



UNIVERSIDAD DE
COSTA RICA

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Tequila Rummy

Autores:

César Andrés Fonseca
villegas C12959 y Margaret
Orozco Arroyo B55172
(grupo 18)

Profesor:

Juan Carlos Coto Ulate

Curso:

IE0117 Programación Bajo Plataformas Abiertas

8 de diciembre de 2022

Índice

1. Introducción	2
2. Diseño general	3
2.1. Nueva Partida	3
2.2. Diez mejores jugadores	3
2.3. Reglas	6
3. Principales retos	8
3.1. Sqlite vs ficheros simples	8
3.2. Metodo de Ordenamiento	9
3.3. Paso de Variables por Referencia y por Valor	9
3.4. Matrices	11
3.5. Creación de mazo	12
4. Conclusiones	13

1. Introducción

El presente proyecto es una recreación en lenguaje C del juego Gin Rummy, el juego está hecho para ser jugado entre 2 o más personas, al inicio del juego se reparten 10 cartas a cada jugador, con estas cartas el jugador debe tratar de alcanzar el objetivo, que es formar triadas de cartas o escaleras. las triadas son combinaciones de cartas del mismo numero que cambia su simbolo, un ejemplo de este caso sería: 7 de espadas + 7 de diamantes + 7 de corazones. Las escaleras son el caso contrario de las triadas, estas se basan en un mismo simbolo, pero de diferente enumeracion ascendente, como lo sería: 4 de diamantes + 5 de diamantes + 6 de diamantes, el jugador deberá seguir este objetivo hasta que se le acaben las cartas. Lo que hace interesante al juego es que el jugador tiene que ir tomando cartas de el mazo, siempre va a haber una carta volteada y otra oculta, y el jugador deberá elegir entre cual de las dos escoger, una vez escogida una de las dos cartas, se verá en la obligación de soltar una carta de su baraja, sin embargo esta carta no puede ser la misma que la que acaba de tomar del mazo, el jugador deberá repetir este procedimiento hasta conseguir un resultado favorable y lograr formar triadas o escaleras con todas sus cartas. Hay dos conceptos importantes que el jugador debe conocer a la hora de practicar este juego, estos conceptos son los de **Knock y gin**, El **Gin** es la frase que dice el jugador a la hora de que haya combinado todas las 10 cartas de su baraja, lo que le da la victoria absoluta en el juego, y el **Knock** es un metodo alternativo por el cual se puede conseguir la victoria, metodo solo se puede utilizar cuando las cartas disponibles en la baraja del jugador no superan los 10 puntos, en este caso el jugador puede gritar KNOCK, el juego se detiene y se comparan las cartas de ambos jugadores y ahí se debe decidir cual de los dos jugadores gana. Una vez explicado en que consiste el juego, el presente proyecto se encarga de recrearlo en el lenguaje C, en el programa habran 3 funciones principales, las cuales consisten en imprimir las reglas del juego, imprimir los 10 mejores jugadores en la historia del juego y por ultimo, jugar una nueva partida, por lo que la sección de diseño general se dividirá en estas 3 principales subsecciones en las cuales se explicará el funcionamiento de cada una.

2. Diseño general

2.1. Nueva Partida

Este proyecto se comienza con la creación de un arreglo, tipo matriz, de la cual consiste en 13 filas y 4 columnas; las 13 filas consisten en los 13 números de las cartas de un mazo (tomando A como 1 y J,Q y K como 11, 12 y 13 respectivamente), las 4 columnas son los símbolos de las cartas del mazo(1,2,3 y 4 como Espadas, Diamantes, Corazones y Tréboles). Seguidamente, se desarrolla un ciclo for para poder realizar un repaso de todos los números del arreglo, así también multiplicando los dos valores del mismo, para poder obtener un mazo completo de 52 cartas. Luego, se utilizó la función de random de c, donde se necesita incorporar la librería de time.h; esto para poder realizar aleatoriedad al mazo existente de cartas.

2.2. Diez mejores jugadores

La sección de los Diez mejores jugadores es una parte que destaca por su necesidad de tener un conocimiento bien fundamentado en archivos y ficheros, de hecho, las primeras líneas de código ya avisan de esto de la siguiente manera:

```
1      void puntajes(char ganador[], int intentos) {
2      FILE *puntajes = fopen("puntajes.txt", "a+");
3
4      if(!puntajes){
5          printf("No he podido abrir el archivo");
6          return 1;
7      }
8
9      fprintf(puntajes, "%s %d\n", ganador, intentos);
10
11     fclose(puntajes);
12 }
```

La presente función es la void puntajes, esta es la función más básica, y es la encargada

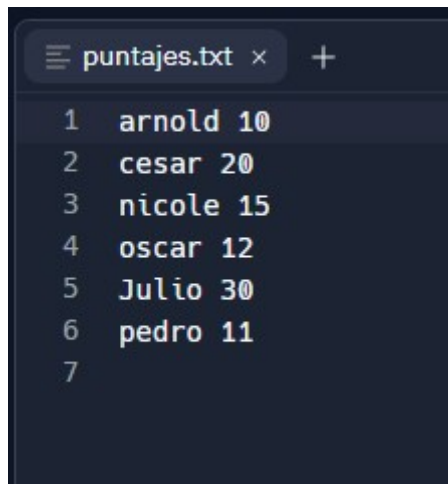


Figura 1: puntajes.txt

de agregar los nombres y la cantidad de intentos del ganador en el archivo puntajes.txt, este archivo la única función que tiene es la de usarse como base de datos para almacenar toda la información de cada ganador históricamente, se muestra de la siguiente manera en la Figura 1:

Una vez almacenados los datos del usuario primero se establecen unas variables con las que se van a poder manipular estos datos, estos serán manipulados mediante la función ordenamiento

```
1  char name[50][1024];  
2  int intentos[20];
```

Una vez se establecen estas variables ya se puede pasar a la función de ordenamiento

```
1  void ordenamiento()  
2  {  
3      FILE *fp = fopen("puntajes.txt", "r"); //abre puntajes.txt  
4      int i=0, size, j, swap;  
5      char ch;  
6  
7      if(fp==NULL)  
8      {  
9          printf("\n No he podido abrir el archivo \n");  
10         exit(0);  
}
```

```

11     }
12     while(ch!=EOF) //recorre puntajes.txt buscando las
        variables
13     {
14         fscanf(fp,"%s %d",&name[i][i],&intentos[i]);
15         ch=fgetc(fp);
16         i++;
17     }

```

Se presenta la primera parte de la función de ordenamiento, esta parte inicialmente se encarga de abrir el archivo de puntajes.txt para tomar los datos iniciales y manipularlos, despues se aplica un bucle while el cual se encarga de recorrer todo el archivo hasta el final tomando cada uno de los datos uno por uno, y almacenandolos en la memoria para luego ser manipulados con los algoritmos de ordenamiento

```

1 size=i-1; //realiza el ordenamiento de burbuja a los nombres e
    intentos de menor a mayor
2     for(i=1;i<size;++i){
3         for(j=0;j<size-i;j++){
4             if(intentos[j+1]<intentos[j]){
5
6                 swap      = intentos[j];
7                 intentos[j]  = intentos[j+1];
8                 intentos[j+1] = swap;
9
10                swap      = name[i][i];
11                name[i][i]  = name[i+1][i+1];
12                name[i+1][i+1] = swap;
13            }
14        }
15    }

```

Luego le sigue la segunda parte de esta función, la cual se encarga de realizar el ordenamiento de burbuja a las matrices para que acomode los numeros y los nombres de la

manera correcta, cabe recalcar que este algoritmo manipula todos los datos del archivo, da igual si son más de 10, este algoritmo los acomodará todos

```
1      fp=fopen("sortedRecord.txt","w"); //abre sortedRecord y
      agrega las copias optenidas despues del ordenamiento
2      for(i=0;i<10;i++){
3          fprintf(fp,"%s %d \n",&name[i][i],intentos[i]);
4          printf ("%s %d \n", &name[i][i],intentos[i]);
5      }
6      fclose(fp);
7  }
```

Mientras tanto, ya la ultima parte del archivo, se encargará de tomar los 10 primeros nombres e intentos que encuentre del nuevo ordenamiento almacenado creando un ciclo for que solo recorre hasta 10, en caso de haber menos de 10 jugadores, este los remplazará con un 0, todo esto lo va realizando mientras se ejecuta el ordenamiento de burbuja. así finalizando esta sección del código

2.3. Reglas

Como sección complementaria se decidió agregar un apartado de reglas para el usuario, este apartado se agregó debido a que durante el proceso de investigación del juego, se encontraron en la web multiples recreaciones del juego hechos en distintos lenguajes de programación y con funcionalidades distintas, sin embargo, ninguna de estas versiones tenía un apartado dedicado a explicarle al usuario el funcionamiento del juego y sus principios, lo cual para un jugador principiante puede generar una fuerte discusión a la hora de comenzar el juego, y creando una curva de aprendizaje bastante empinada en la que el usuario deberá hacer una investigación aparte para poder conocer los principios basicos del juego, a sabiendas que existen multiples formas de jugar el Gin Rummy, por lo que la información investigada por el usuario puede llegar a ser erronea. Teniendo esta justificación, se realizó un código el cual pudiera leer un archivo de formato TXT el cual contiene toda la información de las reglas, el código comienza abriendo el archivo y buscando en el proyecto, para que en caso de que este no exista, devuelva al usuario

```

1      #include <stdio.h>
2
3      void reglas() {
4          FILE *reglas = fopen("reglas.txt", "r");
5
6          if(!reglas){
7              printf("No he podido abrir el archivo");
8              return 1;
9          }

```

Una vez se haya encontrado el archivo el cual contiene las reglas del juego, el cual es llamado "reglas.txt", se procede a crear un buffer el cual tiene un tamaño de 2433 bits, ya que cada caracter de tipo char, requiere tan solo un bit de espacio de memoria, al contar cada caracter del archivo de "reglas.txt", se encuentra que existen 2433 caracteres exactos, por lo que utilizar un buffer con este numero exacto permite una mejor optimización del uso de memoria del sistema. Una vez hecho esto, se crea un ciclo do while el cual se encarga de recibir cada caracter linea por linea del archivo de "reglas.txt", y los almacena en el buffer para luego imprimirlo con una función printf, y esta función no termina hasta que el bucle encuentre un /0 al final del archivo lo cual indicará el fin de este. Una vez impresos todos los caracteres del archivo "reglas.txt", se procede a utilizar una función fclose para cerrar el archivo y continuar con la ejecución del programa

```

1      char buffer[2433];
2      do{
3          if(fgets(buffer, 2433, reglas)){
4              printf("%s", buffer);
5          }
6      }while(!feof(reglas));
7
8      fclose(reglas);
9  }

```


3. Principales retos

3.1. Sqlite vs ficheros simples

Durante el proceso de realización de la sección de las mejores puntuaciones, surgió una incognita, la cual fue, es mejor usar sqlite o ficheros simples?, esta pregunta surgió debido a que al inicio se quería realizar el proyecto en el cual se contara la cantidad de victorias del usuario, esto debido a que Gin Rummy no es un juego que se necesite los puntajes, los puntajes de Gin Rummy solo se utilizan en casos específicos y es completamente aleatorio, esto dependerá del valor de tus cartas al final de la partida, por lo tanto, utilizar puntajes para realizar la clasificación no sería el método ideal ya que realmente no reflejaría el nivel ni la suerte que ha tenido el jugador durante la partida, por lo que se tomó la decisión de realizar el juego utilizando la cantidad de intentos que necesitó el ganador para vencer al otro perdedor, esto debido a que Gin Rummy se divide por rondas, en el cual puede armar las triadas, las escaleras, o tomar una carta, por lo tanto, cada ronda que haga el ganador, equivaldría a un intento, así que las puntuaciones se distribuirían entre los jugadores que menos intentos hagan para ganar la partida. Al inicio cuando se quería hacer el trabajo con la cantidad de victorias se consideró utilizar Sqlite para realizar una base de datos la cual pueda recibir el nombre del jugador y aumentarle con un contador de victorias la puntuación al jugador, el problema que surgió con este método, y la razón por la cual se decidió omitir esta opción, fue debido a la complejidad de las bases de datos de Sqlite, se podría considerar como aprender una tecnología completamente nueva, además que se necesitan conocimientos básicos en bases de datos, debido a las limitaciones de tiempo y las limitaciones debido a la dificultad del lenguaje C, el tiempo no iba a ser suficiente para aprender esta tecnología y entenderla a la perfección, por esta razón se decidió omitirla. De esta forma se decidió realizar la función utilizando ficheros comunes, y tomar estos ficheros como la base de datos, además creando una función que realice el ordenamiento en otro fichero, así simplificando el problema y las líneas de código en gran medida.

3.2. Metodo de Ordenamiento

Para realizar las puntuaciones, la utilización de un metodo de ordenamiento es fundamental, esto debido a que la unica forma de hacer que los datos resultantes se vean de la forma solicitada en el problema es utilizando metodos de ordenamiento, cosa la cual generó problemas al inicio debido al desconocimiento de esta información, sin embargo gracias a la investigación se llegó al conocimiento de que existen multiples metodos de ordenamiento, unos más eficientes en ciertos contextos que otros, en informatica hay varios tipos de metodos que realizan esta función, los principales son Ordenamiento Burbuja (Bubblesort), Ordenamiento por Selección, Ordenamiento por Inserción, y Ordenamiento Rápido (Quicksort), todos estos algoritmos cumplen con una misma función, que es la de ordenar los datos de distintas maneras según lo solicitados, ademas que estos se pueden utilizar tanto como en matrices como en caracteres simples. Durante el proceso de realización del codigo se tomaron todas esas opciones a consideración, sin embargo se decantó finalmente por el Ordenamiento Burbuja (Bubblesort), las razones de esta desición fué debido a la simplicidad del algoritmo, debido a que era el que abarcaba menos lineas de codigo y era de forma más sencilla seguir su logica, otra razón por la que se escogió es debido a que tiene una complejidad de la forma $O(n^2)$, lo cual para la finalidad la cual se va a utilizar, realmente es una complejidad aceptable que va a permitir un trabajo fluido y sin complicaciones, otra opción fué el metodo de Ordenamiento Quicksort, pero debido a que poseia una complejidad mayor a la hora de convertirlo en codigo, por esta razón se escogió su otra alternativa.

3.3. Paso de Variables por Referencia y por Valor

Durante el proceso de desarrollo, se presentó el error de que la información entre los archivos puntajes.txt y sortedRecord.txt realmente no se estaban pasando bien, lo que provocaba que los nombres no concordaran de la manera correcta entre archivos, cometiendo el error también presentado en la Figura 1. Una de las soluciones que se le encontró a ese problema fue el descubrimiento de la teoría de Paso de Variables por Referencia y por Valor, los cuales son diferentes metodos de copiar la información que tiene el sistema, y que en este caso solo funcionaba uno en especifico, el que se estaba usando al inicio de forma inconsiente es el paso por valor, el cual funciona haciendo una

copia de cada uno de los valores recibidos, de una forma normal, sin utilizar punteros, un ejemplo de esto es que yo cree una función que intercambie dos parametros A y B de la siguiente manera:

```
1      #include <stdio.h>
2
3      void intercambiar(int, int);
4
5      int main(void)
6      {
7          int a = 2, b = 3;
8          printf("Valores originales a = %i y b = %i/n", a, b);
9          intercambiar(a, b);
10         printf("Luego de la funcion a = %i y b = %i", a, b);
11         return 0;
12     }
13
14     void intercambiar(int i, int j)
15     {
16         int z;
17         z = i;
18         i = j;
19         j = z;
20     }
```

Este metodo tomará los valores originales de a y b y enviará una copia de Valor a la función intercambiar para mostrar el resultado luego, siempre tomando como base los valores originales de las variables. El metodo por referencia es muy similar al de valor, con la diferencia es que ahora se manda un puntero el cual indica la dirección de memoria de la variable original, así copiando el valor de este desde su dirección, haciendo que si el valor de el parametro cambia, el valor de referencia también vaya cambiando.

```
1      #include <stdio.h>
2
```

```

3      void intercambiar(int, int);
4
5      int main(void)
6      {
7          int a = 2,b = 3;
8          printf("Valores originales  a = %i y b =  %i/n",a,b);
9          intercambiar(a,b);
10         printf("Luego de la funcion a = %i y b = %i",a,b);
11         return 0;
12     }
13
14     void intercambiar(int &i, int &j)
15     {
16         int z;
17         z = i;
18         i = j;
19         j = z;
20     }

```

Como se ve en el código mostrado, lo que cambia es que la función recibe la referencia de los parámetros utilizando el símbolo

```

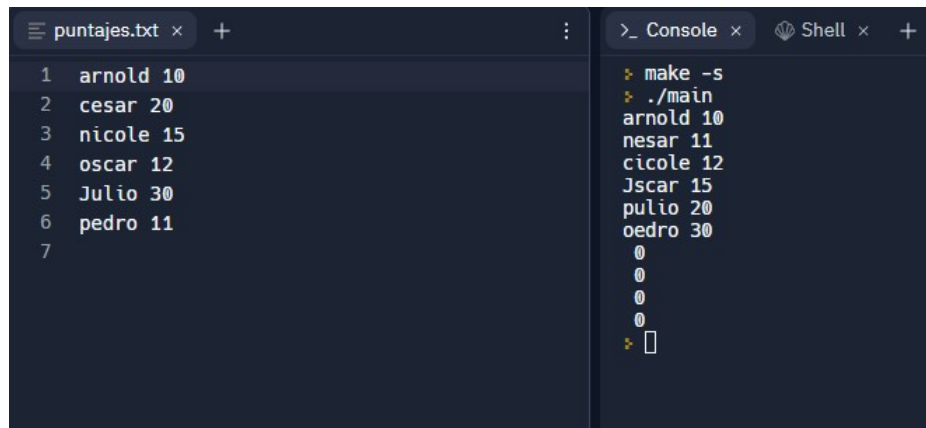
1      &

```

como indicador de esto, imprimiendo el mismo resultado pero con una copia de forma distinta. Este método de referencia se aplicó en el código para enviar los datos entre txt, dando un mejor resultado y manteniendo la información de manera más precisa, sin embargo, no logró solucionar todo el problema, solo lo hizo menos grande

3.4. Matrices

Debido al desconocimiento y la poca mención en horarios de clase sobre este tema, hubo un gran problema a la hora de realizar las puntuaciones, esto debido a que al final, cuando se tenía el código de ordenamiento listo, solamente detectaba la primera letra



```
puntajes.txt x +
1 arnold 10
2 cesar 20
3 nicole 15
4 oscar 12
5 Julio 30
6 pedro 11
7

>_ Console x Shell x +
make -s
./main
arnold 10
nesar 11
cicole 12
Jscar 15
pulio 20
oedro 30
0
0
0
0
0
0
```

Figura 2: Error de impresión debido a la falta de matrices

de cada uno de los nombres, esto generando un error muy peculiar el cual provocaba que a la hora de imprimir el resultado, la cantidad de intentos se ordenara de la forma correcta, pero la unica letra que les seguía era la primera, por lo que los nombres quedaban desordenados y en algunos casos formando palabras realmente extrañas como se muestra en la Figura 2 (la cual por extraña razón no se acomoda en el lugar correcto), donde la barra izquierda es el input y el de la derecha es el output:

La solución que se le encontró a este problema fué crear una matriz, sin embargo no se logró hacer que el problema se solucionara, debido a un funcionamiento diferente de los ordenamientos de burbuja, al final, por falta de tiempo, no se logró solucionar el problema, pero lo que se sabía es que si se lograban acomodar esas matrices y ordenamientos de la forma correcta, el codigo hubiera funcionado correctamente

3.5. Creación de mazo

a la hora de realizar el mazo principal de cartas, se probó utilizando structs de distintas formas, al final se logró llegar a una solución superficial la cual se basaba en utilizar string como forma de represenatar los numeros y los simbolos, ya que esto permitía un mejor manejo de las cartas, luego se pusieron ciclos for los cuales crearan un nuevo maso el cual incluyera cada una de las cartas del maso de 52, ya con el mazo armado, se aplicó la función de randomizar para crear siempre en cada partida un maso general diferente, y así proponer un juego más dinamico y seguro donde lo que más importe sea la suerte del jugador de recibir un maso el cual le sea beneficioso

4. Conclusiones

Debido a la complejidad del proyecto, realmente la necesidad de investigación y aprendizaje continuo fueron elementos cruciales del proceso, principalmente en la sección relacionada a los puntajes de los jugadores, lo cual requirió una investigación completa de múltiples tópicos de la informática y C los cuales como desarrollador, se tenía completo desconocimiento al inicio del proceso, un ejemplo de estos tópicos descubiertos fue la del desarrollo de los algoritmos de ordenamiento, tema el cual puede llegar a resultar bastante interesante por la complejidad numérica que reservan dentro de su lógica, la pregunta sobre este tema que quedó sin responder, es la de como hubieran funcionado distintos tipos de estos algoritmos en el código desarrollado, debido a que dependiendo del contexto, diferentes tipos de estos algoritmos son más eficientes o deficientes, el tener la posibilidad de medir el tiempo de ordenamiento de cada algoritmo hubiera sido una cualidad la cual durante el desarrollo y para el aprendizaje hubieran sido un complemento el cual hubiera enriquecido el aprendizaje. La experiencia se puede decir que fue un proceso difícil, nunca se había realizado un proyecto de este tipo, el simple hecho de que la mayor parte del conocimiento que se necesitaba para realizar el trabajo no se había visto en clases, complicó más las cosas, siendo a veces una experiencia estresante y frustrante como desarrolladores, la complejidad tan respetable que tiene un lenguaje de programación como C hizo que el proyecto se tornara de una dificultad agregada, sin embargo, se obtuvo un aprendizaje más profundo sobre el manejo de memoria en los sistemas informáticos, siento también un gran aporte para el desarrollo personal como desarrolladores e ingenieros. El aprender un lenguaje de estas características, permite un mejor desarrollo lógico como ingenieros, permitiéndonos ver el software desde un punto de vista más lógico y profundo, permitiendo que al momento de aprender otros lenguajes u otras tecnologías, sea más fácil dominarlas, y así además aprender de forma más amena sobre teoría computacional, la cual también fue necesaria investigarla durante el proceso, y que el conocimiento de este lenguaje C, permitió una profundización en estos tópicos esenciales de la informática moderna.