual de la cola.
    - La carta que se elige debe quedar encolada

Vaciar cola de descarte

- *Descripción:* Una vez la cola se llena, estar debe quedar vacía para que las cartas en ella sean barajadas con la baraja principal. Solo debe quedar la última carta que se jugó.
- *Acciones iniciadoras y comportamiento esperado*: La cola alcanza su capacidad máxima, la cual es la mitad del tamaño de la baraja principal. Se espera que la cola quede

con una sola carta y que la baraja principal vuelva a ser mezclada.

- *Requerimientos funcionales*:
  - La cola debe llevar un contador de la cantidad de cartas para saber cuando se tiene que vaciar.
  - Las cartas se van eliminando de la cola deben ser habilitadas en la baraja principal.
  - Se debe mantener el estado de la cola antes del vaciado para que el juego siga fluyendo.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

Para realizar la creación de los mockups, utilizamos una herramienta para la creación de interfaces graficas denominada Figma, creando un frame por cada aspecto relevante del juego, inspirados por la estética del juego real, nuestra implementación del juego de mesa UNO busca ser minimalista y practica en cuanto al desarrollo, en cuanto a los mockups se pueden dividir en tres partes inicio del juego, in-game y final, también mostrando hacia dónde queremos apuntar nuestro diseño y dando también hincapié a la funcionalidad interna del juego, como colores principales escogimos el rojo en tono oscuro para dar una sensación a una mesa tradicional, en general, intentando dar una sensación más próxima de UNO.

https://www.figma.com/file/WWnFnxNGfEwkvP69V1sXdo/DE-1raEntrega?node-id=0-1&t=iKLMaK9wOQcXh8gD-0

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El entorno de desarrollo consiste en utilizar GitHub para el control de versiones, las librerías json y random para leer datos de prueba y elegir datos al azar. El código se escribirá en VS Code y el sistema operativo nativo será Windows.

VII. PROTOTIPO DE SOFTWARE INICIAL

https://github.com/jtndavid/DataStructures_UNO

VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para la primera funcionalidad, que es generar y barajar el manojo principal de cartas, se va a implementar un arreglo. Este arreglo sería estático ya que el juego dispone de solo 108 cartas en total. Como el manojo siempre debe estar mezclado de forma aleatoria, tanto añadir como eliminar elementos de la baraja no depende de ningún orden y con un simple PushBack y PopBack se puede lograr en tiempo constante. El PushBack se utilizaría para devolver una carta al manojo y el PopBack se usaría para retirar una carta del manojo para dársela a un jugador. Si se desea, es posible hacer una búsqueda de una carta en específico o hasta imprimir toda la baraja, pero no es necesario ya que el jugador únicamente obtiene cartas de forma aleatoria.

Las dos siguientes funcionalidades, la de repartir las cartas y elegir una carta, se van a implementar usando listas enlazadas para comer cartas, PushBacks, y eliminar cartas de la baraja del jugador, deletes. Como el tamaño de la baraja cambia todo el tiempo, implementar una lista enlazada es mucho más eficiente en término de memoria que algo como un arreglo dinámico. Actualizar elementos y consultar todos los elementos también es posible y está dentro de los permisos que tendría el jugador.

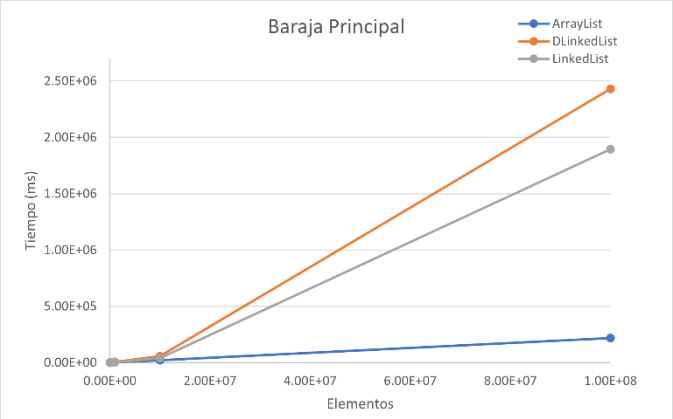
La última funcionalidad es la de vaciar la baraja de descarte y luego devolver esas cartas a la baraja principal, manteniendo una sola carta en la de descarte. Para esta baraja, se utilizó una cola, ya que a medida que se encolan cartas, al momento de vaciar la cola se pueden desencolar n-1 veces, para así mantener una sola carta al final. También, desencolar retorna la carta, que luego se puede insertar en la baraja principal con PushBacks. Al igual que la baraja inicial, se pueden eliminar elementos individuales, buscar un solo elemento, actualizar un solo elemento y hasta consultar el estado actual de la cola.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Para la funcionalidad de generar y barajar las cartas del manojo principal, se compararon tres estructuras de datos: arreglos, listas doblemente enlazadas y listas enlazadas. El proceso que se debía llevar a cabo era primero la generación de la estructura con un cierto número de cartas, o sea los elementos. Luego, se deben mezclar los elementos, para lo que se implementó dos tipos de algoritmos. Para el arreglo, se usó una versión moderna del Fisher-Yates Shuffle, en el que se intercambian los elementos al azar. Para las listas enlazadas, se usó el original, pero los nodos se pasaron a arreglos, que luego fueron mezclados y consecuentemente los índices se reasignaron a la lista original [1].

	Tiempo (ms)		
Elementos	ArrayList	DLinkedList	LinkedList
10000	45.14	94.98	66.20
100000	166.10	601.40	464.89
1000000	2434.42	5656.82	4718.70
10000000	20962.55	59242.32	39234.63
100000000	217486.10	2429922.69	1895485.61

Aquí se puede apreciar los diferentes tiempos de ejecución para cinco cantidades de elementos.



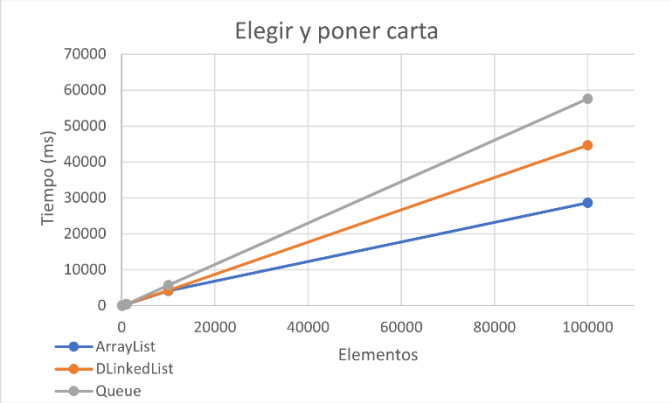
Graficando los valores obtenidos y realizando un análisis asintótico, se puede ver que la complejidad de cada estructura es de O(n), sin embargo, la estructura más rápida es el arreglo ya que solo tiene que iterar una vez la lista entera mientras que cambia elementos aleatoriamente. Por otro lado, ambas listas

enlazadas deben recorrer la longitud de esta por lo menos tres veces: una vez para copiar los nodos a un arreglo, otra vez para hacer la mezcla del arreglo y una última vez para asignar el nuevo orden de los nodos. La diferencia entre la enlazada y la doblemente enlazada es que la doblemente enlazada debe hacer más operaciones para asignar el siguiente nodo y el previo, lo que la hace la más lenta de todas.

Para la tercera funcionalidad de elegir y poner una carta en la baraja de descarte, se compararon las estructuras: arreglos, listas enlazadas y colas. La funcionalidad debe coger un elemento, borrarlo de la lista y luego encolarlo a la cola de descarte. A continuación, la tabla de tiempos:

	Tiempo (ms)		
Elementos	ArrayList	LinkedList	Queue
10	9.2	10.87	13.54
100	40.65	56.22	52.12
1000	391.73	476.31	307.95
10000	4102.35	4182.78	5692.59
100000	28574.59	44615.25	57615.25

Todos los tiempos de las estructuras son de O(n) debido a que, en el peor caso, toca recorrer la lista entera para encontrar el elemento. La eliminación también es tiempo lineal ya que en el arreglo hay que correr todos los elementos y en la lista enlazada hay encontrar el elemento previo. La cola únicamente puede eliminar el primer elemento, lo que implicaría un ordenamiento ineficiente. Esto se puede observar en la gráfica:

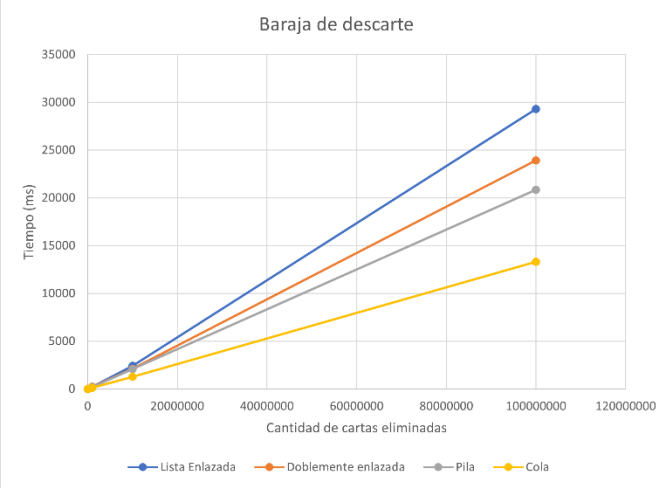


Aunque el arreglo es mucho más rápido que la lista enlazada, es posible reducir el tiempo de ejecución de la lista si se mantiene un puntero del elemento previo al nodo que se quiere eliminar, lo que nos daría una eliminación en tiempo O(1). Adicionalmente Por esto, se elegirá la lista enlazada.

La última funcionalidad es la de vaciar la cola de descarte cuando se llega a cierto número de colas, pero dejando la última carta que se encoló. Esos elementos luego se devuelven a la baraja principal mediante PushBacks seguido por un Shuffle. Se compararon cuatro estructuras: listas enlazadas, doblemente enlazadas, pilas y colas. A continuación, se ve la tabla de tiempos:

	Tiempo (ms)			
Elementos	Lista Enlazada	Doblemente enlazada	Pila	Cola
10000	2	2	3	2
100000	27	22	20	13
1000000	236	219	196	130
10000000	2435	2123	2091	1284
100000000	29288	23928	20834	13326

Todas las estructuras tienen una complejidad O(n), pero mirando los datos directamente se puede concluir que la cola es la más eficiente.



X. ROLES Y ACTIVIDADES

Integrante	**Rol**	**Descripción**
Gabriel Delgado	Líder	Realiza un seguimiento activo de todos los integrantes del grupo, propiciando un buen ambiente y desarrollo de las actividades.
Santiago Velásquez	Coordinador	Optimiza el tiempo de cada uno de los integrantes, para que encontremos un espacio en el cual reunirnos junto con la división del trabajo para cada integrante
Jonathan Velosa	Experto	Propone ideas basadas en las estructuras de datos, para obtener una mejor optimización de las implementaciones, junto con la proposición de herramientas para el desarrollo.
Juan Pablo Ovalle	Investigador	Busca reiterativamente la forma de complementar el planteamiento grupal, mediante fuentes externas, para aprender más sobre las ED y tener una buena

		perspectiva sobre cómo se trabaja este tipo de proyectos.
Juan Acevedo	Observador	Siempre está pendiente de las funcionalidades del código, y analiza las circunstancias, aparte según sea la dificultad de la división siempre está atento a ayudar

XI. DIFICULTADES Y LECCIONES APRENDIDAS

A veces se ponía confuso saber como desarrollar ciertos aspectos de la entrega, ya sea por falta de comunicación clara o simplemente no entender lo que se pedía. Ahí es donde aprendimos que es de suma importancia tener reuniones para definir bien lo que se quiere hacer y resolver cualquier duda que los compañeros tengan.

XII. REFERENCIAS BIBLIOGRÁFICAS

[1] "Shuffle a given array using Fisher–Yates shuffle Algorithm," *GeeksforGeeks*, Oct. 10, 2012. https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/