



Instituto Politécnico Nacional
Unidad Profesional Interdisciplinaria de Ingeniería y
Ciencias Sociales y Administrativas

Algoritmos Computacionales

Unidad III: Proyecto Final

Proyecto departamental: Documentación

Gomez Castillo Miguel Angel

mikee.asdasd@gmail.com

5532406106

Ramírez Estrada Ignacio

ignaciowwe2010@hotmail.com

5514326611

Reyes Pérez Hannia Yael

hannia.reyes0105@gmail.com

5539240950

Tirado Zamudio Carlos Alfonso

carponch78@gmail.com

5565319019

3NM40

22 de Enero de 2021

Resumen

El presente documento contiene el proposito general del proyecto a realizar para el *Tercer Departamental*. Al igual que las especificaciones y el tipo de usuarios a los cuales va a ir dirigido nuestro programa. Adicionalmente se agrega la vista de escenarios, la cual describe de forma detallada el comportamiento de cada escenario planteado en la aplicación.

Índice general

1. Introducción	1
1.1. Descripción general	1
1.2. Propósito	2
1.3. Contenido	2
2. Manual de Usuarios	3
2.1. PASO 1. Entrar al link de descarga	3
2.2. PASO 2. Descargar archivo	4
2.3. PASO 3. Descomprimir carpeta	5
2.4. PASO 4. Abrir y ejecutar	5
3. Manual Técnico	7
3.1. Requerimientos funcionales	7
3.2. Requerimientos no funcionales	8
3.3. Arquitectura	8
3.3.1. Diagrama de Casos de Uso	8
3.3.2. Diagrama de Secuencia	9
3.4. Código Fuente	10
4. Historial de Construcción del Programa	18
4.1. Diseño Preliminar	18
4.1.1. Pruebas a Realizar del Sistema	19
4.1.2. Reporte de Pruebas del Revisor y el Usuario Final . .	19
4.1.3. Cambios Accordados a la Idea Inicial	19
Bibliografía	19

Índice de figuras

2.1. GitHub - Opciones	3
2.2. Opcion - Descarga	4
2.3. Guardar Archivo	4
2.4. Carpeta .ZIP	5
2.5. Descomprimir .ZIP	5
2.6. Juego Sudoku	6
3.1. Diagrama de Casos de Uso	9
3.2. Diagrama de Secuencia	9
4.1. Diseño Preliminar	18
4.2. Diseño Final - Pantalla Principal	20
4.3. Diseño Final - Pantalla Secundaria	20

Capítulo 1

Introducción

En el presente capítulo se muestra la definición general del proyecto, el propósito, el alcance del documento y una descripción general del proyecto planteado, así como de los capítulos posteriores.

1.1. Descripción general

Como proyecto final del tercer departamental de la secuencia 3NM40 de la materia *Algoritmos Computacionales* de la carrera de Ingeniería en Informática, se realizó un programa capaz de emular a la perfección el juego japonés "Sūji wa dokushin ni kagiru", que significa "los dígitos están limitados a una ocurrencia", inventado en 1984.

La finalidad del programa es digitalizar un juego que básicamente es en papel, cuyo objetivo es estructurar en cuadrículas divididas en cajas de 3x3 celdas en las que hay algunos números escritos de antemano. Para jugar, simplemente debes rellenar las celdas en blanco de tal forma que cada fila, columna y caja de 3x3 no tenga números repetidos.

El programa debe ser capaz de darle al usuario causal, intermedio o experto la experiencia de jugar un sudoku pero en versión digital y con las mismas reglas de juego.

El alcance del programa varía en cuestión a las edades, pero cabe resaltar que es un excelente juego pasatiempo.

1.2. Propósito

El presente documento tiene como propósito describir en detalle los requerimientos del proyecto, como el *Manual de Usuario*, el *Manual Técnico* y el *Historial de Construcción del Sistema Archivos*.

1.3. Contenido

A continuación se describe el contenido de los capítulos que conforman el presente documento.

El capítulo 1 da una pequeña introducción al documento, de lo que contiene y el objetivo de crear este proyecto.

El capítulo 2 describe la manera en que el usuario debe instalar el programa, las funciones que tiene, la manera en que se opera y con que plataformas es compatibles.

El capítulo 3 contiene la manera en que debe instalarse el programa desde el punto de vista técnico, los requerimientos y el código fuente.

El capítulo 4 contiene el historial de construcción de nuestro programa, al igual que la especificación inicial, el diseño preliminar de las pantallas con datos reales, las pruebas que se le harán al sistema, el reporte de pruebas del revisor y el usuario final incluyendo sus respectivos cambios realizados.

Por último, la bibliografía contiene el listado de referencias tomadas en cuenta para la elaboración del programa.

Capítulo 2

Manual de Usuarios

Nuestro código se encuentra en la página oficial de GitHub, donde estara disponible para cualquier usuario que requiera tanto el código, como el juego en sí. El usuario podra hacer la descarga en GitHub de la siguiente manera:

2.1. PASO 1. Entrar al link de descarga

- Link de descarga: <https://github.com/Nacho1410/Sudoku.git>
- El usuario debera entrar a link donde se podra observar tanto el código, como las múltiples opciones que te ofrece la página, alguna de ellas son:
 1. Go to File. Para hacer la busqueda individual de los archivos.
 2. Add File. Esto te permite agregar un archivo siempre y cuando se tengan los permisos necesarios.
 3. Code. Esta opción es el que dejara descargar todos los archivos del programa, comprimiendolos en una carpeta .ZIP

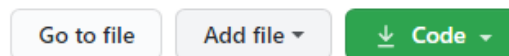


Figura 2.1: GitHub - Opciones

2.2. PASO 2. Descargar archivo

- Nombre del archivo: ProyectoSudoku.cpp
- Al ser el unico archivo disponible solo es necesario darle click al boton CODE. Para que este pueda descargarlo y comprimirlo en una carpeta .zip.

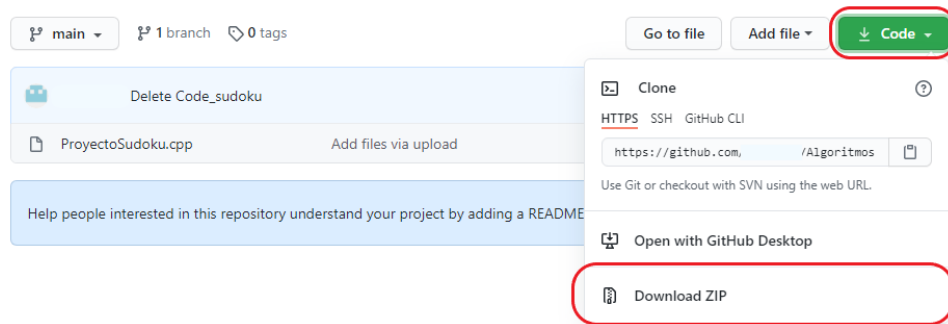


Figura 2.2: Opcion - Descarga

- Y te pedira que elijas en que parte guardara la carpeta.

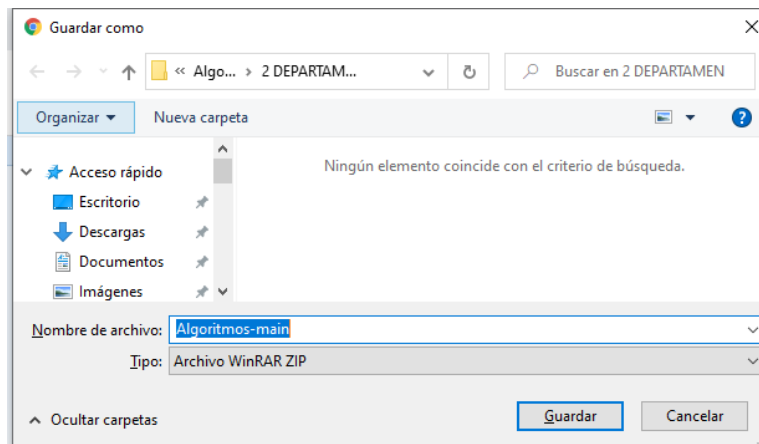


Figura 2.3: Guardar Archivo

2.3. PASO 3. Descomprimir carpeta

- Al finalizar la descarga, en la parte inferior izquierda, podras visualizar el archivo descargado.
- En el cual se dara click en la flecha para que te lleve a la carpeta en donde se guardo y puedas proceder a descomprimi el .ZIP, dando click derecho sobre el y elegir **Extraer aqui**. Esto sera de utilidad para poder empezar a usar el programa de Sudoku.

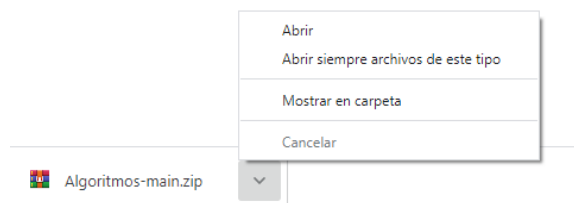


Figura 2.4: Carpeta .ZIP

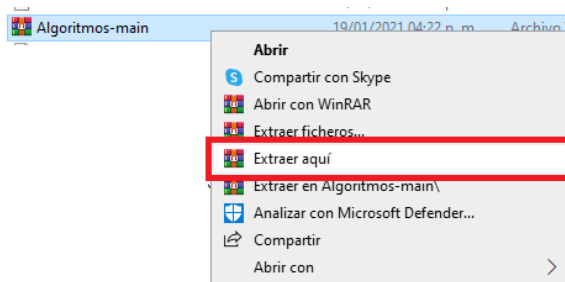
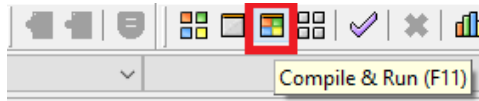


Figura 2.5: Descomprimir .ZIP

2.4. PASO 4. Abrir y ejecutar

- Cuando la carpeta se haya descomprimido, podras acceder a ella y abrir el archivo llamado *Proyecto Sudoku*. El cual mostrara el código de forma estructurada. Para esto deberas tener instalado previamente **Dev C++** o algún programa similar que pueda permitirte ejecutar el código.
- Al abrir el archivo, dentro de las opciones de **Dev C++**:

1. Primero se da click en la opción de ejecutar y compilar de **Dev C++**. Con el objetivo de que el programa lo lea y verifique que no hay errores.



2. El juego se abra en otra ventana de manera automática como se muestra en la Figura 2.6.

```
-- SUDOKU A RESOLVER --  
4 2 9      6 0 0      5 8 1  
0 0 1      4 0 9      6 7 3  
6 7 3      5 8 1      4 2 0  
  
9 4 2      0 6 7      0 5 8  
1 5 8      9 0 2      3 0 0  
0 0 0      1 5 8      9 0 0  
  
2 9 4      7 0 0      0 1 5  
8 1 5      2 0 0      7 0 0  
0 3 6      8 1 5      2 9 0  
  
-----  
Presione una tecla para continuar . . .
```

Figura 2.6: Juego Sudoku

Capítulo 3

Manual Técnico

Este documento contiene toda la información sobre los recursos utilizados por el proyecto, explicado todo el trabajo que se ha realizado al desarrollar un sistema llevan una descripción muy bien detallada sobre las características físicas y técnicas de cada elemento..

3.1. Requerimientos funcionales

RF01. Algoritmos Computacionales: La base

El programa debe permitir visualizar al usuario un sudoku real en donde solo se dé la base para poder resolverlo manualmente. .

RF02. Algoritmos Computacionales: Reglas del juego

El programa solo desplegara como salida la base inicial del sudoku en donde las reglas son:

1. Regla 1: hay que completar las casillas vacías con un solo número del 1 al 9.
2. Regla 2: en una misma fila no puede haber números repetidos.
3. Regla 3: en una misma columna no puede haber números repetidos.
4. Regla 4: en una misma región no puede haber números repetidos.
5. Regla 5: la solución de un sudoku es única

RF03. Algoritmos Computacionales: Fin del juego.

El programa debe permitir dar al usuario la respuesta del sudoku en donde

para determinar si un Sudoku está bien resuelto, se tendrá que ir corrigiendo cada fila por separado y comprobar que dicha fila solo tiene números del 1 al 9 y no están repetidos. Si esto se cumple, el Sudoku es correcto.

3.2. Requerimientos no funcionales

A continuación se describen los requerimientos no funcionales asociados al proyecto.

NF01. Sistema operativo Windows 10

La elección del sistema operativo Windows 10 se debe al número de usuarios que actualmente usan dicho sistema, abarcando un total del 35.83 % de la cuota del mercado, según cifras de mayo del 2020 por parte del sitio web StatCount. Otra de las razones por las que se optó por este S.O. es su facilidad de manipulación por parte de los usuarios debido al entorno gráfico que maneja.

Las PC con Windows 10 son rápidas y duraderas, tienen una autonomía increíble y ofrecen un fantástico diseño a un precio atractivo. Una nueva PC con Windows 10 es rápida, segura y fiable: todo lo que se necesita para afrontar los desafíos, el caos, las sorpresas y la diversión. Windows 10 ofrece antivirus, firewall, anti-ransomware y protección para Internet, todo ello integrado y sin costo adicional. Además, se puede iniciar sesión de forma rápida y segura mediante el reconocimiento facial o la huella dactilar.

NF02. Entorno de Desarrollo DEV-C++

La elección de este entorno de desarrollo se debe a la facilidad con la que se pueden ejecutar archivos en C y C++..

3.3. Arquitectura

3.3.1. Diagrama de Casos de Uso

Diagrama que muestra el flujo del sistema de forma intuitiva, simple y simbólica.

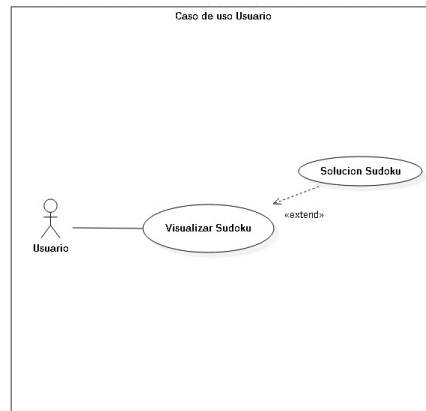


Figura 3.1: Diagrama de Casos de Uso

3.3.2. Diagrama de Secuencia

Diagrama de secuencia que muestra el proceso del programa al ejecutar.

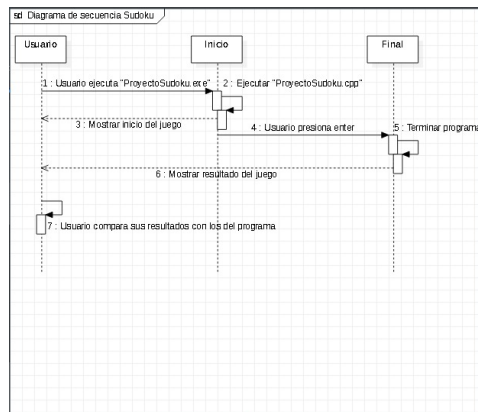


Figura 3.2: Diagrama de Secuencia

3.4. Código Fuente

Para poder disponer del programa, el ejecutable y su código fuente es necesario entrar al enlace siguiente:<https://github.com/Nacho1410/Sudoku.git>

```
// Sudoku por backtracking
#include <bits/stdc++.h>
using namespace std;

// *UNASSIGNED se usa para las celdas vacias
#define UNASSIGNED 0

// N se usa para el tamaño del sudoku
// El tamaño será de NxN
#define N 9
int grid[N][N];

void printGrid(int arr[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ",arr[i][j]);
            if (j==2 || j==5 || j==8) printf("    ");

            if (i==2 || i==5 || i==8) printf("\n\n");
            printf("\n");
        }
    }
}

void crearSudoku()
{
    int i,j,aux;
    srand(time(0)); /*Establece que el origen de los números rand() serán el
```

```

    numero de segundos sucedidos entre el año Nuevo de 1970 hasta hoy: time(0)*/
    grid[0][0] = (rand()%(9))+1; /*Para obtener un entero entre 0 y 8, sacamos el
    resto (con el operador %) de un aleatorio entre 9, dando un numero entre 0 y 8.
    Le sumamos 1 para que sea entre 1 y 9 en numero resultante*/
    do aux = (rand()%(9)) + 1;
        while(aux == grid[0][0]); /*Se busca un numero aleatorio
        que NO sea igual al anterior*/
    grid[0][1] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1]);/*Se busca un
        numero aleatorio que NO sea igual que los anteriores*/
    grid[0][2] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] || aux == grid[0][2]);
    grid[1][0] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] ||
        aux == grid[0][2] || aux == grid[1][0]);
    grid[1][1] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] ||
        aux == grid[0][2] || aux == grid[1][0] || aux == grid[1][1]);
    grid[1][2] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] ||
        aux == grid[0][2] || aux == grid[1][0]
        || aux == grid[1][1] || aux == grid[1][2]);
    grid[2][0] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] ||
        aux == grid[0][2] || aux == grid[1][0] || aux == grid[1][1]
        || aux == grid[1][2] || aux == grid[2][0]);
    grid[2][1] = aux;
    do aux = (rand()%(9))+1;
        while(aux == grid[0][0] || aux == grid[0][1] ||
        aux == grid[0][2] || aux == grid[1][0] || aux == grid[1][1]
        || aux == grid[1][2] || aux == grid[2][0] || aux == grid[2][1]);
    grid[2][2] = aux;
    /*Se rellenan los dos cuadros que quedan de arriba*/
    grid[1][3]=grid[0][0];

```

```
grid[1][4]=grid[0][1];
grid[1][5]=grid[0][2];

grid[2][6]=grid[0][0];
grid[2][7]=grid[0][1];
grid[2][8]=grid[0][2];

grid[2][3]=grid[1][0];
grid[2][4]=grid[1][1];
grid[2][5]=grid[1][2];

grid[0][6]=grid[1][0];
grid[0][7]=grid[1][1];
grid[0][8]=grid[1][2];

grid[0][3]=grid[2][0];
grid[0][4]=grid[2][1];
grid[0][5]=grid[2][2];

grid[1][6]=grid[2][0];
grid[1][7]=grid[2][1];
grid[1][8]=grid[2][2];

/*Se rellenan los cuadros de la izquierda*/
grid[3][1]=grid[0][0];
grid[4][1]=grid[1][0];
grid[5][1]=grid[2][0];

grid[6][2]=grid[0][0];
grid[7][2]=grid[1][0];
grid[8][2]=grid[2][0];

grid[3][2]=grid[0][1];
grid[4][2]=grid[1][1];
grid[5][2]=grid[2][1];

grid[6][0]=grid[0][1];
grid[7][0]=grid[1][1];
grid[8][0]=grid[2][1];
```



```
grid[3][0]=grid[0][2];
grid[4][0]=grid[1][2];
grid[5][0]=grid[2][2];

grid[6][1]=grid[0][2];
grid[7][1]=grid[1][2];
grid[8][1]=grid[2][2];

/* Se rellena el cuadro central y derecho-centro*/
grid[3][3]=grid[5][0];
grid[3][4]=grid[5][1];
grid[3][5]=grid[5][2];

grid[4][6]=grid[5][0];
grid[4][7]=grid[5][1];
grid[4][8]=grid[5][2];

grid[5][3]=grid[4][0];
grid[5][4]=grid[4][1];
grid[5][5]=grid[4][2];

grid[3][6]=grid[4][0];
grid[3][7]=grid[4][1];
grid[3][8]=grid[4][2];

grid[5][6]=grid[3][0];
grid[5][7]=grid[3][1];
grid[5][8]=grid[3][2];

grid[4][3]=grid[3][0];
grid[4][4]=grid[3][1];
grid[4][5]=grid[3][2];

/* Se rellena el cuadro central y derecho-centro*/
grid[6][3]=grid[8][0];
grid[6][4]=grid[8][1];
grid[6][5]=grid[8][2];

grid[7][6]=grid[8][0];
grid[7][7]=grid[8][1];
```

```

grid[7][8]=grid[8][2];

grid[8][3]=grid[7][0];
grid[8][4]=grid[7][1];
grid[8][5]=grid[7][2];

grid[6][6]=grid[7][0];
grid[6][7]=grid[7][1];
grid[6][8]=grid[7][2];

grid[8][6]=grid[6][0];
grid[8][7]=grid[6][1];
grid[8][8]=grid[6][2];

grid[7][3]=grid[6][0];
grid[7][4]=grid[6][1];
grid[7][5]=grid[6][2];


        for (i=0;i<=8;i++) {
        for (j=0;j<=8;j++) {
            if ( (rand()%81)>30 )
            {
                int valor= grid[i][j];
                grid[i][j]=valor;
            }
        else
        {
            grid[i][j]= 0;
        }
        }
        }

        printf(" -- S U D O K U   A   R E S O L V E R -- \n\n");
        printGrid(grid);

}

// esta funcion encuenrtra un espacio en la tabla que no este asignado

```

```
bool FindUnassignedLocation(int grid[N][N],
                           int& row, int& col);

// revisa si se puede asignar un numero a la fila o columna
bool isSafe(int grid[N][N], int row,
            int col, int num);

/* Toma un sudoku parcialmente resuelto e intenta completarlo cumpliendo con los
criterios para la solucion de este */
bool SolveSudoku(int grid[N][N])
{
    int row, col;

    // Si no hay un espacio sin asignar, esta terminado
    if (!FindUnassignedLocation(grid, row, col))
        // success!
        return true;

    // Considera los digitos del 1 al 9
    for (int num = 1; num <= 9; num++)
    {

        // Revisa si se puede rellenar
        if (isSafe(grid, row, col, num))
        {

            // Ingresa un numero tentativo
            grid[row][col] = num;

            // Regresa si hay exito
            if (SolveSudoku(grid))
                return true;

            // Si falla lo deshace y vuelve a intentar
            grid[row][col] = UNASSIGNED;

        }
    }
}
```

```
// Esto acciona el backtracking
return false;
}

/* .Busca un espacio que este sin asignar, si lo encuentra,
los parametros de fila y columna se colocan y devuelve
un TRUE, de lo contrario devuelve un FALSE */

bool FindUnassignedLocation(int grid[N][N],
                           int& row, int& col)
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}

/* Indica si el numero especificado ya fue usado en el mismo renglon */
bool UsedInRow(int grid[N][N], int row, int num)
{
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Indica si el numero indicado ya fue usado en la misma columna */
bool UsedInCol(int grid[N][N], int col, int num)
{
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}

/* Indica si el numero usado ya fue asignado en un espacio de 3x3 */
bool UsedInBox(int grid[N][N], int boxStartRow,
               int boxStartCol, int num)
{

```

```

        for (int row = 0; row < 3; row++)
            for (int col = 0; col < 3; col++)
                if (grid[row + boxStartRow]
                    [col + boxStartCol] ==
                        num)
                    return true;
        return false;
    }

    /* Indica si es seguro asignar el numero en esa fila y columna */
    bool isSafe(int grid[N][N], int row,
               int col, int num)
    {
        /* Revisa si el numero no esta ya colocado en un espacio de 3x3 */
        return !UsedInRow(grid, row, num)
            && !UsedInCol(grid, col, num)
            && !UsedInBox(grid, row - row % 3,
                        col - col % 3, num)
            && grid[row][col] == UNASSIGNED;
    }

    int main()

    {

        crearSudoku();
        cout << "\n ----- \n\n";
        system("pause");
        if (SolveSudoku(grid) == true){
            cout << "\n\n";
            printGrid(grid);}

        else
            cout << "No solution exists";

        return 0;
    }

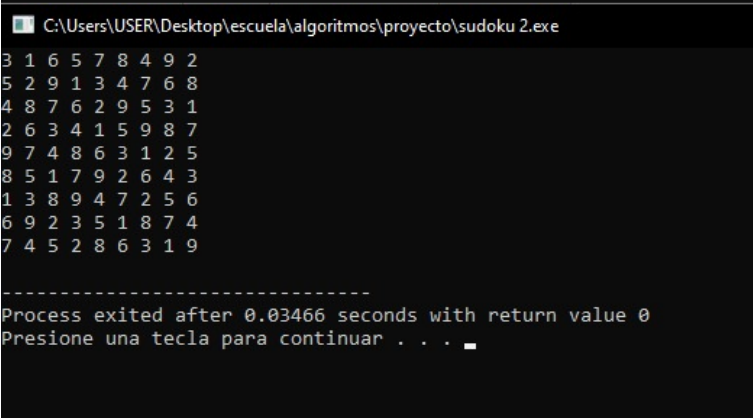
```

Capítulo 4

Historial de Construcción del Programa

Se planea la elaboración de un de un algoritmo que pueda resolver un sudoku tradicional de 9x9. El programa contara con una selección de sudokus predeterminados a elegir por el usuario para su resolución.

4.1. Diseño Preliminar



```
C:\Users\USER\Desktop\escuela\algoritmos\proyecto\sudoku 2.exe
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9

-----
Process exited after 0.03466 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 4.1: Diseño Preliminar

4.1.1. Pruebas a Realizar del Sistema

- Que el sistema compile de forma satisfactoria.
- Que genere un sudoku valido de manera aleatoria.
- Que pueda resolver el sudoku generado anteriormente.

4.1.2. Reporte de Pruebas del Revisor y el Usuario Final

- El programa cumple con su función, despliega en la pantalla un sudoku incompleto, usando ceros (0) como espacios vacíos. Los sudokus siempre son diferentes. Resuelve el sudoku utilizando back tracking, aunque esto no se puede apreciar en el programa, se puede ver en el código. El código esta bien separado, es fácil de entender. Todas las funciones están comentadas en sus partes mas esenciales para entender de manera más simple su funcionamiento.

4.1.3. Cambios Accordados a la Idea Inicial

- Se cambio la lista de sudokus predeterminados por un generador de sudoku aleatorio.
- Ahora se despliega en pantalla el sudoku inicial en su estado antes de ser resuelto.
- Se ajustó el despliegue en pantalla de los números del sudoku.

CAPÍTULO 4. HISTORIAL DE CONSTRUCCIÓN DEL PROGRAMA20

```
C:\Users\ankat\Desktop\ProyectoSudoku.exe
-- S U D O K U   A   R E S O L V E R --

1 7 9   0 0 2   0 0 0
4 6 0   0 0 9   8 0 2
3 0 0   4 0 5   1 0 9

9 1 0   2 0 3   5 0 0
0 4 6   0 1 7   0 8 3
2 8 3   0 4 6   0 0 7

0 0 1   3 2 8   6 5 0
5 0 4   0 9 0   0 0 8
0 2 8   6 0 4   0 9 0

-----
Presione una tecla para continuar . . .
```

Figura 4.2: Diseño Final - Pantalla Principal

```
C:\Users\ankat\Desktop\ProyectoSudoku.exe

0 0 1   3 2 8   6 5 0
6 0 4   0 9 0   0 0 8
0 2 8   6 0 4   0 9 0

-----
Presione una tecla para continuar . . .

1 7 9   8 3 2   4 6 5
4 6 5   1 7 9   8 3 2
8 3 2   4 6 5   1 7 9

9 1 7   2 8 3   5 4 6
5 4 6   9 1 7   2 8 3
2 8 3   5 4 6   9 1 7

7 9 1   3 2 8   6 5 4
6 5 4   7 9 1   3 2 8
3 2 8   6 5 4   7 9 1

-----
Process exited after 37.13 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 4.3: Diseño Final - Pantalla Secundaria

Bibliografía

- [1] <https://opengl-esp.superforo.net/t12-ejemplo-de-c-ns-aleatorios-generador-de-sudokus>
- [2] <https://www.geeksforgeeks.org/sudoku-backtracking-7/?ref=lbp>.