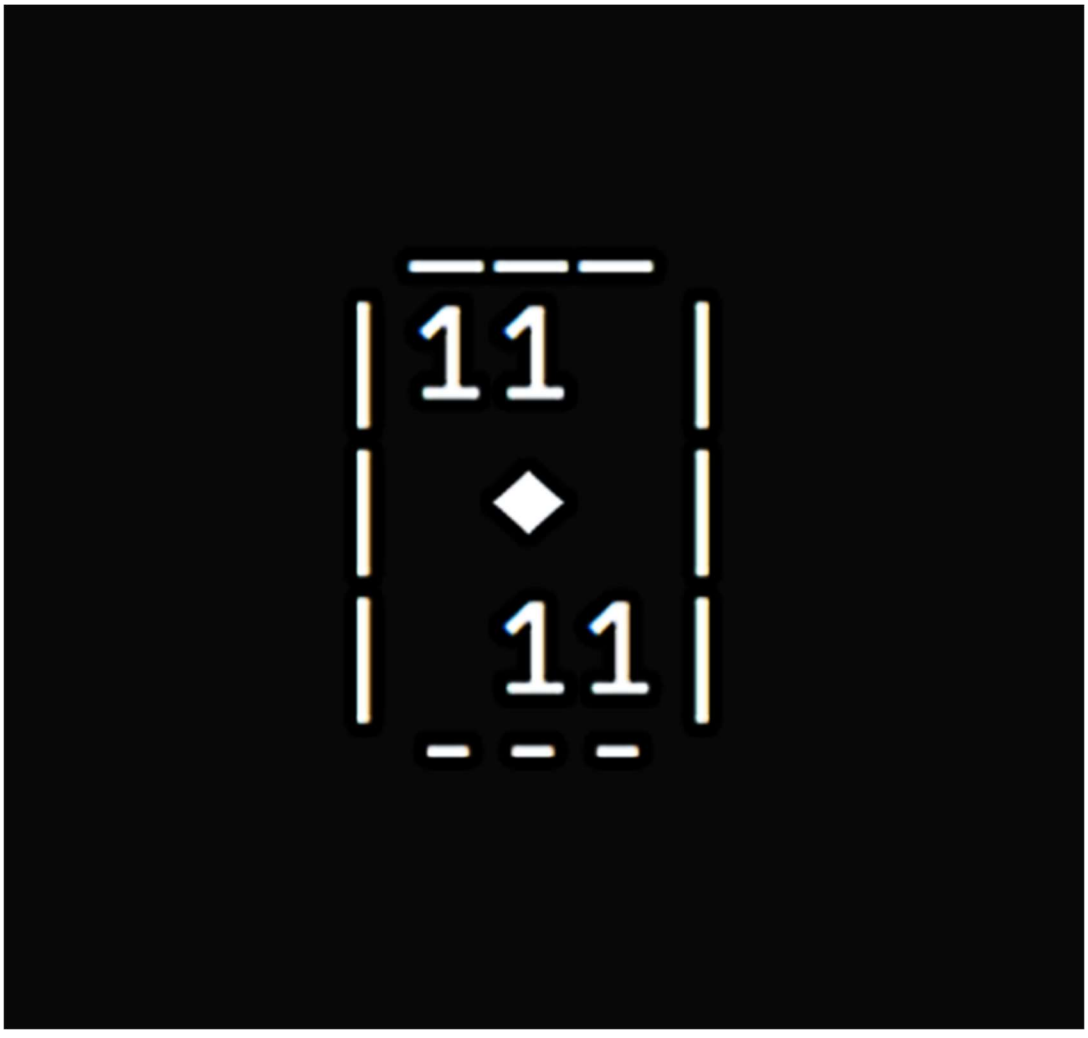
Manual Técnico/Documentación
ONCE DE DIAMANTES

Autor:

Brandon Ramirez Bedoya



Agosto 2024

Tabla de contenido

INTRODUCCIÓN

Una de dos: o eres alguien interesado en el Desarrollo de Software, Ingeniería y/o Ciencias Computacionales, o por el otro lado eres alguien curioso que solo quiere ver cómo funciona este maravilloso videojuego desde dentro. Bienvenido al Manual Técnico que también sirve como Documentación de Once de Diamantes.

¿Qué se puede encontrar en este manual?

A lo largo de este manual encontrarás información muy interesante acerca del desarrollo de Once de Diamantes, las aplicaciones que se utilizaron, explicación detallada de bloques de código, e incluso parte de su historia principal. Esto y entre otros temas no menos importantes.

Repositorio GitHub y página oficial.

Ten en cuenta que Once de Diamantes cuenta con su propio repositorio alojado en la plataforma de GitHub, si quieres aprender bien como funciona todo desde dentro te recomendamos descargar todos los archivos de GitHub y abrir el proyecto en Apache NetBeans IDE 19 para analizar la programación detalladamente. Igualmente en este manual se estará explicando gran parte del código.

No olvides que este videojuego puedes descargarlo desde la página oficial “oncedediamantes.com” junto con su Manual de Usuario, además de encontrar los apartados donde podrás contactarte con los involucrados en este proyecto.

HISTORIA

En este apartado del manual encontrarás, en resumen, tanto la historia de cómo surgió Once de Diamantes y como fue creado, como también la historia que este videojuego tiene para ofrecerle al jugador, está demás decir que sobran los Spoilers.

Idea y desarrollo.

Yo, Brandon Ramirez Bedoya, creador de Once de Diamantes, puedo decir que el desarrollo de este videojuego fue un tanto particular, y a la vez un ejemplo de lo que NO se debe hacer cuando se trabaja en un proyecto de software. En el segundo semestre de mi carrera de Desarrollo de Software tuve que atender una materia la cual se centraba en el desarrollo de programas con el lenguaje de programación Java, y a su vez veíamos la programación Funcional y Orientada a Objetos.

Esta materia requería la presentación de un proyecto final con el que pudiésemos demostrar nuestros conocimientos adquiridos, en este caso mi profesor tenía la brillante idea de que desarrollásemos un videojuego de cartas, que contase puntos y que los imprimiese en un archivo de texto. He de admitir que una parte de la idea de este videojuego proviene de mi profesor, yo fui quien la tomé y la pulí a tal punto de crear un producto similar pero con funciones nuevas e interesantes. Así nace Once de Diamantes.

Crear este videojuego fue una pesadilla, ya que en el desarrollo de proyectos de software siempre debes contar con una exhaustiva planificación, programación y un buen levantamiento de requerimientos si lo que buscas es que tu producto sea asombroso y memorable para quienes lo prueban, y precisamente esto no pasó con Once de Diamantes. Al principio seguiría las ordenes de mi profesor al pie de la letra, pero pronto empecé a modificar un poco lo que se esperaba del producto final, inicié implementando historia al videojuego y agregando otras funciones, luego llegaron ideas como desarrollar un manual, para después querer desarrollar una página web, y luego llevar este videojuego a un repositorio y… bueno, entre muchísimas cosas más.

Estuvo mal. Acatar todo ese montón de ideas sin una planificación adecuada genera problemas innumerables en el desarrollo de un proyecto, aún me pregunto cómo no abandoné Once de Diamantes. A pesar de todo, veo en este videojuego una oportunidad para demostrar mis habilidades en la programación y he conseguido tenerle a lo largo del tiempo un inmenso cariño.

Luego de un año de desarrollo, Once de Diamantes está listo.

Personajes de Once de Diamantes.

Aquí comienzan los Spoilers, ¡mucho cuidado!

- **Tú (???).**
  Saber quién es el protagonista de esta historia siempre ha sido un gran interrogante. ¿Qué tipo de carta es? ¿Cuál es su número? ¿Cuál es su color?

Estas han sido miles y miles de preguntas que se han hecho los habitantes del Reino Naipe, ¿quién es nuestro Héroe?
Afortunadamente, esa es una duda la cual el propio jugador deberá darle una respuesta a lo largo de su travesía por proteger a los reyes y salvar a toda la población del reino. ¿Eres el Héroe que cuidará al Reino Naipe de Once de Diamantes y sus secuaces? Pruébalo…

- **3 de Trébol Negro (3♣3).**
  Esta peculiar carta será nuestro amigo durante toda nuestra historia, aquel que estará constantemente comunicándose con nosotros durante nuestro viaje. Al principio será un poco complicado ganarnos su confianza, pero con suerte lograremos obtener su respeto.
- **Once de Diamantes (11♦11).**
  Los enemigos de nuestra historia, aquellos que solo han llegado con intenciones malévolas. Once de Diamantes es el nombre con el que se refiere a un grupo de cartas criminales que quieren acabar con el Reino Naipe, con su población, y con lo más importante para nuestra historia: los Reyes. Lo peor de todo es que este bando cuenta con una extraña pero poderosa habilidad: poder camuflarse y pasar desapercibidos entre las demás cartas.
- **Reyes (K♣K - Q♥Q).**
  Nuestro objetivo primordial y las cartas más importantes en nuestra historia. El Rey (Trébol) y la Reina (Corazón) han dado al Reino Naipe tanto suerte como amor a todos sus habitantes a lo largo de su existencia.
- **Reino Naipe (población).**
  Imagina ser el Héroe amado por toda la población del Reino Naipe. Sus habitantes viven con la esperanza de que llegue algún día una carta valiente para hacerle frente a todas las adversidades que puedan presentarse y afectar a sus hogares.

Historia de Once de Diamantes.

¡Bienvenido al Reino Naipe!

Gracias al duro e intenso trabajo de los supremos reyes, el Reino Naipe había logrado obtener paz y armonía en todos sus habitantes, ¡por fin era un gran lugar para vivir! Pero no todo era color de rosa, ya que con la llegada de un extraño y peligroso bando de cartas criminales conocidos como Once de Diamantes toda la paz y prosperidad del pueblo se vería afectada. Once de Diamantes buscaba hacer el mal en el reino y acabar con la vida de los supremos reyes.

Cada uno de los integrantes de Once de Diamantes cuenta con la habilidad de ser capaces de camuflarse entre otras cartas normales, ¡esto suena muy peligroso para la salud y protección del reino y de los reyes! Incluso se rumoreaba de como cartas fuertes y poderosas como las Jotas y Ases habían caído ante ellos. Era hora de hacer algo al respecto…

Afortunadamente y como si hubiesen caído del cielo, una conocida carta 3 de Trébol Negro junto con otra carta desconocida se aventuraron juntos a liderar la guerra contra Once de Diamantes, uno protegería al reino, y el otro protegería a los supremos reyes.

Fue una guerra intensa y dura… pero al final el Reino Naipe logró prosperar de nuevo, y toda su población en general, incluidos sus supremos reyes que pudieron ver una vez más la luz de un nuevo amanecer. Pocos integrantes de Once de Diamantes que aún quedaban vivos nunca volverían a acercarse al reino, porque sabían que unos Héroes los estarían esperando.

Sé el Héroe que el Reino Naipe se merece…

PROGRAMAS INVOLUCRADOS

En este apartado del manual encontrarás todos los programas y webs involucradas en el desarrollo de Once de Diamantes.

Replit.com

Esta web fue fundamental al principio del desarrollo de este videojuego, aquí se escribió gran parte del código e incluso estuvo disponible la versión Beta de Once de Diamantes para todo público. La versión Beta ya no se encuentra disponible.

Apache NetBeans IDE 19.

Este entorno de desarrollo fue pieza clave en la programación de Once de Diamantes, todo el código de Replit fue transferido a NetBeans para su posterior continuación donde fueron agregadas más funciones y donde se perfeccionó en lo posible el funcionamiento de este videojuego. Fue necesario la instalación de Java y el JDK 20 para que, cuando ya Once de Diamantes estuviese completamente listo, se pudiese ejecutar en la terminal de Windows.

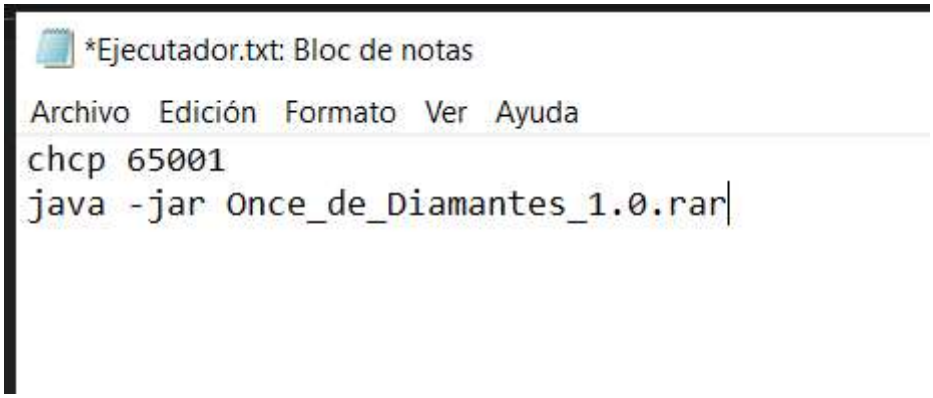
Bat To Exe Converter.

Cuando desarrollas proyectos de software siempre buscas que tu producto sea lo más practico posible para los usuarios/interesados, y Once de Diamantes no se podía quedar atrás en cuanto a esto. No bastaba con darle a las personas un simple código para que se ejecutase, es por eso que era necesario encontrar una solución, y Bat To Exe Converter lo sería. Al principio, este videojuego se ejecutaba en la terminal de Windows por medio de un archivo .bat, el proceso era el siguiente:

1. Tomar el archivo .jar del videojuego.

2. Convertirlo a .rar cambiando su nombre.

3. Crear un nuevo archivo de texto .txt (preferiblemente llamarlo “Ejecutador”) y escribir los códigos que se ven en pantalla.



4. Posteriormente guardar el archivo y convertirlo a un .bat

Y listo, de esta manera cada vez que abres el Ejecutador.bat estarás jugando Once de Diamantes y a su vez se te creará automáticamente en esa carpeta el archivo posiciones.txt (la Tabla de Posiciones).

Esto estaba bien, pero iba a estar mucho mejor cuando el Ejecutador.bat pudiese convertirse en un programa .exe el cual perfectamente puede compartirse con las personas que quieres, y adicionalmente poder crearle su propio instalador.

Bat To Exe Converter se encargaba de convertir este archivo .bat en un .exe

Inno Setup Compiler.

Con el .exe ya creado, perfectamente tendríamos la versión Portable preparada, sin embargo, existen quienes gustan de tener sus programas instalados en su dispositivo, es por eso que Inno Setup Compiler sería el programa encargado de tomar todos los archivos necesarios para convertirlos en un simple instalador, de esta manera obteniendo la versión de Instalador lista para compartirla con las personas.

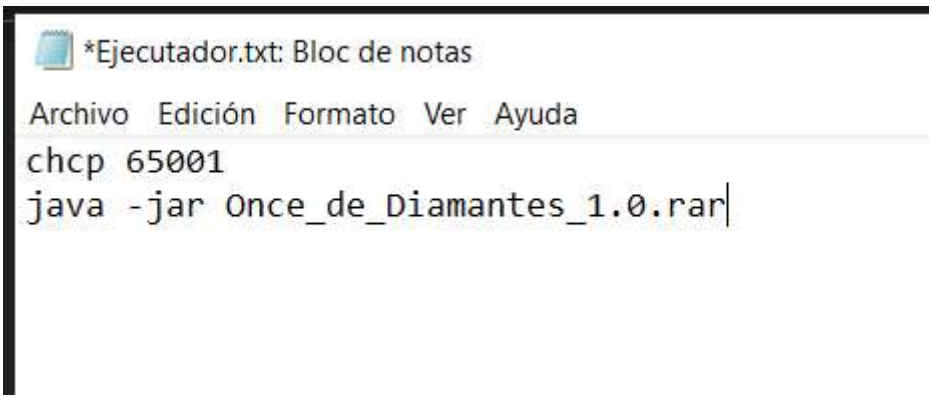
GitHub.com

Finalmente, y por lo que probablemente estás aquí, GitHub es la plataforma en la nube por excelencia de repositorios para proyectos de software. Ahí siempre estará Once de Diamantes y todo su código.

CÓDIGOS A FONDO

En esta sección del manual (y probablemente la más interesante) aprenderás sobre lo que ocurre en varios bloques del código cuando ejecutas este videojuego, y alguna que otra aclaración/explicación.

Códigos de ejecución y CHCP.



Como se mencionó anteriormente, en un principio un archivo .bat es el encargado de la ejecución del código del videojuego para que el usuario pueda disfrutar de este, y posteriormente este archivo pasa a ser convertido a un .exe para luego ser compilado en un instalador. Pero, ¿qué son exactamente esos códigos que contiene el archivo .bat para ejecutar Once de Diamantes?

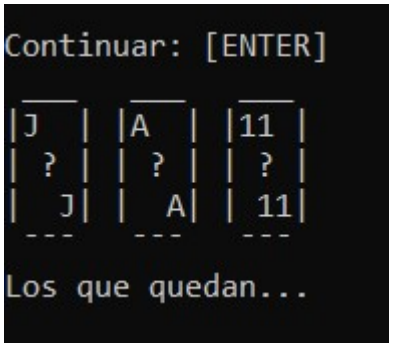
chcp 65001.

Por defecto el CMD de Windows no está configurado para imprimir en pantalla los caracteres más importantes del juego, en este caso son los símbolos de Naipe de las cartas (♥♣♦♠). Para solucionar este problema se debe hacer uso de algo conocido como “página de código”.

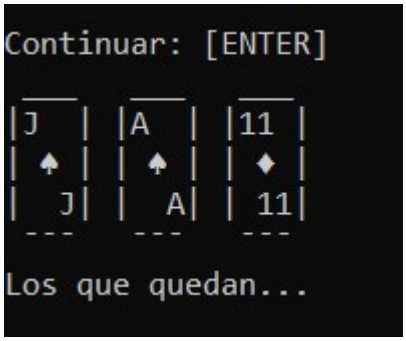
Las páginas de código son asignaciones de caracteres especiales que pueden servir dependiendo el contexto o idioma en que deban de utilizarse. Con ellas se podrían imprimir en pantalla caracteres únicos de idiomas distintos los cuales el CMD no podría imprimir por defecto, esto por poner un ejemplo.

En este caso, una página de códigos que puede imprimir todos nuestros caracteres necesarios (y los más importantes como los símbolos de Naipes) es la página de códigos chcp 65001. Precisamente al escribir en primer lugar este comando nos aseguramos de que CMD implemente los caracteres especiales antes de realizar cualquier acción.

- Sin chcp 65001



- Con chcp 65001



Es importante aclarar que el CMD volverá a su estado normal cuando Once de Diamantes se cierre, es por eso que cada vez que abres el videojuego, chcp 65001 se ejecutará automáticamente en primer lugar.

java -jar Once_de_Diamantes_1.0.rar

Luego de que el CMD ya está preparado para imprimir nuestros caracteres necesarios, por medio del comando en la segunda línea "java -jar Once_de_Diamantes_1.0.rar" le pedimos al CMD que ejecute con Java el código de Once de Diamantes. A pesar de ser un comando simple, este es precisamente el detonante que nos permite, a nosotros los usuarios, jugar este videojuego.

Es importante que el archivo .rar se encuentre en la misma ubicación que el ejecutador, sino es así será necesario modificar el código para que CMD no tenga problemas en encontrar el archivo.

.BAT y los Antivirus (solución).

Probablemente podrías haber tenido problemas a la hora de instalar/ejecutar Once de Diamantes y por eso estás aquí. Si bien todos los casos son diferentes, muchas veces lo que está en juego es la protección que ofrece Windows y los diferentes sistemas antivirus instalados en el dispositivo. Al ser el archivo .bat un programa que ejecuta estos comandos en el CMD de Windows, a pesar de ser simples códigos que no tienen la intención de ser dañinos, los sistemas antivirus o Microsoft SmartScreen podrían detectar esta acción como una posible amenaza peligrosa en el sistema, ósea, se entiende que Windows y los demás sistemas quieran protegerte

a ti y a tu dispositivo de cualquier amenaza. Quizás sea esto la posible causa de tu problema. Para solucionar esto puedes desactivar tus antivirus/SmartScreen ya que podrían afectar negativamente el funcionamiento del juego.

Códigos de Once de Diamantes.

Luego de haber visto los códigos que permiten la ejecución de Once de Diamantes, es hora de visualizar como se ve este videojuego desde dentro, y que es lo que hace cada bloque de código (te recomendamos tener el código a la mano).

Importar librerías.

```
1  /*Zona de Imports*/
2  import java.util.Scanner;
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.Comparator;
6  import java.util.Random;
7  import java.io.*;
8  import java.io.UnsupportedEncodingException;
9  import java.text.SimpleDateFormat;
10 import java.util.Calendar;
```

Para el correcto funcionamiento del código de Once de Diamantes será necesario la importación de las siguientes librerías de Java que aparecen en la imagen. Cada una cumple un rol importante en la ejecución.

Arreglo para la Tabla de Posiciones.

```
12 /*Clase Main (única)*/
13 public class Main{
14
15     /*ArrayList que se imprimirá en el archivo "posiciones.txt"*/
16     static ArrayList<String> cadenas = new ArrayList<>();
17
```

Este arreglo, denominado como “cadenas” es el responsable de guardar todas las posiciones que se imprimirán en el archivo posiciones.txt, y por lo tanto, serán las posiciones que aparecerán en la Tabla de Posiciones.

Orden de las cartas.

```
18      /*Funciones para imprimir las cartas*/
19      static String orden1(){
20        String a=" ___   ";
21        String b1 = "|K  | ";
22        String b2 = "|11 | ";
23        String b3 = "|Q  | ";
24        String c1 = "| ♣ | ";
25        String c2 = "| ♦ | ";
26        String c3 = "| ♥ | ";
27        String d1 = "|  K| ";
28        String d2 = "| 11| ";
29        String d3 = "|  Q| ";
30        String e = " ___   ";
31        String cadena=a+a+a+"\n"+b1+b2+b3+"\n"+c1+c2+c3+"\n"+d1+d2+d3+"\n"+e+e+e;
32        return cadena;
33      }
34
```

Tendremos en nuestro código un total de 6 métodos denominados “orden”, estos bloques de código permiten que, cuando en nuestras partidas el juego debe imprimir el orden en que salen nuestras cartas, ejemplo: Rey en primera posición, Once en segunda posición y Reina en tercera posición. El código llamará al método determinado que proporcione el orden que se busca que se imprima en pantalla.

Cartas “boca abajo”.

```
115     static String desconocido(){
116       String a=" ___   ";
117       String b4="|?  | ";
118       String c4="| ? | ";
119       String d4="|  ?| ";
120       String e=" ___   ";
121       String cadena=a+a+a+"\n"+b4+b4+b4+"\n"+c4+c4+c4+"\n"+d4+d4+d4+"\n"+e+e+e;
122       return cadena;
123     }
```

No hay tanta ciencia detrás de este método, este solo se manifiesta cuando es necesario imprimir las 3 cartas de la partida cuando están boca abajo.

Cartas de la historia.

```
125     /*Funciones para imprimir la historia*/
126     static String historia1(){
127       String a=" ___   ";
128       String b1 = "|K  | ";
129       String b3 = "|Q  | ";
130       String c1 = "| ♣ | ";
131       String c3 = "| ♥ | ";
132       String d1 = "|  K| ";
133       String d3 = "|  Q| ";
134       String e = " ¯¯¯   ";
135       String Historia1=a+a+"\n"+b1+b3+"\n"+c1+c3+'
136       return Historia1;
137     }
138
```

Y seguimos con los métodos de impresión de cartas, en este caso son 3 denominados “historia” y sirven para imprimir en pantalla las cartas que aparecen cuando usas la opción [H] en el menú principal.

“Cargando…”

```
165     /*Función para esperar 1000 milisegundos (1 segundo)*/
166     static void esperar1(){
167       try {
168         Thread.sleep(millis:1000);
169       } catch (InterruptedException e) {}
170     }
```

¿Enserio? ¿Pantallas de carga en un juego de texto? ¡Así es!

Hay un total de 4 métodos denominados “esperar” que tienen como objetivo esperar una determinada cantidad de segundos cada vez que son llamados, ejemplo: cuando el jugador ingresa la opción [S] para salir del juego, el código llamará a “esperar3()” para que el usuario pueda leer el mensaje final de despedida en 3 segundos, y pasado ese tiempo el juego concluirá su ejecución.

Fecha y hora.

```
193     /*Fecha y hora en String*/
194     static String hora(){
195       Calendar tiempoActual=Calendar.getInstance();
196       SimpleDateFormat sdf=new SimpleDateFormat(pattern: "dd-MM-yyyy 'a las' HH:mm:ss");
197       String date=sdf.format(date: tiempoActual.getTime());
198       String escrito="Se jugó el "+date;
199       return escrito;
200     }
201
```

Finalmente y antes de pasar a la clase Main, tenemos este bloque de código que nos permite obtener la información de tiempo del dispositivo del usuario en un formato especifico que luego será impreso en cada una de las posiciones de la Tabla de Posiciones en el archivo posiciones.txt.

Void Main.

```
202     /*Main, Scanner, y definir variables*/
203     public static void main(String[] args) throws UnsupportedEncodingException{
204       Scanner s=new Scanner(source:System.in, charsetName: "ISO-8859-1");
```

Ahora iniciamos con todo lo que se estará ejecutando en tiempo real. Aplicaremos un “UnsupportedEncodingException” en nuestro Main y adicionalmente usaremos “ISO-8859-1” como charsetName en nuestro Scanner (bautizado simplemente como “s”), todo esto con el objetivo de no tener problemas a la hora de imprimir caracteres especiales.

Lectura y escritura de posiciones (parte 1).

```
211     /*Archivo y lectura*/
212     File miArchivo = new File(pathname: "posiciones.txt");
213     try {
214       FileReader fileReader = new FileReader(file: miArchivo);
215       BufferedReader reader;
216       reader = new BufferedReader(in: fileReader);
217       int dosdigitos=0;
218       while ((linea = reader.readLine()) != null) {
219           dosdigitos++;
220           if (dosdigitos>=1000){
221               dividir=linea.substring(beginIndex: 6);
222               cadenas.add(e: dividir);
223               reinicioTabla=true;
224           }else if (dosdigitos>=100 && dosdigitos<1000){
225             dividir=linea.substring(beginIndex: 5);
226             cadenas.add(e: dividir);
227           }else if (dosdigitos>=10 && dosdigitos<100){
228             dividir=linea.substring(beginIndex: 4);
229             cadenas.add(e: dividir);
230           }else{
231             dividir=linea.substring(beginIndex: 3);
232             cadenas.add(e: dividir);
233           }
234       }
```

Este bloque de código se encarga de leer todo el contenido que exista en el archivo posiciones.txt y lo almacenará en el arreglo “cadenas” que anteriormente habíamos definido, esto con el fin de poder jugar con los datos que ya estaban en la tabla y que no se lleguen a perder cada vez que se abre el juego. Adicionalmente este código verifica si no se han sobrepasado las 1000 posiciones, de lo contrario la variable “reinicioTabla” pasará a ser verdadera.

Lectura y escritura de posiciones (parte 2).

```
235     Collections.sort(list:cadenas, new Comparator<String>() {
236         @Override
238         public int compare(String cadena1, String cadena2) {
239             int numero1 = Integer.parseInt(cadena1.split(regex: " ")[0]);
240             int numero2 = Integer.parseInt(cadena2.split(regex: " ")[0]);
241             return Integer.compare(x: numero2, y: numero1);
242         }
      });
```

Este bloque se encargará de poner orden en el arreglo “cadenas”, con el fin de que los nombres que tengan mayor cantidad de puntos sean los que estarán en posiciones elevadas.

Lectura y escritura de posiciones (parte 3).

```
243         try {
244             if (reinicioTabla==true){
245                 FileWriter fileWriter = new FileWriter(file: miArchivo, append: false);
246                 BufferedWriter writer;
247                 writer = new BufferedWriter(out: fileWriter);
248                 writer.write(str: linea2);
249                 writer.close();
250             }else{
251             FileWriter fileWriter = new FileWriter(file: miArchivo, append: false);
252             BufferedWriter writer;
253             writer = new BufferedWriter(out: fileWriter);
254             int posicionTabla=1;
255             for (int m=0; m<cadenas.size(); m++){
256                   writer.write(posicionTabla + ". " + cadenas.get(index: m));
257                   writer.newLine();
258                   posicionTabla++;
259             }
260             writer.close();
261             }
262         } catch (IOException e) {
263           System.out.println(x: e.getMessage());
264         }
265       reader.close();
266     } catch (IOException e) {
267       System.out.println(x: e.getMessage());
268     }
```

Finalmente, este bloque de código (aún dentro del “try{}” de la parte 1) se encarga de escribir en el archivo de texto las posiciones del arreglo “cadenas”. Esto último pasa SOLO si “reinicioTabla” se encuentra falso, ya que si se encuentra verdadero entonces el archivo posiciones.txt se reiniciará completamente. Once de Diamantes solo está programado intencionalmente para recibir no más de 999 posiciones.

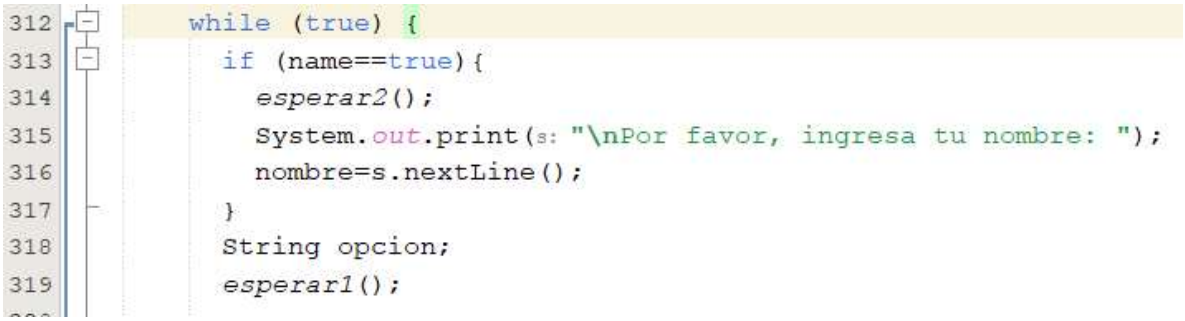
Variables de posición.

```
270         /*Otras variables*/
271         String nombre="";
272         String carta1="""
273                       ___
274                      |K  |
275                      | ♠ |
276                      |  K|
277                       ___
278                      """;
279         String carta2="""
280                       ___
281                      |11 |
282                      | ♦ |
283                      | 11|
284                       ___
285                      """;
286         String carta3="""
287                       ___
288                      |Q  |
289                      | ♥ |
290                      |  Q|
291                       ___
292                      """;
293         String carta0="""
294                       ___
295                      |3  |
296                      | ♠ |
297                      |  3|
298                       ___""";
299         Random random=new Random();
300         int puntaje=0;
301         String Posicion1;
302         boolean name=true;
```

En este bloque se encuentran definidas las variables con las que trabajaremos a la hora de encontrar a Once de Diamantes, como las cartas y la variable “random” que permitirá mover las cartas aleatoriamente. Adicionalmente aparece la variable clave “puntaje” definida en 0 la cual será de vital importancia para el avance de nuestras

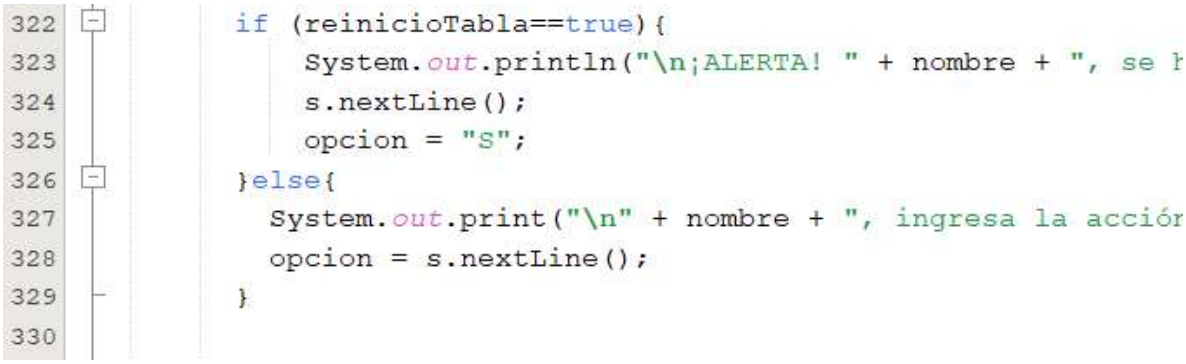
partidas. Esta última variable estará contando la cantidad de puntos que estaremos ganando conforme la partida avance.

El ciclo While.



Luego de una breve introducción a Once de Diamantes, en el código se presenta el inicio de este ciclo While que estará acompañando al usuario desde el inicio hasta el final (literalmente). Es gracias a este ciclo While que el jugador disfruta de Once de Diamantes hasta el momento en que el juego se cierra de cualquier forma. En este bloque se pide el nombre de usuario, ya sea porque el juego apenas se abrió o porque el jugador usó la opción de cambiar de nombre [C]. Y aquí tendremos la variable “opcion” que dependiendo que letra se ingrese sucederán determinadas acciones.

Las 1000 posiciones.



Volviendo con el tema de las más de 999 posiciones en el archivo posiciones.txt, si la variable “reinicioTabla” se encuentra activa entonces se imprimirá en pantalla un mensaje de alerta y automáticamente el juego se cerrará en el momento en que el usuario presione [ENTER]. Con tener “reinicioTabla” activado la tabla se reiniciará y el juego se cerrará. De lo contrario la ejecución sigue su rumbo normal.

Opción [J].

```
        if (opcion.equals(anObject: "J")) {
333         String[] baraja=new String[]{carta1, carta2, carta3};
334         for (int i=baraja.length-1; i>0; i--){
335             int indicerandom=random.nextInt(i+1);
336             String temp=baraja[i];
337             baraja[i]=baraja[indicerandom];
338             baraja[indicerandom]=temp;
339         }
340         int q=0;
341         if (baraja[q+2].equals(anObject: carta3) && baraja[q].equ
342             System.out.print(s: orden1());
343         }else if (baraja[q].equals(anObject: carta1) && baraja[q+
344             System.out.print(s: orden2());
345         }else if (baraja[q].equals(anObject: carta2) && baraja[q+
346             System.out.print(s: orden3());
347         }else if (baraja[q].equals(anObject: carta2) && baraja[q+
348             System.out.print(s: orden4());
349         }else if (baraja[q].equals(anObject: carta3) && baraja[q+
350             System.out.print(s: orden5());
351         }else if (baraja[q].equals(anObject: carta3) && baraja[q+
352             System.out.print(s: orden6());
353         }
354
```

Hora de comenzar con la partida: al ingresar la opción [J] la baraja de cartas se moverá aleatoriamente gracias a “random”. Al tener ya el orden aleatorio y dependiendo de cómo hayan quedado, será el “if” y todos los “else if” los encargados de la impresión en pantalla de las cartas.

Siguiente cambio.

```
363             /*Se cambia aleatoriamente la posición de las cartas*/
364             System.out.print(s: "\n¡AQUÍ VIENEN LOS INTERCAMBIOS!\n");
365             System.out.print(s: "\n");
366             int cambios=0;
367             while (cambios<6){
368                 int indice1=random.nextInt(bound: 3);
369                 int indice2=random.nextInt(bound: 3);
370                 while (indice1==indice2) {
371                     indice2 = random.nextInt(bound: 3);
372                 }
373                 String temp = baraja[indice1];
374                 baraja[indice1] = baraja[indice2];
375                 baraja[indice2] = temp;
376                 cambios+=2;
377                 esperar1();
378 
379                 /*Se hacen 3 cambios y se imprimen*/
380                 System.out.println("La carta en la posición [ " + (ind
381             }
382             cambios=0;
```

En total son 2 cambios los que se hacen, el primero lo acabamos de hacer al iniciar la partida, el segundo está a punto de concretarse ya con las cartas puesta boca abajo, esto para dar la ilusión de que estuvieron moviéndose todo este tiempo.

Durante el segundo cambio (que serán 6 en total en 2 ciclos While) se imprimirá en pantalla que posiciones (1, 2, 3) se movieron.

Números incorrectos.

```
400             int posicion;
401             while (true){
402                 System.out.print(s: "\nIngresa el número de posición en el que consideras que podría
403                 posicion=s.nextInt();
404                 if (posicion<=3 && posicion>=1){
405                     posicion=posicion-1;
406                     break;
407                 }else{
408                     System.out.println(x: "Incorrecto. Por favor, ingresa un número entre 1 y 3.");
409                 }
410             }
411 
```

Aquí le pedimos al usuario que nos indique del 1 al 3, luego de haber hechos los cálculos de donde se encuentra Once de Diamantes, en que posición podría estar la dichosa carta. No obstante, en el desarrollo de proyectos de software siempre se debe tener en cuenta todas las posibilidades, y con ello estamos refiriéndonos a que podrían ingresarse otros números o caracteres que no fuesen 1, 2 o 3. Para remediar eso este bloque de código se encarga de asegurarse de que el usuario no ingrese otro número que no sean esos 3.

¡Posición correcta! :D

```
412             /*Se verifica si la posición ingresada es correcta o no*/
413             boolean aviso=false;
414             for (int i=0; i<baraja.length; i++){
415                 if (baraja[i].equals(anObject: carta2) && i==posicion){
416                     System.out.println(x: "\n¡Es correcto, ganaste! :D");
417                     puntaje+=1;
418                     name=false;
419                     System.out.println("Tu puntaje actual es de: " + puntaje + " punto(s).");
```

Este bloque de código nos contabiliza un punto y nos muestra en pantalla nuestra cantidad de puntaje recolectado. Primero verifica que la posición que elegimos fuese la correcta, luego contabiliza el punto, mantiene la variable “name” en falso (si estuviese en verdadero estaríamos obligados a cambiar nuestro nombre) y nos muestra en pantalla un mensaje.

Nuevo diálogo de nuestro mejor amigo.

```
421             /*Mensajes que se imprimen en pantalla referente a la historia dependiendo el puntaje del jugador*/
422             if (puntaje==5){
423                 esperar1();
424                 System.out.println(x: "\n- Veo que me equivoqué contigo al principio, espero seguir así de equivo
425             }
```

Luego de terminar la partida actual, el juego verifica nuestra cantidad de puntos y dependiendo de cuantos tengamos nos saldrá un diálogo de 3 de Trébol Negro que sirve como aquel medio por el que sabemos lo que acontece en la historia. Cada 5 puntos aparecerá un nuevo diálogo de él, esto hasta llegar a 50 puntos donde el juego termina y pasan los créditos.

Posición incorrecta :(

```
525             if (baraja[i].equals(anObject: carta2) && i!=posicion){
526                 name=false;
527                 aviso=true;
528                 System.out.println(x: "\nIncorrecto, has perdido :(");
529                 System.out.println("Tu puntaje ha sido de: " + puntaje + " punto(s).");
```

Si el jugador elige la posición incorrecta seguirá teniendo la variable “name” en falso (ya que no es necesario cambiar su nombre a no ser que el jugador lo quiera), tendrá la variable “aviso” en verdadero la cual se encargará de darle un ultimo diálogo de 3 de Trébol Negro al jugador por haber perdido. Se imprimirá en pantalla la elección incorrecta y los puntos que el jugador tuvo alguna vez. Luego de este bloque de código vendrá otro similar al de **Lectura y escritura de posiciones** el cual se encargará de ingresar a la Tabla de Posiciones toda la información de la partida perdida.

Independientemente de si el jugador perdió o ganó, a continuación en otro siguiente bloque de código se imprimirá en pantalla la posición final en que quedaron las 3 cartas, ahora volteadas “boca arriba”. Y adicionalmente se imprimirá en pantalla el ultimo diálogo de 3 de Trébol Negro si el jugador perdió.

Luego de todo esto, el jugador vuelve a tener a su disposición todas las opciones del menú de inicio.

Opción [T].

```
602             /*Presionando la tecla [T] se imprime la tabla de posiciones*/
603         } else if (opcion.equals(anObject: "T")) {
604             System.out.println(x: "\nPosiciones de Once de Diamantes: ");
605             if (puntaje>=1){
606             }
607             if (reinicioTabla==true){
608                 cadenas.clear();
609             }else{
610                 for (String cadena : cadenas) {
611                     System.out.println(x: cadena);
612                 }
613             }
614             System.out.println(x: "Fin de las posiciones (Juega hasta que p:
615             name=false;
```

Al usar la opción [T] el juego imprimirá en pantalla todas las posiciones registradas en el arreglo “cadenas” que sería lo mismo a lo que se encuentra en el archivo posiciones.txt, cabe recalcar que es necesario haber jugado al menos más de una partida y haber obtenido puntos.

Opción [H].

```
617         /*Presionando la tecla [H] se imprimen los textos de la historia cada que el jugador presiona [ENTER]*/
618         } else if (opcion.equals(anObject: "H")) {
619             esperar1();
620             System.out.println(x: historia1());
621             System.out.print(s: "\nContinuar: [ENTER] ");
622             s.nextLine();
623             System.out.println(x: historia2());
624             System.out.print(s: "\nContinuar: [ENTER] ");
625             s.nextLine();
626             System.out.println(x: historia3());
627             System.out.print(s: "\nTerminar y volver al menú: [ENTER] ");
628             s.nextLine();
629             name=false;
```

No hay mucha ciencia detrás de esta opción. Únicamente se encarga de imprimir en pantalla los textos e historia que ya habíamos escrito anteriormente.

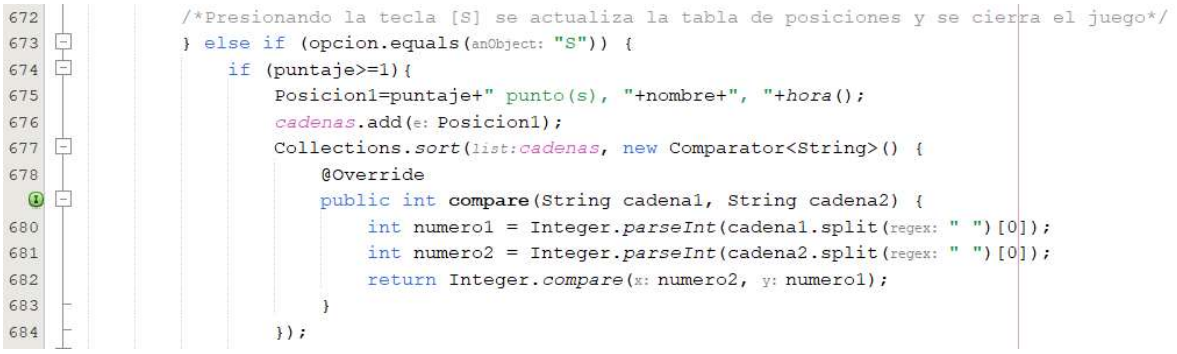
Opción [C].

```
631         /*Presionando la tecla [C] se actualiza la tabla de posiciones y se permite al usuario cambiar de nombre*/
632         } else if (opcion.equals(anObject: "C")) {
633             if (puntaje>=1){
634                 Posicion1=puntaje+" punto(s), "+nombre+", "+hora();
635                 cadenas.add(e: Posicion1);
636                 Collections.sort(list:cadenas, new Comparator<String>() {
637                     @Override
                    public int compare(String cadena1, String cadena2) {
639                     int numero1 = Integer.parseInt(cadena1.split(regex: " ")[0]);
640                     int numero2 = Integer.parseInt(cadena2.split(regex: " ")[0]);
641                     return Integer.compare(x: numero2, y: numero1);
642                 }
643             });
```

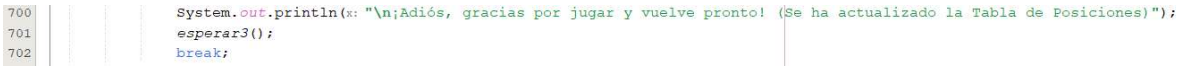
La penúltima opción de todas: [C]. Esta opción primero hace un guardado con los 3 bloques de código necesarios para ingresar información en el archivo

posiciones.txt, para luego pasar la variable “name” a verdadera y dejar nuestro “puntaje” en 0. Por último imprime en pantalla un mensaje de que la tabla se ha actualizado. El jugador es consciente de que al usar esta opción su progreso se reiniciará.

Opción [S].

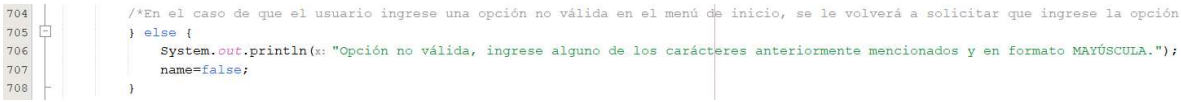


Esta opción es casi lo mismo a la opción [C], en lo que se diferencian ambas es que mientras que [C] tiene el objetivo de que el jugador inicie desde 0 con un nuevo nombre a su gusto, [S] directamente rompe con un “break” el **ciclo While**, cerrando el juego de esta manera.



No sin antes dar un mensaje de despedida (como nos gusta), esperar 3 segundos y finalmente cerrarse.

Opción [desconocida].



¿Creías que eso era todo? No. Como se había mencionado anteriormente, siempre hay que pensar en todas las posibilidades, entonces ¿qué pasaría si el jugador ingresa una opción incorrecta en el menú? Sencillamente se imprime en pantalla el mensaje que aparece en la imagen y aparecerá las veces que sean necesarias hasta que el jugador ingrese una opción válida.

Ultílogo de CÓDIGOS A FONDO.

Y ahí va, así se ve Once de Diamantes desde dentro.

Si bien no se mostró COMPLETAMENTE TODO el código del juego, se omitieron partes que eran innecesarias enseñar en esta documentación, o que directamente eran reutilizadas en otros puntos de la estructura, así que hubiese sido redundante.

Ahora se sabe cuantas veces se sobrescribe y se guardan datos en la Tabla de Posiciones, cuáles son los códigos que permiten la ejecución de Once de Diamantes, y en general todo su funcionamiento como tal.

EL ARCHIVO POSICIONES.TXT

Siempre se menciona el archivo posiciones.txt, pero nunca se habla de él a fondo.

Este archivo es el responsable de llevar nuestra Tabla de Posiciones bien organizada y de servir como el sistema de guardado del juego. Este archivo siempre deberá permanecer en el mismo directorio en que se ubica el archivo .exe de Once de Diamantes.

En caso contrario de que este archivo no se encuentre ahí, y que por lo tanto nuestro dispositivo no sea capaz de encontrarlo, el código de este videojuego será capaz de generarlo por si mismo y de dejarlo automáticamente en la ubicación donde debería de estar.

Al inicio de la ejecución de Once de Diamantes, el arreglo “cadenas” leerá y guardará toda la información de posiciones.txt, para que luego durante la partida y con la inserción de nuevos puntajes y nombres, “cadenas” sobrescriba todo este archivo desde cero con toda su nueva información. Y así es como funciona, toda la información que antes estaba registrada no se perderá e incluso interactuará con las nuevas posiciones y puntajes.

- ¿Se puede modificar? Si, pero NO es lo más recomendable. Si realizas una mala modificación del archivo podrías llegar a romper Once de Diamantes y tendrías que eliminar este archivo por completo. Si lo que quieres es presumir que llegaste a muchísimos puntos pues juega Once de Diamantes todo lo que puedas y ya. No hagas trampas.
- ¿Qué pasa si lo elimino? Pierdes todo el progreso que hubieses tenido registrado, pero igualmente el mismo videojuego se encargará de generarlo cuando se registre una nueva posición. Así que es una buena opción para cuando quieres reiniciar tu Tabla de Posiciones sin la necesidad de tener que llegar hasta las 1000 posiciones.

VERSIONES (Instalación y Desinstalación)

Los pasos a seguir para la descarga/instalación de cualquiera de las versiones de Once de Diamantes se encuentra en el Manual de Usuario, puedes encontrarlo en la página oficial “oncedediamantes.com”.

No hay diferencias notables entre cada versión, el descargar una u otra ya dependerá de tus gustos personales en el manejo de descargas.

Versión Portable.

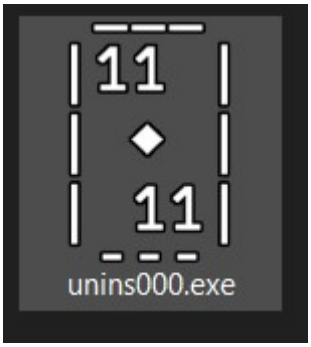
Al descargar esta versión únicamente tendrás que descomprimir el archivo .rar y eso será todo. En la carpeta que obtendrás vas a encontrar todos los archivos necesarios para la ejecución de Once de Diamantes.

En el caso de querer eliminar todo contenido de este videojuego de tu ordenador solo deberás de eliminar la carpeta del juego y eso será todo, obviamente sin olvidar vaciar la Papelera de reciclaje de tu dispositivo.

Versión de Instalador.

Al descargar esta versión solo tendrás que descomprimir el archivo .rar el cual te entregará el programa instalador. Ejecuta el programa y sigue todos los pasos del asistente de instalación. Una vez en tu computadora podrás incluso tener su icono en el escritorio.

En el caso de querer eliminar Once de Diamantes de tu computadora solo tendrás que dirigirte a la ubicación del juego, aquella ubicación la cual permitiste que se instalase. Si por alguna razón tienes problemas en encontrar donde se encuentra Once de Diamantes instalado puedes hacer clic derecho sobre su icono y hacer clic en “Abrir la ubicación del archivo”.



Busca el archivo “**unins000.exe**”, ejecútalo y sigue los pasos que el programa te pida. Y listo, Once de Diamantes se habrá ido.

PUNTOS IMPORTANTES

Para finalizar con la documentación de este videojuego, en este apartado encontrarás una conclusión de los puntos más importantes vistos en este manual.

1. El sitio web oficial de Once de Diamantes es “oncedediamantes.com” y proporciona enlaces al repositorio en GitHub.com
2. La Tabla de Posiciones solo guarda información en 3 momentos: al perder, al cambiar de usuario y al salir del juego.
3. Se recomienda no manipular el archivo posiciones.txt
4. Es necesario la ejecución de comandos como “chcp 65001” para jugar este videojuego.
5. Es gracias a la ejecución de estos comandos desde un archivo .bat que probablemente varios sistemas antivirus lo detecten como malicioso.
6. Este juego se termina al llegar a 50 puntos, luego de ahí será decisión del jugador si sigue recolectando puntos o si cambia de acción.
7. Debes proteger a los Reyes para salvar el Reino Naipe, afortunadamente 3 de Trébol Negro siempre estará ahí para ayudarnos. ¡Arriba el Reino, abajo Once!

El Reino Naipe te necesita…