



UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE SISTEMAS ELECTRÓNICOS

ACADEMIA DE REDES Y PROGRAMACIÓN DE SISTEMAS

INGENIERÍA EN SISTEMAS COMPUTACIONALES

TERCER SEMESTRE. GRUPO A

MATERIAS: PROGRAMACIÓN II / ESTRUCTURAS DE DATOS

DOCENTE: EDUARDO SERNA PÉREZ

INTEGRANTES DE EQUIPO:

269686. ERIK ALEJANDRO GÓMEZ MARTÍNEZ

211694. ISRAEL ALEJANDRO MORA GONZÁLEZ

212276. ÁNGEL GABRIEL GALINDO LÓPEZ

269770. JOSÉ EMMANUEL RODRÍGUEZ LÓPEZ

FECHAS DE ENTREGA: 21-23 DE DICIEMBRE DE 2020

DOCUMENTACIÓN: N-PUZZLE Y ANÁLISIS DE ALGORITMOS

RESUMEN DESCRIPTIVO

FORTALEZAS

El proyecto fue programado con el uso del IDE Visual Studio 2019; asimismo, las funcionalidades gráficas se incorporaron gracias a la librería Allegro 5.2.6, que está diseñada para la creación de videojuegos, por lo cual incluye diversas funciones del manejo de mouse y teclado, flexibilizando, de esta manera, un sistema de validaciones y mecánicas sólidas. El uso de múltiples estructuras de la STL (Standard Template Library, o Librería de Plantillas Estándar), como «vector» y «list», además de «string», permitió implementar algoritmos (calcular coordenadas, definir tableros, controlar texto, gestionar archivos, etcétera) de mayor agilidad y optimización.

Por otro lado, el juego permite la cancelación de todo tipo de partidas, e incluso el retorno entre menús de opciones (modos de operación, dificultades o nombres), lo que lo vuelve más flexible ante situaciones específicas (datos no intencionados, necesidad de cerrar el programa inmediatamente, aburrimiento, etcétera).

DEBILIDADES

La instalación del juego (N-Puzzle) representa cierto problema, ya que la librería Allegro 5.2.6 es muy difícil de implementar en diversos tipos de editores (Dev-C++, CodeBlocks, entre otros); así que puede ser necesario efectuar el proceso de compilación, en el IDE Visual Studio 2019, pues, al menos en nuestra experiencia, ha sido el único software que incluye, de forma nativa, todos los archivos necesarios, aunque demanda muchos recursos del CPU y memoria RAM, por lo que esta metodología no se podría llevar a cabo en equipos de muy bajas especificaciones.

Debido a la forma en la que el código fuente fue distribuido a lo largo de archivos .hpp y .cpp, así como su enorme extensión y complejidad, resulta ciertamente difícil de leer e interpretar (incluso es preferible consultar algún método, a través de su fichero, con la declaración).

Es probable que algunos algoritmos, como el sistema para detectar eventos del mouse, esquemas de impresión bajo cierta unidad de tiempo, validación de entradas, mecánicas núcleo del juego o Branch and Bound, presenten un comportamiento «gigantesco» (requerir millones de operaciones, por ejemplo), lo cual compromete a la funcionalidad, rendimiento y comodidad del programa mismo. Por otro lado, la resolución automática solo puede realizarse con puzzles de 3x3, ya que las medidas mayores podrían colapsar la estabilidad del equipo utilizado (gastando dos o más Gb de memoria RAM); además, el protocolo para sugerir movimientos es ciertamente burdo (no funciona a la perfección).

TEMAS INVESTIGADOS

CUESTIONES ACERCA DE LOS ALGORITMOS

A) ¿Cómo se mide el tiempo de ejecución de un algoritmo?

La eficiencia de los algoritmos se mide por el número de comparaciones e intercambios que tienen que hacer; por ejemplo, se toma n como el número de elementos que tiene el arreglo a ordenar, y se dice que un algoritmo realiza $O(n^2)$ comparaciones cuando procesa 'n' veces los 'n' elementos: $n * n = n^2$.

B) ¿De qué factores depende el tiempo de ejecución de un algoritmo? Describe cada uno.

- *Complejidad:* consiste en la cantidad de recursos, tiempo y espacio de memoria que un algoritmo requiere para ser procesado.
- *Longitud del problema:* es una magnitud que influye directamente en la complejidad de un algoritmo, pues determina qué cifra de procesos deben realizarse para resolver un problema; por ejemplo: la cantidad de elementos que componen a un vector a ordenar, u obtener el factorial de un valor 'n'.
- *Casos variados:* según los datos que el algoritmo deba manejar, la demanda temporal, de recursos y espacio en memoria puede incrementar o disminuir; esto es conocido como la naturaleza de la información de entrada.
- *Hardware y software:* cada arquitectura computacional está compuesta principalmente por CPU's y una memoria RAM, de distinta capacidad y rendimiento, lo cual influye, de forma directa, en la velocidad del equipo; un mismo programa corre distinto en dos computadores independientes. De igual manera, el sistema operativo, gracias a sus políticas de planificación, gestiona los recursos disponibles, lo que genera variaciones en el desempeño del dispositivo.

C) ¿Qué es una medida asintótica?

Es una forma de estimar el tiempo de ejecución de un algoritmo, así como su comportamiento (variabilidad de la eficiencia), sin importar el hardware, o software basal. Normalmente, se representa mediante la notación asintótica, que varía con el tamaño de entrada; algunas escalas serían las siguientes: O (cota superior), Ω (cota inferior) y θ (cota ajustada).

D) ¿Qué es la cota superior de un algoritmo, y con qué símbolo se representa?

Es la que se utiliza para describir el comportamiento de cierto algoritmo, cuya función equivalente tiende a un valor (sea infinito o no), gracias a lo cual puede clasificarse por medio del incremento de los requisitos de ejecución (tiempo o espacio), según la variación del tamaño de entrada; genera un acotado superior, por lo que analiza los ‘peores casos’. Se representa con el símbolo O .

E) ¿Qué es la cota inferior de un algoritmo, y con qué símbolo se representa?

Se utiliza para acotar inferiormente una función, correspondiente a cierto algoritmo, que tiende a un valor específico (incluso, a infinito); asimismo, permite analizar su comportamiento, según las variaciones de los tiempos de ejecución, que están basadas en el incremento del dato de entrada. Se representa con el símbolo Ω y equivale a los ‘mejores casos’.

F) ¿Qué es una notación O grande?

Es la que se utiliza para clasificar a los diversos algoritmos, según su tasa de crecimiento, a raíz de la variabilidad del dato de entrada. Asimismo, representa la cota superior, por lo que corresponde a los ‘peores casos’.

G) ¿A qué se refiere analizar un algoritmo en el mejor, peor y caso promedio?

- *Mejor caso:* los requisitos de tiempo, espacio y procesamiento son los mínimos que el algoritmo permite, por lo que podría decirse que se genera una ejecución óptima.
- *Caso promedio:* el rendimiento y eficacia del algoritmo se determinan de manera aleatoria, por lo que corresponden a un término medio.

- *Peor caso*: el algoritmo requiere un índice máximo de tiempo, espacio en memoria y procesamiento para ejecutarse por completo, lo cual genera un esquema de optimización nula.

GENERACIÓN DE PUZZLES RESOLVIBLES

A) Cálculo de las inversiones: las inversiones, de cualquier puzzle, se calculan con base en la cantidad de pares numéricos con orden inverso (es decir, primer valor $>$ segundo valor) en comparación al tablero de referencia (meta predeterminada), el cual es el siguiente:

1	2	3
4	5	6
7	8	

Cabe mencionar que el mismo orden ascendente se determina con tableros de mayores dimensiones (4x4, 5x5, entre otras).

B) Paridad de un puzzle: si la paridad es par, entonces hay solución; en caso contrario, no existe alguna (si el tablero final no es el predeterminado, entonces el sistema es resoluble si tiene la misma paridad que la del punto inicial). El cálculo de este atributo depende del número de columnas del puzzle:

- *Anchura impar*: la paridad de la cantidad total de inversiones es directamente proporcional a la del puzzle. Esto aplica para tableros de 3x3, 5x5, etcétera.
- *Anchura par*: si el espacio se encuentra en una fila par, entonces la paridad del puzzle será inversamente proporcional a la del total de inversiones; en caso contrario, directamente proporcional.

PROGRAMACIÓN DEL PROYECTO

N-PUZZLE

Esta parte del proyecto fue estructurada principalmente con las siguientes clases:

- **‘Juego’:** permite controlar todo el funcionamiento del programa, ya sea la gestión de los menús, almacenamiento de archivos, manejo de la información del usuario, mecánicas, etcétera. Sus métodos principales son los siguientes:
 - **pantalla_titulo:** se genera la pantalla de título, correspondiente al juego.
 - **pantalla_menu:** se genera el menú principal (selección para comenzar juego, consultar récords o salir).
 - **pantalla_menu _Dificultad:** se genera la pantalla para seleccionar la dificultad del próximo juego, ya sea este manual o automático.
 - **pantalla_menu_Modo:** se imprime la interfaz para la selección del modo de juego.
 - **pantalla_Records:** se imprime la tabla de récords.
 - **pantalla_Eliminacion:** se imprime la pantalla que indica la eliminación del registro de récords.
 - **pantalla_finJuego:** se imprime la pantalla que indica la finalización de una partida manual, incluyendo el nombre y puntaje del jugador.
 - **pantalla_fin_simulacion:** se imprime la pantalla que indica la finalización de una simulación automática; según la validez de los puzzles, este método muestra un mensaje u otro.
 - **modo_manual:** se gestionan todos los mecanismos para llevar a cabo una partida en modo manual.
 - **capturar_puzzles:** se genera el sistema para obtener los puzzles, tanto de inicio como meta, en el modo automático.

- **‘Jugador’**: contiene la información del jugador, como su nombre, puntaje, última fecha de juego. Su método más importante es el siguiente:
 - **capturaNombre**: se genera el sistema para capturar el nombre del jugador actual.
- **‘Nodo’**: permite generar el árbol para solucionar tableros, lo cual utiliza el algoritmo Branch and Bound. Su método más importante es el siguiente:
 - **swap**: se intercambian dos elementos del tablero correspondiente.
- **‘Puzzle’**: almacena, gestiona y procesa información acerca de un tablero particular. Sus métodos principales son los siguientes:
 - **calculaCosto**: se calcula el costo que representa resolver un tablero inicial a una meta.
 - **imprimeRuta**: se imprime la ruta óptima para resolver un tablero.
 - **generaPuzzle**: se genera un puzzle aleatorio y resoluble (esto último respecto a una meta predeterminada).
 - **resuelve**: se resuelve el puzzle, por medio del algoritmo Branch and Bound.
 - **inversiones**: se calcula la cantidad de inversiones que tiene el puzzle.
 - **resoluble_manual (polimorfismo)**: se verifica si el puzzle es resoluble o no (el código es diferente para tableros de 3x3, 4x4 y 5x5).

HERENCIA Y POLIMORFISMO

```
//-----Clase 'Puzzle_facil', hija de 'Puzzle'-----//.
class Puzzle_facil : public Puzzle {
public:
    //---Constructor y destructor---//.
    Puzzle_facil(ALLEGRO_FONT* letra, ALLEGRO_DISPLAY* pantalla, int valor) : Puzzle(letra, pantalla, valor) {}
    ~Puzzle_facil() { Puzzle::~~Puzzle(); } //Destructor.

    //---Métodos---//.
    bool resoluble_manual(); //Se verifica si el 8-puzzle puede resolverse con la meta predeterminada.
};
```

```
//-----Clase 'Puzzle_medio', hija de 'Puzzle'-----//.
class Puzzle_medio : public Puzzle {
private:
    int posicion_x_cero(); //Se retorna la posición horizontal del espacio.
public:
    //---Constructor y destructor---//.
    Puzzle_medio(ALLEGRO_FONT* letra, ALLEGRO_DISPLAY* pantalla, int valor) : Puzzle(letra, pantalla, valor) {}
    ~Puzzle_medio() { Puzzle::~~Puzzle(); } //Destructor.

    //---Métodos---//.
    bool resoluble_manual(); //Se verifica si el 15-puzzle puede resolverse con la meta predeterminada.
};
```

```
//-----Clase 'Puzzle_dificil', hija de 'Puzzle'-----//.
class Puzzle_dificil : public Puzzle {
public:
    //---Constructor y destructor---//.
    Puzzle_dificil(ALLEGRO_FONT* letra, ALLEGRO_DISPLAY* pantalla, int valor) : Puzzle(letra, pantalla, valor) {}
    ~Puzzle_dificil() { Puzzle::~~Puzzle(); } //Destructor.

    //---Métodos---//.
    bool resolvable_manual(); //Se verifica si el 24-puzzle puede resolverse con la meta predeterminada.
};
```

ESTRUCTURAS DE DATOS

VECTOR

```
//Se gestiona el modo manual del juego.
void Juego::modo_manual(Jugador &jugador_actual) {
    ALLEGRO_EVENT_QUEUE* fila_evento = al_create_event_queue();
    ALLEGRO_TIMER* temporizador = al_create_timer(1);
    ALLEGRO_SAMPLE* apuntado = al_load_sample("Sounds/smw_map_move_to_spot.wav");
    ALLEGRO_SAMPLE* deslizar = al_load_sample("Sounds/smw_fireball.wav");
    ALLEGRO_SAMPLE* avance = al_load_sample("Sounds/smw_message_block.wav");
    ALLEGRO_SAMPLE* musica_juego = al_load_sample("Sounds/Casino Kid (NES) Music Casino Theme.mp3");
    ALLEGRO_SAMPLE_ID id1;
    al_reserve_samples(3);
    Puzzle* tablero_final, *tablero_inicial;
```

```
do {
    tablero_inicial->generaPuzzle();
} while (*tablero_inicial == *tablero_final || !tablero_inicial->resolvable_manual());

vector<Par_coordenadas>* parametros = new vector<Par_coordenadas>;
*parametros = this->fichas_a_mover(tablero_inicial);

bool continuar = false, reanudar, sonido = false;
int auxiliar = 2, veces_ayuda = 0;

al_play_sample(musica_juego, 0.5, 0, 1, ALLEGRO_PLAYMODE_LOOP, &id1);
```

```
case ALLEGRO_EVENT_MOUSE_BUTTON_DOWN:
    if (auxiliar >= 1 && auxiliar <= 4) {
        tablero_inicial->swap(auxiliar);
        this->imprimir_interfaz_modos_manual(jugador_actual, tablero_inicial, tablero_final);

        delete parametros;
        parametros = new vector<Par_coordenadas>;
        *parametros = this->fichas_a_mover(tablero_inicial);

        al_play_sample(deslizar, 2.0, 0, 1.0, ALLEGRO_PLAYMODE_ONCE, 0);
    }
    else if (auxiliar == 6) {
        al_play_sample(avance, 5.0, 0, 1.0, ALLEGRO_PLAYMODE_ONCE, 0);
    }
```



```
//Se genera el menú para la solicitar los puzzles.
void Juego::capturar_puzzles(Puzzle*& inicio, Puzzle*& meta, ALLEGRO_SAMPLE_ID id) {
    ALLEGRO_SAMPLE* apuntado = al_load_sample("Sounds/smw_map_move_to_spot.wav");
    ALLEGRO_SAMPLE* deslizar = al_load_sample("Sounds/smw_fireball.wav");
    ALLEGRO_SAMPLE* avance = al_load_sample("Sounds/smw_message_block.wav");
    bool cancelar = false; Puzzle* manejador;
    vector<Coordenadas> *lista_espacios = new vector<Coordenadas>;

    if (this->dificultad == 3) {
        inicio = new Puzzle_facil(this->letra, this->pantalla, 3);
        meta = new Puzzle_facil(this->letra, this->pantalla, 3);
    }
}
```

```
        delete lista_espacios;
        lista_espacios = new vector<Coordenadas>;
        continuar = true;
    }
    else if (auxiliar == 26) {
        al_play_sample(avance, 5.0, 0, 1.0, ALLEGRO_PLAYMODE_ONCE, 0);
        inicio[0][0][0] = "d";
    }
}
```

```
//Se determina en cuál ícono se ha posicionado el cursor.
int Juego::posicionado_modos_manual(vector<Par_coordenadas> parametros) {
    for (auto it : parametros) {
        if (this->x >= it.x_inicial && this->x <= it.x_final && this->y >= it.y_inicial && this->y <= it.y_final) {
            return it.posicion;
        }
    }
    if (this->x >= 470 && this->x <= 750 && this->y >= 460 && this->y <= 520) return 6;
    else return 7;
}
```

```
vector<ALLEGRO_BITMAP*> fichas = new vector<ALLEGRO_BITMAP*>;
for (int i = 1; i < 25; i++) {
    string aux = "Sources/Fichas/F";
    ALLEGRO_BITMAP* elemento = al_load_bitmap((aux + to_string(i) + ".png").c_str());
    fichas->push_back(elemento);
}
}
```

```
//-----Clase 'Puzzle'-----//.
class Puzzle {
protected:
    //---Atributos---//.
    int tipo_puzzle;
    vector<vector<string>>* puzzle;
    int x, y;
    int contador;
    ALLEGRO_FONT* letra;
    ALLEGRO_DISPLAY* pantalla;
}
```

LIST

```
//Se actualiza el contenido del registro.
void Juego::actualizar_archivo(Jugador jugador_actual) {
    fstream archivo("registro.dat", ios::binary | ios::in);
    if (!archivo) {
        archivo.close();
        archivo.open("registro.dat", ios::binary | ios::out);
        archivo.close();
        fstream archivo("registro.dat", ios::binary | ios::in);
    }
    Jugador* aux1 = new Jugador;
    Datos_Guardar* aux2 = new Datos_Guardar;
    list<Jugador>* ordenamiento = new list<Jugador>;
    bool encontrado = false;
```

PRIORITY QUEUE

```
//Si tiene solución, se resuelve el puzzle por medio del algoritmo branch and bound.
void Puzzle::resuelve(int x,int y, vector<vector<string>> &puzzle_final) {
    // Para los movimientos de las fichas.
    // Abajo, Izq, Arriba, Der.
    int row[] = { 1, 0, -1, 0 };
    int col[] = { 0, -1, 0, 1 };

    // Creamos una cola de prioridad para almacenar los nodos vivos del arbol de busqueda.
    priority_queue<Nodo*, vector<Nodo*>, comp> nodosVivos;

    // Creamos un nuevo nodo y calculamos su costo.
    Nodo* raiz = new Nodo(*(this->puzzle), x, y, x, y, 0, NULL);
```

ALGORITMOS DE ORDENAMIENTO

Esta parte del proyecto fue estructurada principalmente con las siguientes clases:

- **‘Sistema’:** controla el funcionamiento de todo el programa, gestionando la generación de vectores, métodos de ordenamiento, cálculo de tiempos, movimientos, comparaciones, entre otros. Sus métodos principales son los siguientes:
 - **pantalla_Principal:** se genera la pantalla inicial, en la cual se solicita el arreglo a procesar.

- **pantallaVectores:** se muestran los vectores generados (mejor, peor y caso promedio).
 - **pantalla_StatusTiempo:** se imprime la lista de tiempos para cada método de ordenamiento, así como los respectivos casos para los algoritmos
 - **pantalla_StatusCalculos:** se imprime la tabla de comparaciones y movimientos para cada método de ordenamiento, según los respectivos casos algorítmicos
 - **pantalla_EjecutaMetodos:** se ejecutan todos los métodos de ordenamiento.
- **‘Timer’:** gestiona el cálculo de los tiempos de ejecución, para cada método de ordenamiento. Contiene métodos para iniciar, finalizar y obtener el cronómetro.
 - **‘Vector’:** gestiona los tres vectores generados; asimismo, permite realizar los distintos métodos de ordenamiento. Sus métodos principales son los siguientes:
 - **insertionSort:** el vector correspondiente se ordena por el método de inserción.
 - **selectionSort:** el vector correspondiente se ordena por el método de selección.
 - **bubbleSort:** el vector correspondiente se ordena por el método de burbuja.
 - **mergeSort:** el vector correspondiente se ordena por el método por mezcla.
 - **shellSort:** el vector correspondiente se ordena por el método por shell.
 - **quickSort:** el vector correspondiente se ordena por el método rápido.
 - **heapSort:** el vector correspondiente se ordena por el método de montículos.

INSTALACIÓN DEL PROYECTO

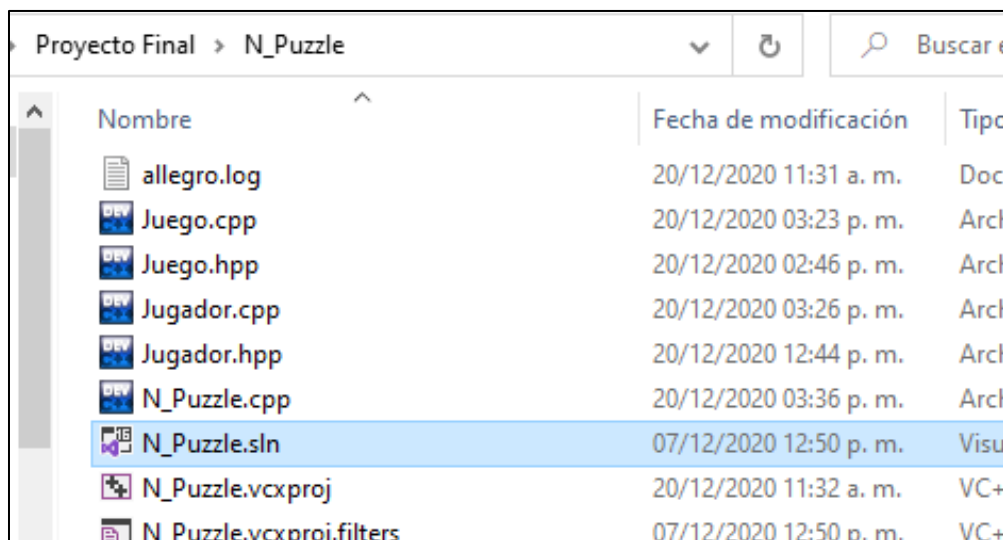
N-PUZZLE

Si no se cuenta con la librería Allegro 5.2.6, entonces se deben realizar los siguientes pasos; en caso contrario, solo resulta necesario asegurarse de que el lenguaje se encuentre en su versión C++11:

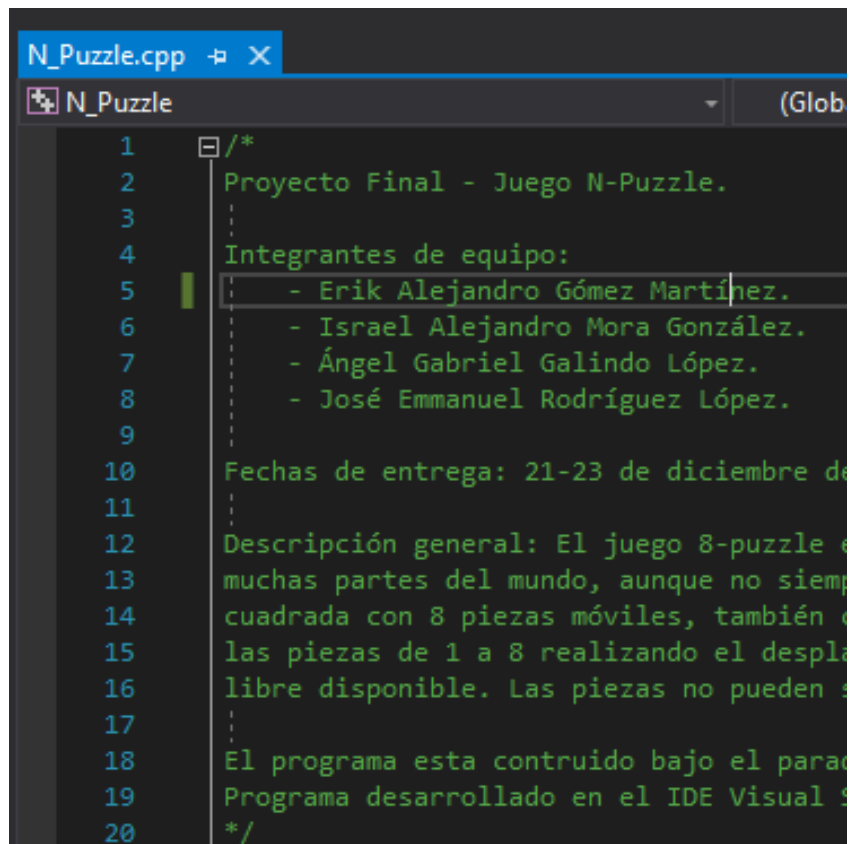
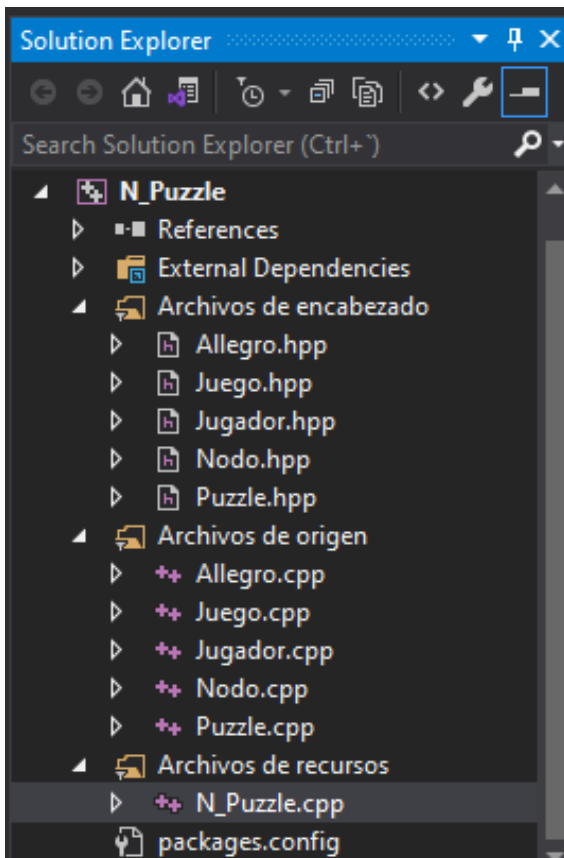
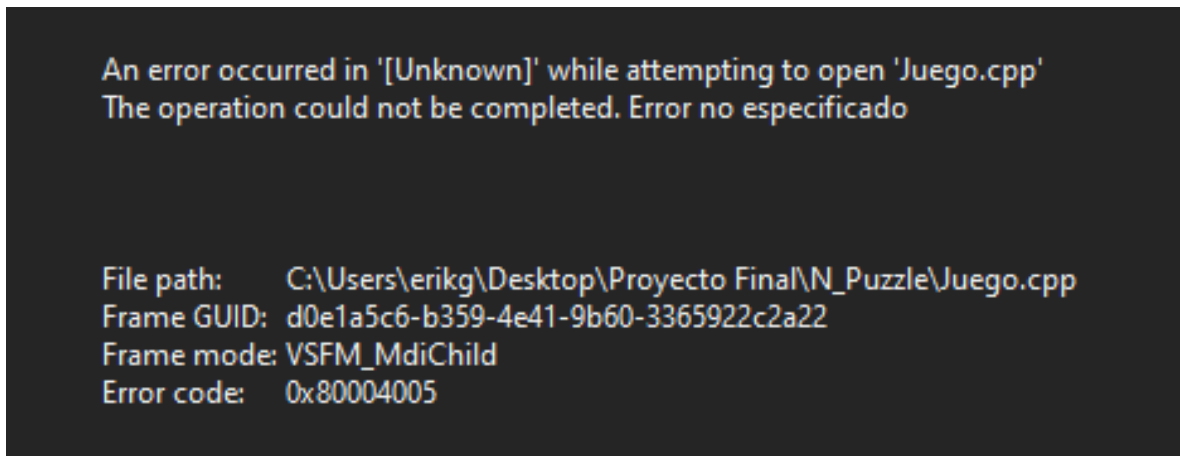
1.- Instalar el IDE Visual Studio 2019 (edición para la comunidad). El instalador puede obtenerse desde el siguiente enlace: <https://visualstudio.microsoft.com/es/vs/>. Cabe mencionar que se deben incluir las dependencias para trabajar con C++ (esto se puede realizar durante el proceso de instalación).



2.- Descomprimir el archivo .rar, y después abrir el archivo «N_Puzzle.sln».



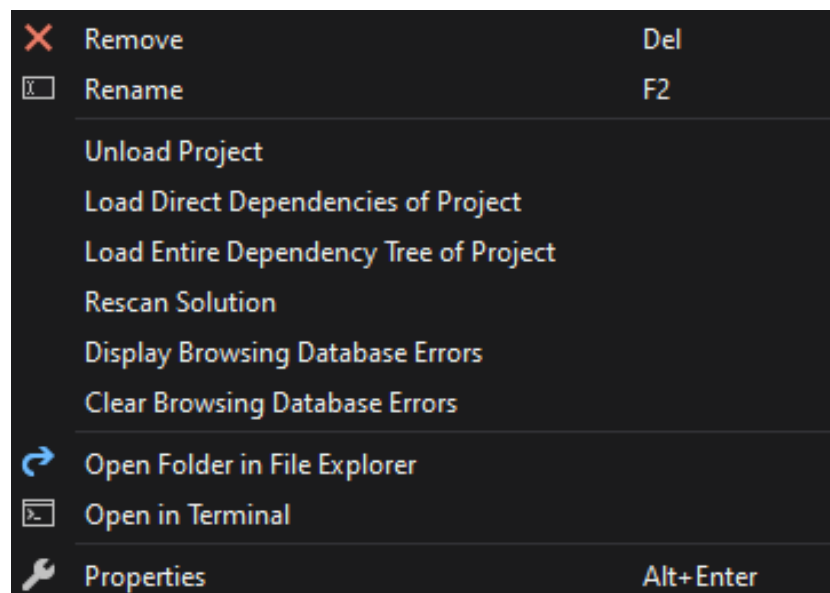
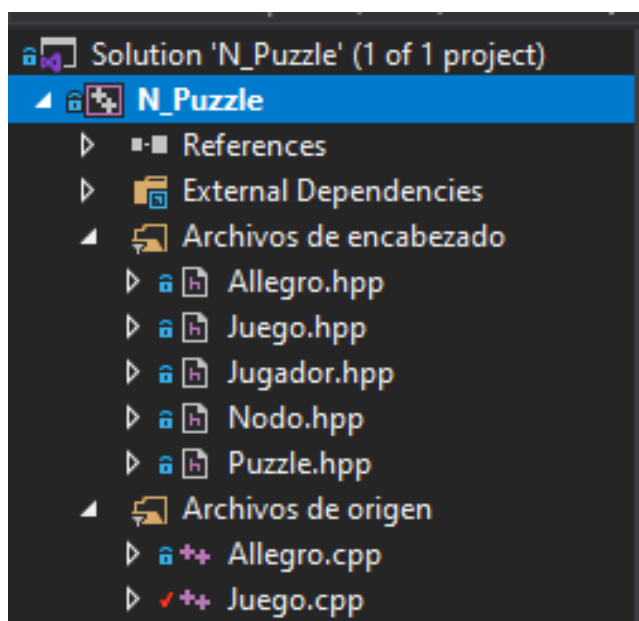
3.- Una vez abierto el proyecto, se indicará que cierto archivo no pudo ser abierto, lo cual se debe al cambio del sistema de directorios; de todas formas, esto no representa ningún problema serio, así que se puede solucionar abriendo cualquier fichero de la pestaña de «Soluciones».

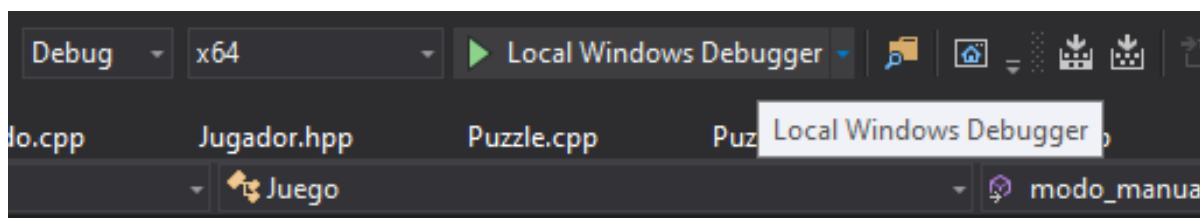
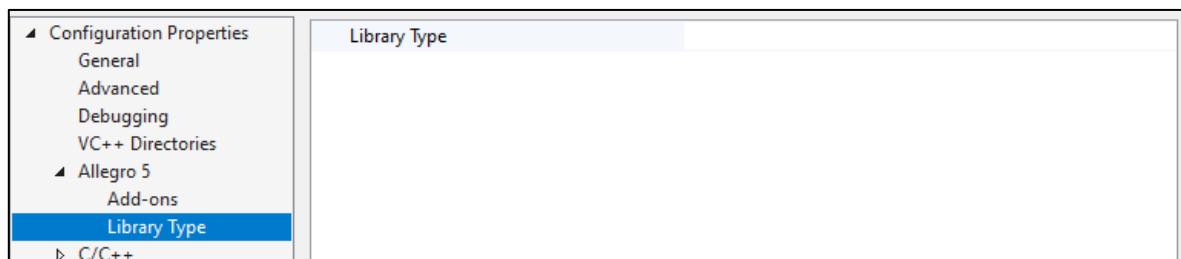
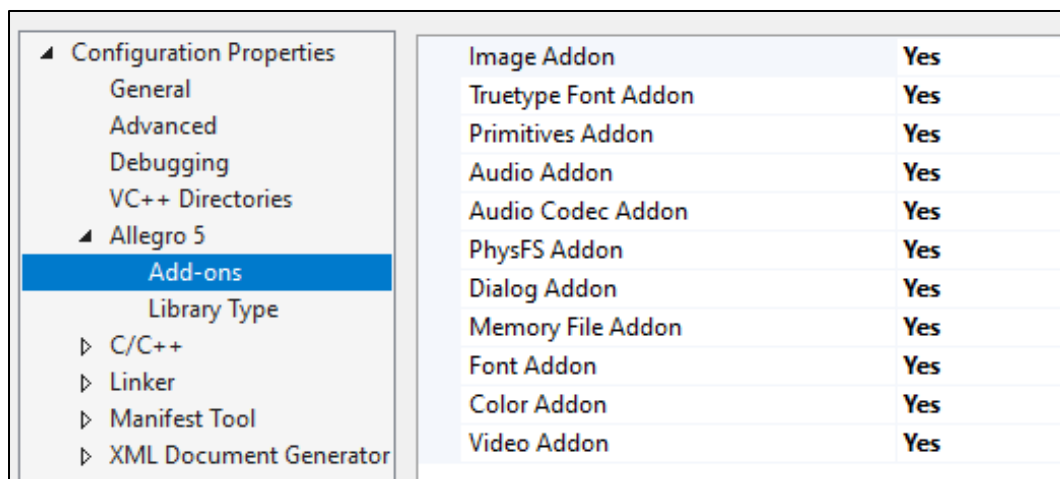


4.- Cabe mencionar que los archivos ya incluyen los paquetes de Allegro, así como sus respectivas configuraciones, así que solo es necesario compilar el proyecto (utilizando el esquema con depuración).

Proyecto Final > N_Puzzle			
Buscar en N_Puzzle			
Nombre	Fecha de modificación	Tipo	Tamaño
.vs	13/12/2020 11:37 p. m.	Carpeta de archivos	
.vscode	13/12/2020 11:37 p. m.	Carpeta de archivos	
Bitmaps	12/12/2020 04:08 p. m.	Carpeta de archivos	
Fonts	13/12/2020 11:38 p. m.	Carpeta de archivos	
packages	13/12/2020 11:38 p. m.	Carpeta de archivos	
Sounds	13/12/2020 11:38 p. m.	Carpeta de archivos	
Sources	13/12/2020 01:53 p. m.	Carpeta de archivos	

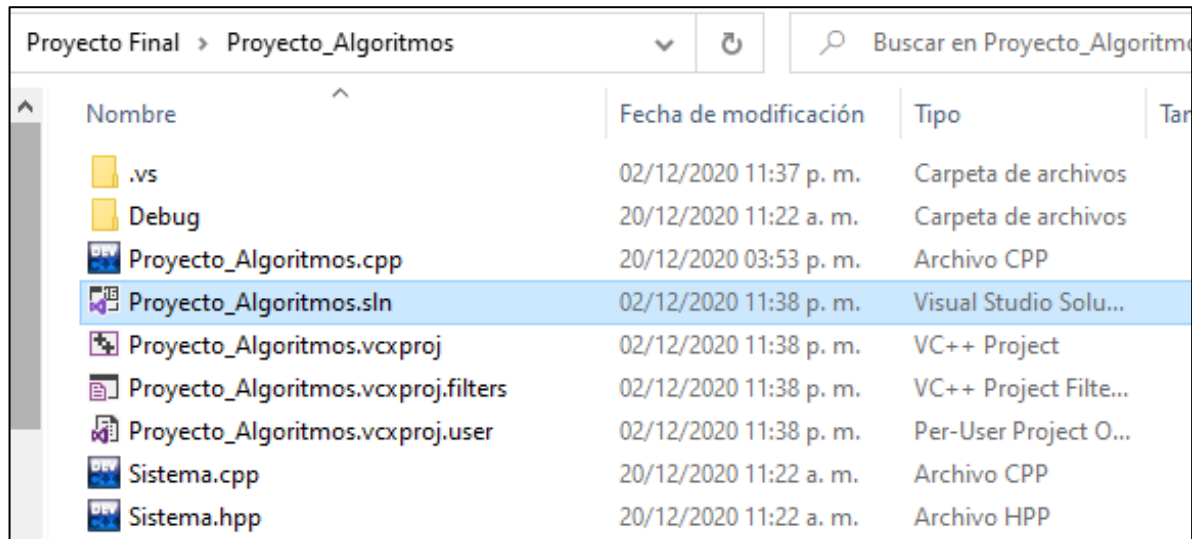
Proyecto Final > N_Puzzle > packages			
Buscar en packages			
Nombre	Fecha de modificación	Tipo	Tamaño
Allegro.5.2.6	13/12/2020 11:38 p. m.	Carpeta de archivos	
AllegroDeps.1.11.0	13/12/2020 11:38 p. m.	Carpeta de archivos	





COMPARACIÓN DE ALGORITMOS

1.- Tras haber instalado el IDE Visual Studio 2019, abrir el archivo «Proyecto_Algoritmos.sln».

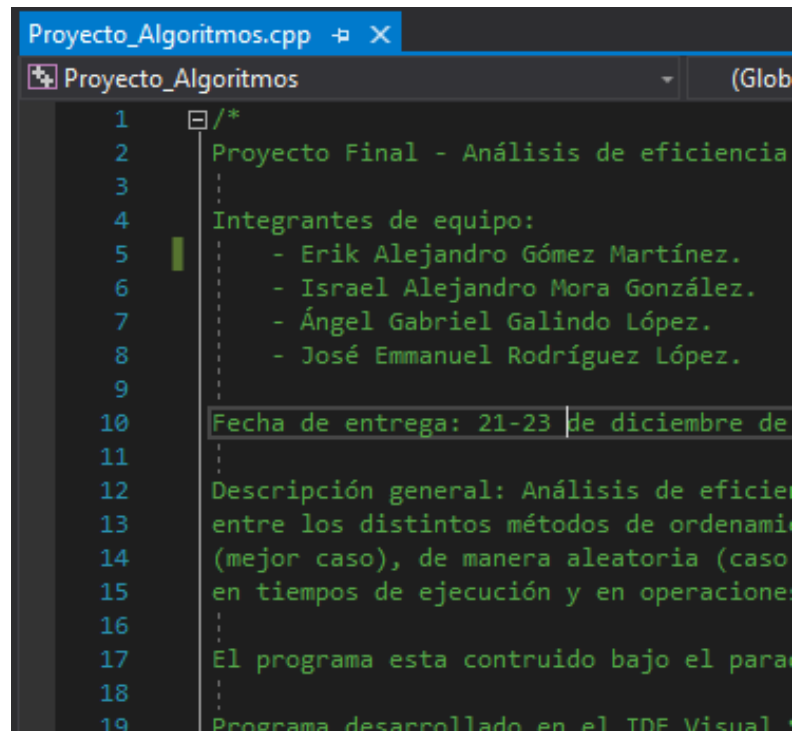
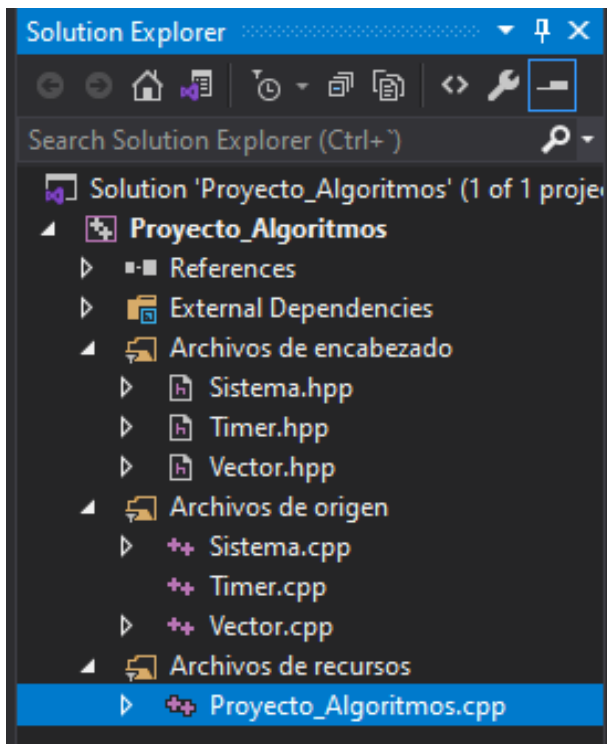


Proyecto Final > Proyecto_Algoritmos		Buscar en Proyecto_Algoritmos
Nombre	Fecha de modificación	Tipo
.vs	02/12/2020 11:37 p. m.	Carpeta de archivos
Debug	20/12/2020 11:22 a. m.	Carpeta de archivos
Proyecto_Algoritmos.cpp	20/12/2020 03:53 p. m.	Archivo CPP
Proyecto_Algoritmos.sln	02/12/2020 11:38 p. m.	Visual Studio Solu...
Proyecto_Algoritmos.vcxproj	02/12/2020 11:38 p. m.	VC++ Project
Proyecto_Algoritmos.vcxproj.filters	02/12/2020 11:38 p. m.	VC++ Project Filte...
Proyecto_Algoritmos.vcxproj.user	02/12/2020 11:38 p. m.	Per-User Project O...
Sistema.cpp	20/12/2020 11:22 a. m.	Archivo CPP
Sistema.hpp	20/12/2020 11:22 a. m.	Archivo HPP

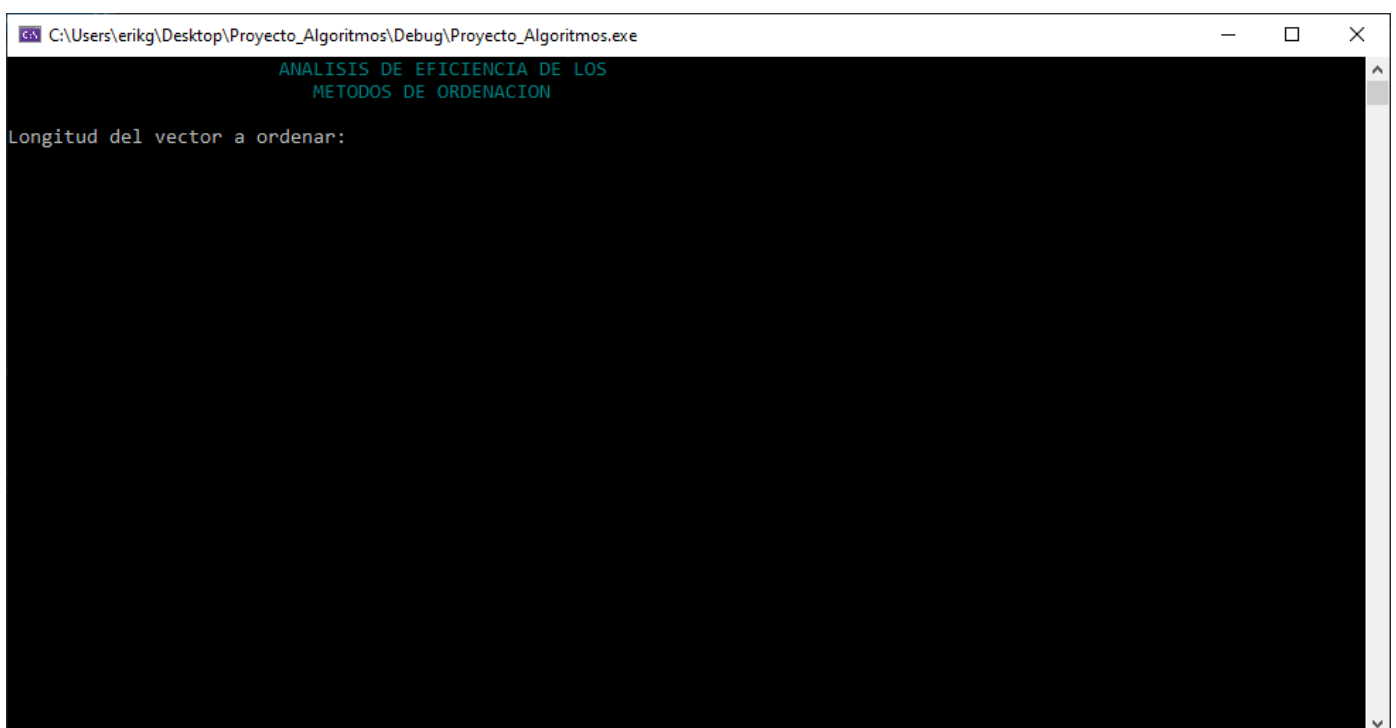
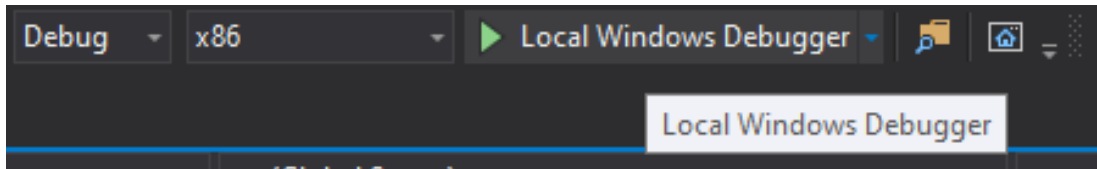
2.- Al igual que con el N-Puzzle, una vez abierto el proyecto, se indicará que cierto archivo no pudo ser abierto, lo cual se debe al cambio del sistema de directorios; de todas formas, esto no representa ningún problema serio, así que se puede solucionar abriendo cualquier fichero de la pestaña de «Soluciones».

An error occurred in '[Unknown]' while attempting to open 'Proyecto_Algoritmos.cpp'
The operation could not be completed. Error no especificado

File path: C:\Users\erikg\Desktop\Proyecto Final\Proyecto_Algoritmos\Proyecto_Algoritmos.cpp
Frame GUID: d0e1a5c6-b359-4e41-9b60-3365922c2a22
Frame mode: VSFM_MdiChild
Error code: 0x80004005



3.- Finalmente, se debe compilar el proyecto (de preferencia, con el compilador para depurar).



BITÁCORA DE TRABAJO

	<i>ACTIVIDADES REALIZADAS</i>			
FECHA	<i>ERIK ALEJANDRO GÓMEZ MARTÍNEZ</i>	<i>ISRAEL ALEJANDRO MORA GONZÁLEZ</i>	<i>JOSÉ EMMANUEL RODRÍGUEZ LÓPEZ</i>	<i>ÁNGEL GABRIEL GALINDO LÓPEZ</i>
21 DE NOVIEMBRE	Prototipo del sistema de archivos	Diseño y planificación de la interfaz	Diseño y planificación de la interfaz	Diseño y planificación de la interfaz
22 DE NOVIEMBRE	Cálculo de tamaños y proporciones para los assets	Investigación acerca de los distintos algoritmos de ordenamiento	Investigación acerca de los distintos algoritmos de ordenamiento	Diseño y elaboración de assets
23 DE NOVIEMBRE	Cálculo de tamaños y proporciones para los assets	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
24 DE NOVIEMBRE	Cálculo de tamaños y proporciones para los assets	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
25 DE NOVIEMBRE	Implementación de menús de récords, dificultades y modos de juego	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
26 DE NOVIEMBRE	Desarrollo del sistema de tableros	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
27 DE NOVIEMBRE	Finalización de prototipos para las interfaces	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
28 DE NOVIEMBRE	Planificación sobre Branch and Bound	Planificación sobre Branch and Bound	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets
29 DE NOVIEMBRE	Implementación del código en POO	Desarrollo del código para probar algoritmos de ordenamiento	Desarrollo del código para probar algoritmos de ordenamiento	Diseño y elaboración de assets

30 DE NOVIEMBRE	Implementación del código en POO	Planificación sobre Branch and Bound	Planificación sobre Branch and Bound	Diseño y elaboración de assets
1 DE DICIEMBRE	Implementación del código en POO	Planificación sobre Branch and Bound	Planificación sobre Branch and Bound	Diseño y elaboración de assets
2 DE DICIEMBRE	Implementación del código en POO	Investigación sobre Branch and Bound	Investigación sobre Branch and Bound	Diseño y elaboración de assets
3 DE DICIEMBRE	Código implementado en POO	Desarrollo sobre Branch and Bound	Investigación sobre cuestiones de algoritmos	Diseño y elaboración de assets
4 DE DICIEMBRE	Rediseño de mecánicas e interfaces	Desarrollo sobre Branch and Bound	Investigación sobre cuestiones de algoritmos	Diseño y elaboración de assets
5 DE DICIEMBRE	Rediseño de mecánicas e interfaces	Desarrollo sobre Branch and Bound	Investigación sobre cuestiones de algoritmos	Diseño y elaboración de assets
6 DE DICIEMBRE	Rediseño de mecánicas e interfaces	Rediseño del código	Desarrollo de la documentación	Diseño y elaboración de assets
7 DE DICIEMBRE	Adición de música y efectos de sonidos	Adición de música y efectos de sonidos	Desarrollo de la documentación	Diseño y elaboración de assets
8 DE DICIEMBRE	Prototipo de impresión para los tableros	Adición de música y efectos de sonidos	Desarrollo de la documentación	Diseño y elaboración de assets
9 DE DICIEMBRE	Prototipo de impresión para los tableros	Sistema para validar puzles aleatorios	Desarrollo de la documentación	Diseño y elaboración de assets
10 DE DICIEMBRE	Desarrollo del modo manual para el juego	Sistema para validar puzles aleatorios	Desarrollo de la documentación	Diseño y elaboración de assets
11 DE DICIEMBRE	Desarrollo del modo manual para el juego	Sistema para validar puzles aleatorios	Desarrollo de la documentación	Diseño y elaboración de assets

12 DE DICIEMBRE	Desarrollo de la documentación	Implementación de Branch and Bound	Desarrollo de la documentación	Diseño y elaboración de assets
13 DE DICIEMBRE	Desarrollo de la documentación	Implementación de Branch and Bound	Desarrollo de la documentación	Diseño y elaboración de assets
14 DE DICIEMBRE	Sistema para construir puzzles en el modo automático	Implementación de Branch and Bound	Desarrollo de la documentación	Diseño y elaboración de assets
15 DE DICIEMBRE	Sistema para construir puzzles en el modo automático	Implementación de Branch and Bound	Desarrollo de la documentación	Diseño y elaboración de assets
16 DE DICIEMBRE	Sistema para construir puzzles en el modo automático	Implementación de Branch and Bound	Desarrollo de la documentación	Diseño y elaboración de assets
17 DE DICIEMBRE	Incorporación de assets	Depuración del esquema Branch and Bound	Desarrollo de la documentación	Finalización de assets
18 DE DICIEMBRE	Desarrollo de la documentación	Depuración del esquema Branch and Bound	Desarrollo de la documentación	Depuración de código y testeo del programa
19 DE DICIEMBRE	Desarrollo de la documentación	Depuración del esquema Branch and Bound	Desarrollo de la documentación	Depuración de código y testeo del programa
20 DE DICIEMBRE	Finalización de la documentación	Depuración del esquema Branch and Bound	Finalización de la documentación	Depuración de código y testeo del programa

CONCLUSIONES

ERIK ALEJANDRO GÓMEZ MARTÍNEZ

El sistema de cabeceras .hpp es una excelente forma de organizar el código, ya que permite separar funcionalidades específicas (clases, métodos, funciones, herencia, polimorfismo, etcétera) y, por ende, lograr una estructuración mucho más legible. Asimismo, los objetos «vector», de la SLT (Standard Template Library, o Librería de Plantillas Estándar), y «string» han sido de muchísima utilidad, ya que, por ejemplo, declarar arreglos dinámicos, permiten agilizar el diseño algorítmico (operar con «char *» parece arcaico ahora).

Quizás Allegro sea una librería muy equipada y potente, pero carece de portabilidad (es difícilísimo instalarla en cualquier tipo de editor), además de solo ofrecer opciones muy arcaicas para el diseño de interfaces (a diferencia del sistema de Java, es necesario modificar, compilar y ejecutar varias veces el programa, si es que se desea posicionar adecuadamente íconos), siendo algo complicada, y tediosa, de usar.

ISRAEL ALEJANDRO MORA GONZÁLEZ

Sin lugar a duda, la Programación Orientada a Objetos es un paradigma bastante potente, lo cual se hace evidente en la creación de aplicaciones de mayor complejidad, las cuales, incluso, serían inimaginables sin el empleo de clases, atributos y métodos. Durante el desarrollo de este proyecto (el juego N-Puzzle), se aplicaron los conocimientos vistos en Programación II, y, de hecho, representó enormes retos que los requirieron forzosamente.

Por otro lado, es importante señalar que el uso de las estructuras de datos, aprendidas durante este período, fue fundamental para el desarrollo de ambas partes del proyecto (juego y comparación de algoritmos), ya que son piezas fundamentales de cualquier lenguaje de programación, siendo herramientas muy eficientes a la hora de resolver problemas complejos, como los que se nos presentaron. Sin duda, estoy satisfecho con el trabajo realizado y los conocimientos adquiridos.

ÁNGEL GABRIEL GALINDO LÓPEZ

Los proyectos finales, que realizamos en cada semestre, realmente ayudan a fortalecer las destrezas para trabajar en conjunto con otras personas, combinando así todas las habilidades y, de esta forma, generar entregas de calidad. Respecto a mi caso personal, hice todo lo que pude (destacando el desarrollo de los assets) en el avance de nuestro, tratando de realizar un esfuerzo equivalente a mis compañeros del equipo.

Es curioso que nosotros, quienes hemos desarrollado videojuegos realmente simples y pequeños (un Pac-Man es muy distinto a un juego AAA, como GTAV, por ejemplo) representen grandes retos: estructurar los esquemas gráficos, interfaces, fondos, íconos, letreros, algoritmos, mecánicas, etcétera. Ahora entiendo la razón por la cual enormes corporativos (Microsoft, Sony o Nintendo) invierten miles de dólares para crear sus productos.

JOSÉ EMMANUEL RODRÍGUEZ LÓPEZ

En mi caso personal, este proyecto fue una excelente oportunidad para reforzar los conocimientos adquiridos en clase, y, sobre todo, aclarar dudas respecto a la programación misma (la educación a distancia representa cierto reto, en mayor medida, por los distractores domésticos). De igual forma, aprendí acerca de la adecuada implementación de clases (constructores, herencia, polimorfismo, sobrecarga de operadores o métodos, entre otros); incluso, pude comprender una de las aplicaciones más valiosas de los árboles binarios: esquematizar soluciones. Sin duda alguna, también incrementé mis destrezas en el ámbito de la investigación (encontrar fuentes confiables, recabar las ideas globales, resumir el contenido, etcétera) y trabajo en equipo.

BIBLIOGRAFÍA

- 1.- Celis Hernández, R. (junio de 2015). *Medir tiempo de ejecución en C++*. Obtenido de:
[https://mascandobits.es/programacion/medir-tiempo-de-ejecucion-en-c/#:~:text=double%20time%20%3D%20\(%20double%20\(t1,traducir%20los%20ticks%20a%20segundos.](https://mascandobits.es/programacion/medir-tiempo-de-ejecucion-en-c/#:~:text=double%20time%20%3D%20(%20double%20(t1,traducir%20los%20ticks%20a%20segundos.)
- 2.- Cid, F. (marzo de 2014). *Complejidad de algoritmos*. Obtenido de:
<https://es.slideshare.net/Francocidacuna/complejidad-de-algoritmos-32683525>
- 3.- Serna Pérez, E. (s.f.). 5.- *Ordenación y búsqueda*. Universidad Autónoma de Aguascalientes: Aguascalientes. Obtenido de:
<https://es.slideshare.net/Francocidacuna/complejidad-de-algoritmos-32683525>
- 4.- Báez, L. (s.f.). *Análisis de cota superior asintótica*. Obtenido de:
http://www.luchonet.com.ar/aed/?page_id=226
- 5.- Universidad Rey Juan Carlos. (s.f.). *Tema 2.2: Notaciones Asintóticas*. Obtenido de:
https://www.cartagena99.com/recursos/alumnos/temarios/Notaciones_asintoticas.pdf
- 6.- Collier, A. (abril de 2019). *Sliding Puzzle Solvable?* Obtenido de:
<https://datawookie.netlify.app/blog/2019/04/sliding-puzzle-solvable/>
- 7.- *How to check if an instance of 15 puzzle is solvable?* (octubre de 2020). Obtenido de:
<https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/>