Manual Técnico
Menu del juego:

```
{
    int seleccion;
    do
    {
        std::cout << "Seleccione una opcion\n";
        std::cout << "1. Jugar" << std::endl
                  << " 2. Usuarios" << std::endl
                  << " 3. Regresar\n";
        std::cin >> seleccion;
        switch (seleccion)
        {
        case 1:
            jugar();
            break;
        case 2:
            // Ver Reportes
            generacionUsuario();
            break;
        case 3:
            // FIn juego
            cout << "Regresando al menu principal" << endl;
            break;
        default:
            // presiona algo mas
            cout << "No existe esta opcion" << endl;
            break;
        }
    } while (seleccion != 3);
}
```

Juego

```
void jugar()
{
    // Aqui eestablezco el juego
    auto start = sc.now(); // Comienza a contar el tiempo
    // le solicito al usuario que seleccione con que "usuario" jugara
    cout << "Escriba el numero del usuario con el que jugara" << endl;
    mostrarJugadores();
    int seleccionJugador;
    cin >> seleccionJugador;
    bool jugadorEncontrado = false;

    for (size_t i = 0; i < personasArrayFinal.size(); i++)
    {
        if (seleccionJugador == i)
        {
            cout << "Jugara con el usuario: " << personasArrayFinal[i].getNombre();
            jugadorEncontrado = true;
            break;
        }
    }

    if (jugadorEncontrado)
    {
        string caaBoard[4][4];
        InitializeBoard(caaBoard);
        Randomize(caaBoard);
        bool continuacion = true;
        do
        {
            PrintBoard(caaBoard);
            int pasos = cantidadMovimientos - 1000000;
            cout<<"Nombre del jugador: "<<personasArrayFinal[seleccionJugador].getNombre()<<"\t";
            cout << "Cantidad de Movimientos: " << pasos << endl;
            cout << endl
                 << "w = Arriba - s = Abajo - a = Izquierda - d = Derecha - x = Terminar" << endl;
            char cNextMove;
            cin >> cNextMove;
            if (cNextMove == 'x' || comprobarPuntaje(caaBoard)==32)
            {
                imprimirMatriz(caaBoard);
                cout<<"Su puntaje fue de:" << comprobarPuntaje(caaBoard)<<"\n";
                auto end = sc.now();
                auto time_span = static_cast<chrono::duration<double>>(end - start);
                cout << "Tiempo de Juego: " << time_span.count() << " segundos\n";

                // CREACION Y ASIGNACION DE VARIABLES DE PARTIDAS
```

```
        // CREACION Y ASIGNACION DE VARIABLES DE PARTIDAS
        Partida partidaTempo(personasArrayFinal[seleccionJugador].getNombre(),comprobarPuntaje(caaBoard),
        (cantidadMovimientos - 1000000),time_span.count() );
        partidasArreglo.push_back(partidaTempo);
        continuacion = false;
        break;
    }
    else
    {
        EMove eNextMove = (EMove)cNextMove;
        Move(caaBoard, eNextMove);
        cout << endl;
    }

} while (continuacion == true);

e

 cout << "Error Jugador no existente o sin jugadores";
```

Random Basico

```
void random(string caaBoard[4][4])
{
    using namespace std;
    srand((unsigned int)time(0));
    for (int iIndex = 0; iIndex < 1000000; ++iIndex)
    {
        const int kiNextMove = (rand() % 4);
        switch (kiNextMove)
        {
        case 0:
        {
            Move(caaBoard, keUp);
            break;
        }
        case 1:
        {
            Move(caaBoard, keDown);
            break;
        }
        case 2:
        {
            Move(caaBoard, keLeft);
            break;
        }
        case 3:
        {
            Move(caaBoard, keRight);
            break;
        }
        }
    }
}
```

```
//.....INGRESO MANUAL
// Metodo que determina un ordenamiento del arreglo
void ordenamiento(string **arreglo, int *array)
{

    int contadorOrden =1;
    for (size_t i = 0; i < numeroFilas; i++)
    {
        for (size_t j = 0; j < numeroColumnas; j++)
        {
            if (numeroColumnas-1 == j && numeroFilas-1 == i)
            {
            arreglo[numeroFilas-1][numeroColumnas-1] = "X";

            } else {
                arreglo[i][j] = to_string(array[contadorOrden]);
                contadorOrden++;
            }

        }

    }


}
```

Ganador dinamico

```
// Determina la cantidad de puntos del aleatorio
int comprobarGanadorDinamico(string **arreglo, string **arreglo2)
{
    int cantidadPuntos = 0;
    for (size_t i = 0; i < numeroFilas; i++)
    {
        for (size_t j = 0; j < numeroColumnas; j++)
        {
            if (arreglo[i][j] == arreglo2[i][j])
            {
                cantidadPuntos = cantidadPuntos + 2;
            }
        }
    }
    return cantidadPuntos;
}
```

Aleatorio:

```
// Metodo para genenrar el orden aleatorio en la matriz dinamica
void shufelingMatrix(string **matriz)
{
    int rowindex = rand() % numeroFilas;
    int colindex = rand() % numeroColumnas;
    string auxiliar;
    for (size_t i = 0; i < numeroFilas; i++)
    {

        for (size_t j = 0; j < numeroColumnas; j++)
        {
            auxiliar = matriz[i][j];
            matriz[i][j] = matriz[rowindex][colindex];
            matriz[rowindex][colindex] = auxiliar;
        }
    }
}
```

Ordenamiento Descendente:

```
void ordenarNumeros(int *elemento,int nElementos){
        int aux;

        //Ordenamiento por metodo burbuja con punteros
        for(int i=0;i<nElementos;i++){
            for(int j=0;j<nElementos-1;j++){
                if(*(elemento+j) > *(elemento+j+1)){
                    aux = *(elemento+j);
                    *(elemento+j) = *(elemento+j+1);
                    *(elemento+j+1) = aux;
                }
            }
        }
}
```

AleatorioJuego:
Modo de juego con numeros aleatorios.

```
void ingresoAleatorioNumeros()
{
    cout << "INgrese la cantidad de Filas";
    cin >> numeroFilas;
    cout << "Ingrese la cantidad de columnas";
    cin >> numeroColumnas;
    int cantidadTablero = (numeroFilas * numeroColumnas);

    nElementos = cantidadTablero;
    elementos = new int[nElementos];

    punteroMatriz = new string *[numeroFilas];
    matrixOriginal = new string *[numeroFilas];

    int contador=0;
    for (size_t i = 0; i < numeroFilas; i++)
    {
        punteroMatriz[i] = new string[numeroColumnas];
        matrixOriginal[i] = new string[numeroColumnas];
    }
    // INGRESANDO ELEMENTOS A LA MATRIZ
    for (size_t i = 0; i < numeroFilas; i++)
    {
        for (size_t j = 0; j < numeroColumnas; j++)
        {
            if (j == numeroColumnas - 1 && i == numeroFilas - 1)
            {
                punteroMatriz[i][j] = "X";
                matrixOriginal[i][j] = "X";
                elementos[contador] = 0;

                break;
            }
            else
            {
                int numero;
                // desde 1 al numero final de tablero
                numero = (rand() % cantidadTablero + 1);
                elementos[contador] = numero;
                contador++;

                punteroMatriz[i][j] =to_string(numero);
                matrixOriginal[i][j] = to_string(numero);
            }
        }
    }
```

Juego de forma Manual:
Juego general de formal manual (ingresando numeros por el usuario o con numeros aleatorios).

```
// metodo de juego (manual - Insertando datos -  Aleatorio)
void jugarManual(int seleccion)
{
    cantidadMovimientos = 0;
    // Aqui eestablezco el juego
    auto start = sc.now(); // Comienza a contar el tiempo
    // le solicito al usuario que seleccione con que "usuario" jugara
    cout << "Escriba el numero del usuario con el que jugara" << endl;
    mostrarJugadores();
    int seleccionJugador;
    cin >> seleccionJugador;
    bool jugadorEncontrado = false;

    for (size_t i = 0; i < personasArrayFinal.size(); i++)
    {
        if (seleccionJugador == i)
        {
            cout << "Jugara con el usuario: " << personasArrayFinal[i].getNombre();
            jugadorEncontrado = true;
            break;
        }
    }

    if (jugadorEncontrado)
    {

        if (seleccion == 2)
        {
            ingresoManual();
        }
        else if (seleccion == 3)
        {
            ingresoAleatorioNumeros();
        }
        cout << "La matriz generada es:\n";
        imprimir(punteroMatriz);
        shufelingMatrix(punteroMatriz);
        cout<<"Seleccione que modo quiere jugar: \n"<<
                "1. Orden descendente\n"<<
                "2. Orden estipulado por el programa/Usuario";
        int seleccionModo;
        cin>>seleccionModo;
        if (seleccionModo ==1)
        {
            /**Continua*/
        } else if (seleccionModo ==2) {
        cout << "La matriz generada es:\n";
        ordenarNumeros(elementos, nElementos);
        ordenamiento(matrixOriginal, elementos);
        imprimir(matrixOriginal);
```

```cpp
bool continuacion = true;
do
{
    imprimir(punteroMatriz);
    // imprimir(punteroMatriz, numeroFilas, numeroColumnas);
    int pasos = cantidadMovimientos;

    cout << "Nombre del jugador: " << personasArrayFinal[seleccionJugador].getNombre() << "\t
    cout << "Cantidad de Movimientos: " << pasos << endl;
    cout << endl
        << "w = Arriba - s = Abajo - a = Izquierda - d = Derecha - x = Terminar" << endl;
    char cNextMove;
    cin >> cNextMove;

    if (cNextMove == 'x' || comprobarGanadorDinamico(punteroMatriz, matrixOriginal) == 2 * (
    {
        imprimir(punteroMatriz);
        cout << "Su puntaje fue de:" << comprobarGanadorDinamico(punteroMatriz, matrixOrigin
            << "\n";
        // Generacion de Tiempo en Partida
        auto end = sc.now();
        auto time_span = static_cast<chrono::duration<double>>(end - start);
        cout << "Tiempo de Juego: " << time_span.count() << " segundos\n";

        // CREACION Y ASIGNACION DE VARIABLES DE PARTIDAS

        Partida partidaTempo(personasArrayFinal[seleccionJugador].getNombre(), comprobarGana

        // agrega la partida
        partidasArreglo.push_back(partidaTempo);
        // eLIMINACION DE PUNTEROS
        for (size_t i = 0; i < numeroFilas; i++)
        {
            delete[] punteroMatriz[i];
            delete[] matrixOriginal[i];
        }
        // eliminacion de puntero
        delete[] punteroMatriz;
        delete[] matrixOriginal;
        delete[] elementos;

        continuacion = false;
        break;
    }
    else
    {
        EMove eNextMove = (EMove)cNextMove;
        moverEspacios(punteroMatriz, eNextMove);
        cout << endl;
    }
```