

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS

CATEDRÁTICO: ING. WILLIAM ESTUARDO ESCOBAR ARGUETA

TUTOR ACADÉMICO: JOSUÉ RODOLFO MORALES CASTILLO



ENNER ESAÍ MENDIZABAL CASTRO

CARNÉ: 202302220

SECCIÓN: B

GUATEMALA, 16 DE FEBRERO DEL 2,024

ÍNDICE

ÍNDICE	1
INTRODUCCIÓN	2
OBJETIVOS	2
1. GENERAL	2
2. ESPECÍFICOS	2
ALCANCES DEL SISTEMA	2
ESPECIFICACIÓN TÉCNICA	3
• REQUISITOS DE HARDWARE	3
• REQUISITOS DE SOFTWARE	3
DESCRIPCIÓN DE LA SOLUCIÓN	4
LÓGICA DEL PROGRAMA	7
❖ MiPractical	
Captura de las librerías usadas	
➤ Librerías	7
➤ Variables Globales de la clase _(MiPractical)	8
➤ Función Main	8
➤ Métodos y Funciones utilizadas	9

INTRODUCCIÓN

Este manual fue creado con la finalidad de dar a entender los distintos métodos y estrategias que se utilizaron en la creación de TABLERO MATEMÁTICO, principalmente enfocado hacia un público más técnico el cual esté interesado en este tipo de aspectos del juego.

OBJETIVOS

1. GENERAL

- 1.1. Proporcionar información técnica acerca del programa informático

2. ESPECÍFICOS

- 2.1. Objetivo 1: Permitir un mayor entendimiento de los métodos numéricos usados para la solución de los problemas.
- 2.2. Objetivo 2: Introducir al lector a la lógica abstracta que se llevó a cabo en la programación de TABLERO MATEMÁTICO.

ALCANCES DEL SISTEMA

Este manual busca dar todo lo que un usuario experimentado o técnico pueda requerir para entender, comprender, estructurar o implementar el mismo tipo de métodos y estrategias utilizadas para la creación de TABLERO MATEMÁTICO junto a una descripción de los recursos más técnico que se requieren para utilizar este juego.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**

- Procesador: 2 GHz o superior
- Disco duro: Espacio se al menos 1.5 GB
- RAM: 2GB o superior

- **REQUISITOS DE SOFTWARE**

- Sistema operativo: Windows, macOS o Linux
- Java development kit (JDK)
- Entorno de desarrollo integrado (IDE) compatible con Java (se recomienda NetBeans)

DESCRIPCIÓN DE LA SOLUCIÓN

- Para generar el TABLERO MATEMÁTICO, se pensó en la manera en que funcionaba un juego, primero con un menú, y luego con cada menú realizando las distintas acciones, por lo que a partir de esa idea se comenzó con un *switch case* el cual fungía como el menú principal, luego de esto se decidió ir opción por opción realizando lo que fuera necesario para que trabajara como se esperaba. En la opción en donde más atención se puso fue en la de *iniciar juego*, dado a que en ella se desarrollaba la mayor parte del tiempo de juego, por lo que dentro de ella creó un sistema que podía generar una matriz de tipo *String* la cual contenía todos los valores que permitían que se viera como un tablero, de este modo, cada vez que se iniciara un juego se crearía nuevamente una matriz. Luego de haber creado esto, se decidió realizar el mismo procedimiento, pero con una matriz booleana, que iría paralela a la matriz del tablero la cual permitiría generar aleatoriamente las penalizaciones.
- **CREACIÓN DEL JUGADOR:** Una vez con eso listo, se comenzó con la creación del jugador, la cual se hizo por medio del carácter de: “@”, por lo que, al ser la matriz de tipo *String*, para realizar el movimiento, se que se concatenara el “@” en la posición nueva en donde estaría, además, para mantener solo un arroba, se crearon variables las cuales guardaban el valor de la posición antes de que el “@” se concatenara, para sí cuando este cambiara de lugar, estas variables se puedan encargar de regresar el valor a como era previamente.
- **LÓGICA DEL MOVIMIENTO:** El movimiento se logró por medio de una variable encargada de recibir el valor proporcionado por el dado y restárselo a otra variable dedicada a la modificación de la posición del jugador, además, cuando eran necesario que se escalara de fila, se decidió colocar un condicional que leería la posición de la columna, y si esta iba a

desbordarse, le restaba 8 (la cantidad de valores que tiene la columna) y se restaba una fila (pues todo va de atrás para adelante). Adicionalmente, con ese método de la matriz parecía que “se teletransportaba” a la otra esquina de la matriz, dado a la misma configuración de estas, por lo que, para solucionar ese problema visual, se decidió mostrar la matriz de una forma en la que, si el número de fila era par, entonces se mostraba de atrás hacia adelante, logrando de ese modo que se mejore la estética.

- **EMPLEO DE LA ALEATORIEDAD:** Todo lo aleatorio dentro del programa se logró por medio de la librería *util.random*, la cual proporcionó los valores para que el jugador se moviera, las penalizaciones aparecieran y entre otras cosas que requerían de esa aleatoriedad.
- **LECTURA DEL MOVIMIENTO:** Para el movimiento se usó la letra “e” y “p” para la pausa, en este caso, la ejecución de esos fue mediante un *switch case* el cual dirigiría a la opción deseada.
- **PENALIZACIONES:** Para las penalizaciones, la cual quizá es la parte más importante del programa, se realizó un condicional el cual verificaría si la posición del jugador se encontraba justo donde había una penalización, y dado caso sí lo estuviera, se verificaría la fila para saber qué tipo de penalización se requeriría para luego mandar el método pertinente a la penalización
- **LEY DE COSENOS:** Para la penalización de la ley de cosenos se decidió realizar un método que devolviera un valor que sería un ángulo, y otro que devolvería un lado, ambos hechos a partir de la ecuación de la ley de cosenos despejada a conveniencia.
- **SUMA DE MATRICES:** Para la penalización de la suma de matrices, simplemente se realizó un bucle que recorrería cada valor de la matriz y lo sumaría al de la misma posición de la otra matriz a sumar.
- **DIVISIÓN DE MATRICES:** Por último, para la división de matrices, la que fue la más difícil, se decidió realizar varios métodos que permitirían

un mayor orden a la hora de realizar la operación, primero se realizó el método de cofactores, el cual se dedicaría a hallar el valor de los cofactores dependiendo de la posición a la que se indique que se necesita. Este método de cofactores se realizó por medio de la regla de Sarrus para matrices 3×3 . Luego se realizó el método de determinante, el cual simplemente se dedicó a identificar los primero cuatro elementos de la matriz, y encontrar sus cofactores para luego hallar la determinante operándolos. A posteriori, se halló la matriz adjunta por medio del método de cofactores, recorriendo cada uno de los valores de la matriz y hallando su cofactor que sería colocado en la matriz adjunta. Por último, se halló la inversa por medio de la ecuación que multiplica el recíproco de la determinante de la matriz por la transpuesta. Y se multiplico por la matriz que se solicitaba para así hallar la división de matrices

En general, esa fue la lógica principal que se utilizó para la creación del programa, pero hay que dejar claro que es un programa extenso con unas ideas abstractos que no se pueden expresar de manera clara desde un párrafo, pero con estas pocas ideas se espera que como lector se pueda comprender la lógica vertebral del programa TABLERO MATEMÁTICO.

LÓGICA DEL PROGRAMA

❖ MiPractica1

➤ Librerías

```
package mipractical;  
  
import java.util.Random;  
import java.util.Scanner;
```

Para la creación del programa se utilizaron las librerías *util.Random* y *util.Scanner*, la primera se utilizó para la generación de los valores aleatorios dentro del juego, como fueron las penalizaciones en el tablero o los valores que arroja el dado que permite que se mueva el jugador; y la segunda se utilizó para recibir los datos por medio del teclado.

➤ Variables Globales de la clase _(MiPractica1)

```
static double[][] rD = new double[4][4];
static double[][] rdD = new double[4][4];
static boolean verificado;

static double dA[][] = {
    {5, 10, 1, 3},
    {9, 14, 2, 6},
    {7, 8, 15, 3},
    {6, 8, 9, 2}
};

static double dB[][] = {
    {5, 13, 9, 4},
    {1, 9, 6, 3},
    {8, 11, 69, 33},
    {25, 6, 7, 4}
};

static double dC[][] = {
    {1, 12, 9, 8},
    {7, 6, 3, 2},
    {0, 5, 6, 14},
    {6, 9, 6, 10}
};

static double dD[][] = {
    {8, 19, 20, 4},
    {12, 33, 6, 8},
    {4, 5, 9, 7},
    {8, 22, 14, 6}
};

static int[][] matrizC = {
    {4, 9, 7, 45, 18},
    {7, 51, 26, 8, 38},
    {48, 26, 37, 21, 19},
    {1, 0, 6, 8, 1},
    {2, 19, 55, 25, 15}};

static int[][] matrizD = {
    {0, 2, 15, 1, 66},
    {21, 48, 62, 7, 33},
    {4, 88, 0, 68, 4},
    {25, 18, 24, 7, 55},
    {24, 15, 36, 5, 98}};

static int[][] matrizR = new int[5][5];
static int[][] matrizRR = new int[5][5];
static int[][] matrizA = {
    {7, 48, 5, 0, 1},
    {57, 8, 4, 6, 14},
    {0, 5, 6, 78, 15},
    {21, 14, 8, 19, 54},
    {32, 20, 26, 47, 12}};

static int[][] matrizB = {
    {9, 5, 2, 1, 8},
    {4, 2, 3, 47, 8},
    {48, 55, 32, 19, 6},
    {7, 56, 32, 14, 8},
    {32, 87, 0, 1, 7}};

static boolean VerificoM;

static int[][] matrizE = {
    {0, 1, 15, 5, 36},
    {1, 78, 65, 32, 4},
    {48, 66, 39, 0, 55},
    {14, 98, 63, 20, 15},
    {11, 39, 84, 7, 1}};

static int[][] matrizF = {
    {78, 25, 66, 48, 98},
    {0, 45, 2, 3, 1},
    {2, 9, 14, 10, 20},
    {35, 87, 65, 2, 32},
    {25, 8, 4, 9, 39}};
```

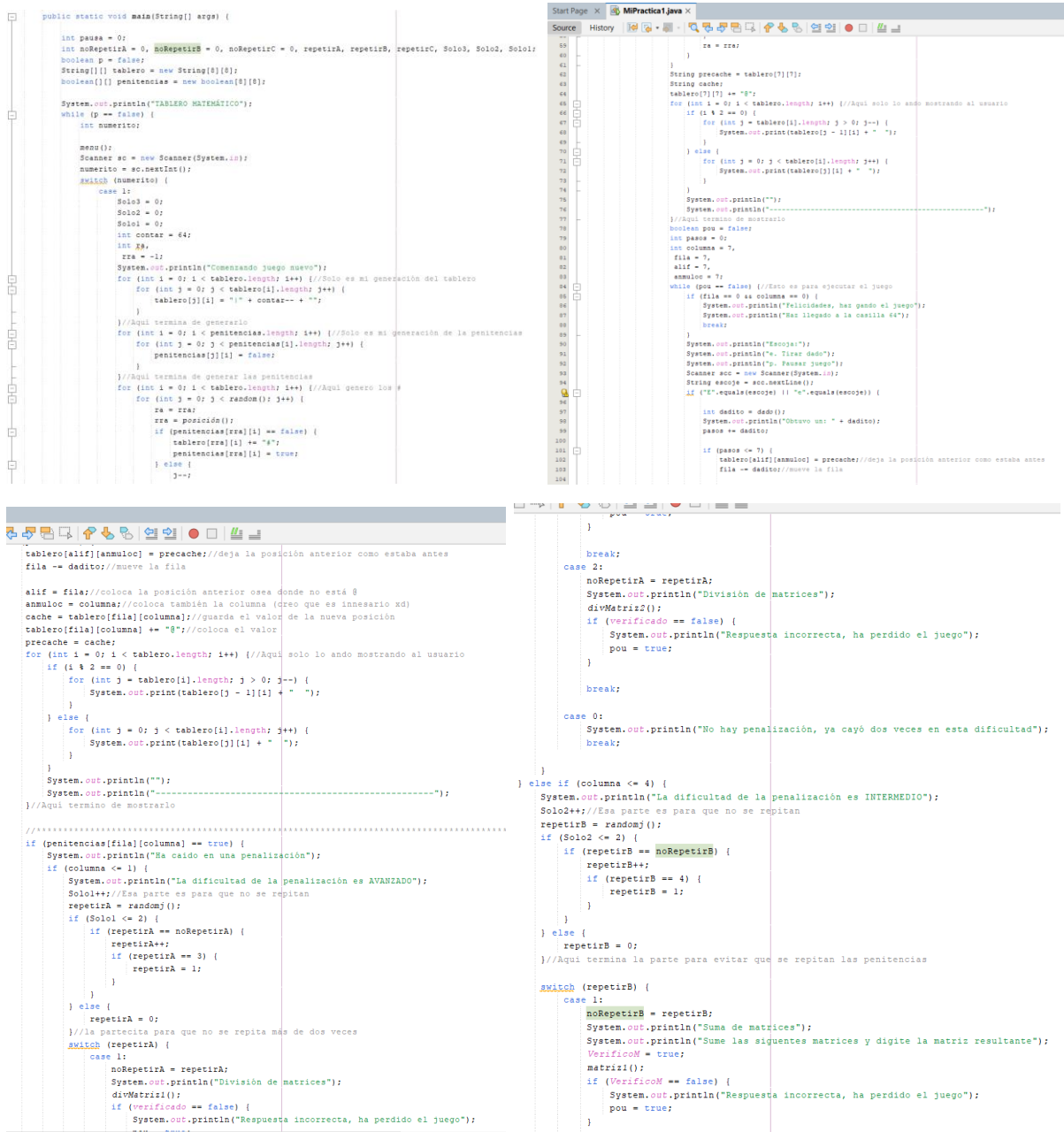
Todas estas variables fueron creadas con la finalidad de ser utilizadas por método ajenos a la función main, los cuales luego serán llamado por ella. Son principalmente los valores usados para la ejecución de las penalizaciones

➤ Función Main

```
15 public static void main(String[] args) { ...624 lines }
```

Dentro de esta función es la cual se ejecuta la espina dorsal del código, en esta se tiene cada una de las condicionales y ciclos que hacen posible la partida de TABLERO MATEMÁTICO.

Principalmente se encarga de la creación del tablero, sus penalizaciones y el movimiento del jugador que, cuando cae en alguna casilla con penalización, llamará un método externo que se encargará de la ejecución de su respectiva penitencia entre muchas cosas más hechas a partir de la utilización de condicionales y bucles.



```

public static void main(String[] args) {
    int pausa = 0;
    int noRepetirA = 0, noRepetirB = 0, noRepetirC = 0, repetirA, repetirB, repetirC, Solo3, Solo2, Solo1;
    boolean p = false;
    String[][] tablero = new String[8][8];
    boolean[][] penitencias = new boolean[8][8];

    System.out.println("TABLERO MATEMÁTICO");
    while (p == false) {
        int numerito;

        menu();
        Scanner sc = new Scanner(System.in);
        numerito = sc.nextInt();
        switch (numerito) {
            case 1:
                Solo3 = 0;
                Solo2 = 0;
                Solo1 = 0;
                int contar = 64;
                int rA,
                rB = -1;

                System.out.println("Comenzando juego nuevo");
                for (int i = 0; i < tablero.length; i++) { //Solo es la generación del tablero
                    for (int j = 0; j < tablero[i].length; j++) {
                        tablero[j][i] = " " + contar-- + " ";
                    }
                }
                //Aquí termina de generar
                for (int i = 0; i < penitencias.length; i++) { //Solo es la generación de la penitencias
                    for (int j = 0; j < penitencias[i].length; j++) {
                        penitencias[j][i] = false;
                    }
                }
                //Aquí termina de generar las penitencias
                for (int i = 0; i < tablero.length; i++) { //Aquí genero los %
                    for (int j = 0; j < random(); j++) {
                        rA = rA;
                        rB = posición();
                        if (penitencias[rA][i] == false) {
                            tablero[rA][i] += "R";
                            penitencias[rA][i] = true;
                        } else {
                            j++;
                        }
                    }
                }

                tablero[alif][anuloc] = precache; //deja la posición anterior como estaba antes
                fila -- dadito; //mueve la fila

                alif = fila; //coloca la posición anterior osea donde no está R
                anuloc = columna; //coloca también la columna (digo que es innecesario xd)
                cache = tablero[fila][columna]; //guarda el valor de la nueva posición
                tablero[fila][columna] += "R"; //coloca el valor
                precache = cache;

                for (int i = 0; i < tablero.length; i++) { //Aquí solo lo ando mostrando al usuario
                    if (i % 2 == 0) {
                        for (int j = 0; j < tablero[i].length; j > 0; j--) {
                            System.out.print(tablero[j] - 1)[i] + " ";
                        }
                    } else {
                        for (int j = 0; j < tablero[i].length; j++) {
                            System.out.print(tablero[j][i] + " ");
                        }
                    }
                }
                System.out.println("");
                System.out.println("-----");
            } //Aquí termino de mostrarlo

            //*****
            if (penitencias[fila][columna] == true) {
                System.out.println("Ha caído en una penalización");
                if (columna <= 1) {
                    System.out.println("La dificultad de la penalización es AVANZADO");
                    Solo1++; //Esa parte es para que no se repitan
                    repetirA = random();
                    if (Solo1 <= 2) {
                        if (repetirA == noRepetirA) {
                            repetirA++;
                            if (repetirA == 3) {
                                repetirA = 1;
                            }
                        } else {
                            repetirA = 0;
                        }
                    } //La parte para que no se repita más de dos veces
                    switch (repetirA) {
                        case 1:
                            noRepetirA = repetirA;
                            System.out.println("División de matrices");
                            divMatriz1();
                            if (verificado == false) {
                                System.out.println("Respuesta incorrecta, ha perdido el juego");
                                sc = null;
                            }
                    }
                }
            }
        }
    }
}

//*****
}

break;
case 3:
    noRepetirA = repetirA;
    System.out.println("División de matrices");
    divMatriz2();
    if (verificado == false) {
        System.out.println("Respuesta incorrecta, ha perdido el juego");
        pou = true;
    }
    break;

case 0:
    System.out.println("No hay penalización, ya cayó dos veces en esta dificultad");
    break;

} else if (columna <= 4) {
    System.out.println("La dificultad de la penalización es INTERMEDIO");
    Solo2++; //Esa parte es para que no se repitan
    repetirB = random();
    if (Solo2 <= 2) {
        if (repetirB == noRepetirB) {
            repetirB++;
            if (repetirB == 4) {
                repetirB = 1;
            }
        } else {
            repetirB = 0;
        }
    } //Aquí termina la parte para evitar que se repitan las penitencias
    switch (repetirB) {
        case 1:
            noRepetirB = repetirB;
            System.out.println("Suma de matrices");
            System.out.println("Suma las siguientes matrices y digite la matriz resultante");
            VerificoM = true;
            matriz1();
            if (VerificoM == false) {
                System.out.println("Respuesta incorrecta, ha perdido el juego");
                pou = true;
            }
        }
    }
}

```

➤ Métodos y Funciones utilizadas

A continuación, se dará una explicación general de lo que hace cada función:

```

1144
1145 + static void divMatriz1() {...59 lines }
1204
1205 + static void divMatriz2() {...60 lines }
1265

```

Estas dos funciones se encargan de la ejecución de la división de matrices los dos tipos de ejercicios posibles dentro del programa

```

906 + static double determinante(double[][] matriz) {...7 lines }
913 + static double cofactor(double[][] matri, int position, int position2) {...205 lines }
1118
1119 + static double[][] adjunta(double[][] matri) {...9 lines }
1128
1129 + static double[][] inversa(double[][] matrizinha) {...11 lines }
1140

```

Estas funciones, como sus nombres lo indican, se encargan de hallar la determinante, cofactor, adjunta e inversa que se utilizan en la división de fracciones.

```

159 public static void readJSON() {

```

Breve descripción del uso de la función o para qué sirve

```

831 + static void matriz3() {...50 lines }//Aquí acaban la suma de matrices ---
881 //También comienzo a trabajar con lo de la división de matrices -----

```

```

766
767 + static void matriz2() {...51 lines }

```

```

703 + static void matriz1() {...51 lines }

```

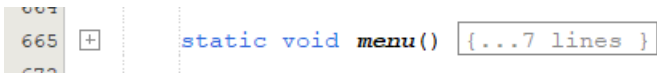
Estos métodos se encargan mostrar y ejecutar la penalización de suma de matrices, uno por cada una de las opciones.

```

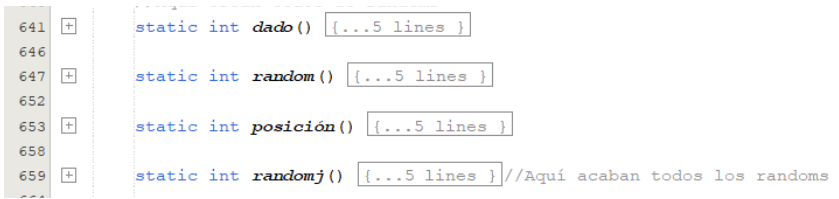
673 + static float opción1de1(float n1, float n2, float a) {...5 lines }
678
679 + static float opción2de2(float l, float l2, float l3) {...5 lines }

```

Para la generación de los problemas de la ley de coseno, se usan estos dos métodos los cuales se encargan de hallar un ángulo por medio de la ley de coseno y de hallar un lado con la ley de cosenos despejada.

-  `static void menu() {...7 lines }`

Simplemente muestra el menú principal

-  `static int dado() {...5 lines }`
`static int random() {...5 lines }`
`static int posición() {...5 lines }`
`static int randomj() {...5 lines } //Aquí acaban todos los randoms`

Todos estos métodos se encargan de la misma función, que es genera valores aleatorios para sus distintos propósitos, como el dado, la posición de las penalizaciones, el número de estas y así.