|  |
| --- |
| **Estructura de Datos y Análisis de Algoritmos** |
| **Laboratorio 2 : Poto Sucio** |
| Bastián Gonzalo Vera Palacios |

Profesor: Jacqueline Köhler

Ayudantes: Nicole Henríquez

Sebastián Vallejos

Javiera Torres

|  |  |  |
| --- | --- | --- |
|  | Santiago - Chile |  |
|  | 2-2017 |  |

Tabla de Contenidos

[CAPÍTULO 1. Introducción 4](#_Toc491820408)

[CAPÍTULO 2. Descripción de la solución 5](#_Toc491820409)

[2.1 Marco Teórico 5](#_Toc491820410)

[2.2 Herramientas y Técnicas 6](#_Toc491820411)

[2.3 Algoritmos y Estructuras de Datos 7](#_Toc491820412)

[CAPÍTULO 3. Análisis de los resultados 10](#_Toc491820413)

[CAPÍTULO 4. Conclusiones 12](#_Toc491820414)

[CAPÍTULO 5. Referncias 12](#_Toc491820415)

Índice de Figuras

[Tabla 1.1: Tabla de tiempos de ejecución 10](#_Toc491886438)

# Introducción

# Descripción de la solución

## Marco Teórico

Una vez analizada la temática de este juego, se plantea que existan jugadores los cuales vayan avanzando según turnos y a la vez cada uno de ellos debe tener su mano de cartas para poder retirar los pares e ir desechando hasta llegar al punto en que solo quede una carta en la mano de un jugador, lo que significaria la derrota del juego.

Antes de poder entregar a cada jugador su mano de cartas, se tiene que generar un mazo, el cual consiste en una baraja inglesa sin uno de los jokers para poder seguir la dinamica del juego de eliminar pares y que quede sin pareja con el fin de encontrar un perdedor. A la hora de generar el mazo, se utilizó un arreglo estatico con las cartas ya definidas en el. Junto con esto se definen dos variables las cuales usaremos para revolver el mazo y generar un nuevo arreglo de cartas con la ayuda de el modulo *rand()*.

Para poder abordar la dinamica del juego, se plantea una solución la cual consiste en la implementación de listas, ya que facilita el trabajo al ser una estructura dinamica y poder asi generar los turnos entre los jugadores y a la vez, la mano de cada uno de estos. El hecho de utilizar listas permite trabajar con cada jugador de manera independiente y con un facil acceso para la lectura/escritura de información en este. Como se quieren definir distintos tipos de datos para cada lista, se utilizaran estructuras, las cuales nos ayudaran a poder mantener un formato en cada una de nuestras listas y nuestros nodos con informacion del jugador y de las cartas.

Cuando ya se cuente con todas las funciones de listas creadas junto con sus respectivas estructuras, se puede trabajar en una función principal, la cual se encargará de llamar a todas las funciones y empezar a ensamblar todas las partes de nuestro juego.

Dentro de la funcion principal llamada *iniciar()* se deben realizar todas las consultas al usuario, respecto al nombre con el que jugará y la cantidad de rivales con los que quiera jugar.

## Herramientas y Técnicas

Entre de las herramientas utilizadas para el desarrollo de este laboratorio, se encuentran las librerías de:

* stdio.h: Para la entrada, salida de datos y manejo de archivos.
* stdlib.h: Utilizada para el manejo de la memoria dinámica.
* time.h: Utilizada para generar un número aleatorio con la función *srand( ).*
* ctype.h: Usada en la función *tolower( )* para ajustar el carácter recibido por el usuario.

Dentro de Técnicas utilizadas para la resolución de este juego

## Algoritmos y Estructuras de Datos

Para ordenar de manera óptima la información se utilizan diferentes estructuras de datos, una estructura sirve para agrupar distintos tipos de datos con el fin de ordenar y trabajar de manera más clara la información.

Las estructuras utilizadas fueron las siguientes:

**Turnos:** Estructura utilizada para generar una lista circular con los Nodos Jugadores para darle dinamica al juego.

**Jugador:** Nodo el cual contiene la información de cada jugador, como su nombre, su mano, un verificador y un puntero al siguiente jugador.

**Mano:** Estructura utilizada para generar una lista enlazada con los Nodos carta para poder generar la mano de cada jugador.

**Carta:** Nodo el cual contenia el valor de cada carta y el puntero hacia la carta siguiente.

Para utilizar las distintas funcionalidades que posee una lista, se implementan modelos de operaciones definidas, llamados TDA(Tipo de Dato Abstracto), los cuales se aplican durante la implementación del código.

El TDA desarrollado es el siguiente:

**TDA Mano de Cartas**

***Mano \* crearMano()***

Función que inicializa la estructura Mano y define en ella sus valores iniciales.

Entradas: sin entrada

Salidas: lista inicializada

***void agregarCarta(Mano \* list,int valor)***

Función encargada de insertar un Nodo Carta dentro de la lista Mano, es decir, las cartas del la mano de cada jugador.

Entradas: lista Mano, int valor

Salidas: void

***Mano \* borrarDeMano(Mano \* list,int valor)***

Función encargada de eliminar un nodo de la lista Mano según su valor.

Entradas: lista Mano, int valor

Salidas: lista Mano

***void EliminarPares(Mano \* list)***

Función que recorre la lista Mano hasta encontrar dos elementos iguales y posterior a esto, eliminar ambos elementos.

Para esto, se pasaron todos los elementos de la lista dentro de un arreglo y a la vez, se eliminaron de la lista, con el fin de facilitar la eliminación de los elementos. Dentro del arreglo se realiza la doble busqueda de elementos repetidos y una vez localizados, dentro de un arreglo auxiliar se definen como valores Nulos. Posterior a esto se revisan los elementos dentro de nuestro arreglo auxiliar, para asi ir agregando a la lista los elementos que sean distintos al valor nulo. Finalmente nos encontramos con la lista Mano con los valores duplicados ya removidos.

Entradas: lista Mano

Salidas: void

**TDA Turnos de cada Jugador**

***Turnos \* crearTurnos( )***

Función encargada de inicializar la lista Turnos y definir sus valores iniciales.

Entradas: Sin entrada

Salidas: lista Turnos

***void agregarJugador(Turnos \* list,char\* nombre)***

Función que inserta un Nodo Jugador dentro de la lista Turnos.

Entradas: lista Turnos, char\* nombre

Salidas: void

**TDA Mazo**

***void revolverMazo( )***

Función que cambia las posiciones de los valores en el arreglo mazo de manera aleatoria.

Entradas: Sin entradas

Salidas: void

**Funcion Principal**

***Iniciar( )***

Función principal en nuestro codigo la cual se encarga de juntar todo el TDA de Listas creado anteriormente y poder utilizarlo para el desarrollo del juego.

Dentro de las etapas que tenemos en la Función Iniciar( ) se encuentran:

* Solicitud de al usuario de la información que se utilizará.
* Inicialización de la lista Turnos con los jugadores correspondientes.
* Inicializacion de la Mano de cada jugador junto con la repartición de cartas inicial.
* Etapa 1 : Eliminación de pares en la Mano.
* Etapa 2 : Robo de cartas al Mazo por turnos, con su respectiva eliminación de pares.
* Etapa 3 : Robo de cartas al jugador siguiente, eliminación de pares y Conclusion del juego.

Entradas: Sin Entrada

Salidas: void

Anexo a estas funciones nos encontramos con otras funciones las cuales cumplen un rol puntual dentro del código, tales como *buscarCarta, mostrarMano, obtenerPrimerValor,* entre otras.

# Análisis de los resultados

Estos son los T(n) calculados del codigo:

Tabla 1.1: Tabla de tiempos de ejecución

|  |  |
| --- | --- |
| Funciones | T(n) |
| crearMano ( ) |  |
| agregarCarta ( ) |  |
| mostrarMano ( ) |  |
| buscarCarta ( ) |  |
| borrarDeMano( ) |  |
| obtenerPrimerValor( ) |  |
| obtenerUltimoValor( ) |  |
| obtenerAleatorio( ) |  |
| eliminarPares( ) |  |
| revolverMano( ) |  |
| crearTurnos( ) |  |
| agregarJugador ( ) |  |
| mostrarJugador( ) |  |
| revolverMazo( ) |  |
| mostrarMazo( ) |  |
| Iniciar( ) |  |

El orden de complejidad de este codigo es O(n) =

Dentro del codigo podemos apreciar que se mantuvo en general un nivel alto en tiempos de ejecución, debido a que siempre nos encontramos trabajando con matrices, por lo que en la mayoria de nuestras funciones nos encontramos con ciclos anidados logrando un tiempo promedio de en la mayoria de las funciones. La función expandirMatriz ( ) es la de mayor costo, debido a que es una función recursiva, lo que implica que cada vez que la función realiza una acción (8 en este caso particular) esta vuelve a llamarse y cada vez con una recursión menos por ejecutar. Dentro de el codigo nos encontramos con un caso particular el cual se concidera como falencia, que consiste en que la funcion expandirMatriz ( ) al empezar a expandirse y lograr avanzar en las posiciones sin valor, esta no entrega los valores numericos alrededor de las bombas, si no que solo retorna las casillas las cuales no poseian valor alguno. Esto es debido a que en el ciclo de la recursión, cuando la función debe entregar ese valor, no entra en esa sentencia y por ende, no retorna el valor. Por temas de tiempo la función no se termino del todo, pero se encuentra funcional para el desarrollo del juego ya que se puede asumir que toda la orilla del area son valores a los cuales se puede acceder sin problema de que sean bombas.



# Conclusiones

Durante el desarrollo de este laboratorio se pudo empezar a interactuar de frente con la programación en el lenguaje C de tal modo de poder familiarizarse con los conceptos de tiempos de ejecución de nuestros codigos y a la vez con el concepto de optimización, ya que de a poco se logra entender de mejor manera como funciona nuestro codigo y el costo de cada una de las funciones que implementamos. Este proyecto fue bastante util para poder entender como se utiliza el lenguaje de programación C, ya que se utilizaron la mayoria de las funciones basicas para poder implementar las funciones en nuestro codigo, tales como las lecturas de datos por usuario, trabajos con archivos para escribir información, manejo de memoria estatica y dinamica, uso de librerias, etc. Como resultado de lo realizado, se logro completar el enunciado en su mayoria, logrando una implementación idonea de las funciones y de todas las funciones solicitadas.

# Referncias

*Buscaminas. (2017, 28) de agosto. Wikipedia, La enciclopedia libre. Fecha de consulta: 11:43, agosto 30, 2017 from*[*https://es.wikipedia.org/wiki/Buscaminas*](https://es.wikipedia.org/wiki/Buscaminas)