



UNIVERSIDAD AUTÓNOMA
DE AGUASCALIENTES

PROYECTO FINAL

“Klotski”



Universidad autónoma
de Aguascalientes.

Centro de Ciencias
Básicas

Ingeniería en Sistemas
Computacionales

Aguascalientes, Ags.

Resumen descriptivo

Fortalezas

Las fortalezas del programa es que puede correr y ser compilado en cualquier sistema operativo Windows si tiene el compilador de GNU de C++ (g++) ya sea a través del proyecto MINGW64 o MSYS2, o en sistemas operativos similares a UNIX como lo sería MacOS X en adelante o sistemas operativos basados en Linux con el compilador de GNU de C++ (g++) e incluyendo la librería de "ncurses.h".

Relacionado a la compilación si se usa el programa de GNU Make para compilar el programa, este ya incluye todas las configuraciones y optimizaciones para generar el binario del programa, por lo que con tan solo ingresar el comando "make" se genera un archivo llamado "Klotski-"seguido del identificador del sistema operativo junto con su forma de ejecutable en Windows sería "Klotski-Windows.exe", donde dicho binario tiene un peso promedio de maso menos 100 KB de almacenamiento secundario cuando se aplica la bandera strip (cuando no tiene icono el binario).

Otra fortaleza del programa es que su código está organizado en varios archivos dando lugar a una manera fácil de acceder a cada parte del código, mientras que a su vez hay comentarios que explican que es lo que pasa en varias partes del código

También el programa tiene pantallas y menús gráficos que son agradables al usuario donde también cada entrada ya sea de teclado o de archivos del usuario se validan, dando lugar a que el programa de mensajes de error en lugar de "romperse".

Debilidades

Las debilidades del programa recaen en el tablero ya que, si este no tiene solución incluso al ser validado por el validador de archivos, el algoritmo de solución intentara solucionarlo, gastando así recursos del sistema.

Pero otra debilidad del programa es en la cuestión en la que el algoritmo del solucionador implementa de manera abstracta los métodos de DFS y back tracking, ya que se usan distintas estructuras de datos en memoria dando lugar a un código el cual, aunque incluya comentarios puede ser abstracto debido a las diferentes funciones de cada archivo y tipos de datos, donde también se utilizan arreglos de tipo char donde el índice es la pieza a mover donde se pierda algo de memoria al costo de velocidad de ejecución y escritura, debido a esta solución no se encuentra el camino más corto pero se garantiza la solución en cualquier tablero soluble.

La última desventaja del programa es que este es relativamente "pesado" en estado de ejecución, debido a los diferentes tipos y estructuras de datos usadas en la solución y al mostrar dicha solución el programa usa máximo 6.4 MB de memoria principal, lo cual puede ser un problema si no se tiene acceso a dicha cantidad de memoria o mecanismos de desbordamiento de esta.

Resumen del archivo compatibilidad.h

Funciones:

- **clrscr()**: Limpia la pantalla de la consola usando secuencias de escape ANSI para sistemas UNIX y Windows y en Windows usa `system("cls");`.
- **delay(ms)**: Introduce una pausa en milisegundos, utilizando `usleep`.
- **gotoxy(int x, int y)**: Mueve el cursor a la posición especificada en la pantalla.
- **setANSI()**: Inicia la consola virtual y habilita el procesamiento de secuencias de escape ANSI en Windows.
- **setUTF8()**: Establece la codificación de caracteres a UTF-8 para compatibilidad con Unicode en Windows.
- **startCompat()**: Inicia el entorno de compatibilidad según el sistema operativo.
- **endCompat()**: Finaliza el entorno de compatibilidad.

Definiciones:

- Definiciones para teclas compartidas como `KEY_SPACE`, `KEY_TAB`, etc.
- Modificadores de texto como `BOLD_ON`, `BOLD_OFF`, `UNDERLINE_ON`, etc.
- Modificadores de terminal como `RESET_COLOR`, `CURSOR_ON`, `CURSOR_OFF`, etc.
- Colores ANSI para el texto y el fondo de este como `FG_BLUE`, `BG_BLUE`, etc.
- Compatibilidad con teclas y funciones específicas de Windows utilizando `#ifdef`.

Resumen del archivo TiposDeDatos.h :

Definiciones y Enumeraciones:

- **LIMITE_DE_CHAR:** Define el límite superior para el tipo de dato char.
- **TipoDeSolucion:** Enumeración para representar los estados de la solución del juego, como "SOLUCION_ENCONTRADA" y "EN_PROGRESO".
- **Posicion:** Clase que representa las coordenadas x e y de una posición en el tablero.
- **TipoDePieza:** Enumeración que define los tipos de piezas en el tablero, como "PIEZA_VACIA", "PIEZA_PARED", "PIEZA_SINGULAR", "PIEZA_PUERTA" y "PIEZA_OBJETIVO".
- **Direccion:** Enumeración que representa las direcciones posibles para mover las piezas, como "ARRIBA", "ABAJO", "IZQUIERDA" y "DERECHA".
- **OrdenDeMovimiento:** Clase que guarda información sobre la dirección e identificación de un bloque, utilizada en las funciones `solucion()` y `printMovimientosSolucion()`
- **Solucion:** Clase que encapsula información relacionada con el estado de la solución del juego Klotski, incluyendo el estado actual, la profundidad, el último hash y movimiento realizado.
- **direccionOpuesta(Direccion dir):** Función que devuelve la dirección opuesta a la proporcionada, utilizada para evitar movimientos innecesarios o redundantes.

Resumen del archivo FuncionesAuxiliares.h :

El archivo `FuncionesAuxiliares.h` contiene funciones auxiliares utilizadas en el juego Klotski. Aquí se presenta un resumen de las funciones y su propósito:

Funciones y Definiciones:

- **stringDireccion(Direccion dir):** Función que convierte una dirección en enumeración a su representación en cadena para impresión en pantalla.
- **recuadro():** Función que imprime un recuadro en la pantalla, utilizada para destacar mensajes o secciones en la interfaz.
- **validarEntradaInt():** Función auxiliar que solicita al usuario ingresar un número entero y valida la entrada para asegurar que sea un número válido.

Resumen de la Clase Nivel:

Atributos privados:

- **numNivel**: Número del nivel.
- **nombreArchivo**: Nombre del archivo que contiene la información del nivel.
- **nombreNivel**: Nombre del nivel.
- **anchoNivel, altoNivel**: Dimensiones del tablero del nivel.
- **tableroNivel**: Matriz que representa el tablero del nivel.

Métodos privados:

- **leerNombreNivel(ifstream& archivo)**: Lee el nombre del nivel desde un archivo.
- **leerDimensionesTablero(ifstream& archivo)**: Lee las dimensiones del tablero desde un archivo.
- **leerTablero(ifstream& archivo)**: Lee el tablero desde un archivo.

Métodos públicos:

- **Nivel()**: Constructor vacío.
- **Nivel(unsigned int numNivel)**: Constructor.
- **cargarNivel()**: Carga la información del nivel desde un archivo.
- **revisarCaracteres()**: Revisa los caracteres del tablero.
- **cambiarVacioPorAmpersand()**: Cambia espacios por ampersands en el tablero.
- **tieneSalidaYsingular()**: Verifica si hay una salida y una pieza singular en el tablero.
- **getNombreNivel()**: Obtiene el nombre del nivel.
- **getNombreArchivo()**: Obtiene el nombre del archivo.
- **getAnchoNivel()**: Obtiene el ancho del nivel.
- **getAltoNivel()**: Obtiene el alto del nivel.
- **getTableroNivel()**: Obtiene el tablero del nivel.

Resumen de la Clase Tabla:

Atributos:

- **bloques**: Un array de objetos `Bloque` que representa las piezas en el tablero.
- **tableroDeJuego**: Una matriz que representa el estado actual del juego.
- **baseDelTablero**: Una copia de seguridad del estado original del tablero.

Métodos:

- **imprimirBloques()**: Imprime en la consola información detallada sobre cada bloque en el tablero.
- **getBloques()**: Devuelve un puntero al array de bloques.
- **getAltoTablero()**: Devuelve la altura del tablero.
- **getAnchoTablero()**: Devuelve el ancho del tablero.
- **printTabla()**: Imprime en la consola el estado actual del tablero.
- **bloquePuedeMoverse(Direccion, char)**: Verifica si un bloque puede moverse en una dirección específica.
- **moverBloque(Direccion, char)**: Mueve un bloque en una dirección específica y actualiza el tablero.
- **juegoGanado()**: Verifica si se ha ganado el juego.

Funciones Relacionadas:

- **Solucionador()**: Función principal para encontrar la solución al juego Klotski.
- **buscarSolucion(unsigned int&, OrdenDeMovimiento&)**: Función recursiva de búsqueda de solución utilizando DFS y Backtracking.
- **printMovimientosSolucion(unsigned int)**: Imprime los movimientos de la solución en la consola.

Amistad con Klotski:

La clase `Tabla` es amiga de la clase `Klotski`, lo que permite a `Klotski` acceder a los miembros privados de `Tabla`. Esto facilita la implementación de la solución del juego.

Resumen de la Clase Bloque:

Atributos:

- **x, y**: Coordenadas que indican la posición del bloque en el tablero.
- **ancho, alto**: Dimensiones del bloque.
- **esPiezaSingular**: Bandera que indica si el bloque es la pieza singular del juego.
- **id**: Identificación única del bloque, representada por caracteres ASCII y otros caracteres.
- **reduccion**: Valor único asociado a la combinación de ancho y alto del bloque, utilizado para la optimización de hash.

Métodos:

- **Constructores**: Dos constructores, uno vacío y otro que inicializa los atributos con valores dados.
- **Funciones Get**: Métodos para obtener valores específicos del bloque, como ancho, alto, ID, reducción, coordenadas x e y.
- **puedeMoverse(char pieza)**: Verifica si el bloque puede moverse a una posición específica en el tablero según la pieza en esa posición.
- **mover(Direccion dir)**: Modifica las coordenadas x e y del bloque para simular su movimiento en una dirección dada.

Resumen de la Clase Klotski:

Atributos:

- **memoria:** Una tabla de hash de tipo `unordered_map` para almacenar información relacionada con cada estado del juego.
- **profundidad:** Profundidad de la solución.
- **tablaSolucion:** Tabla del juego modificada en la ejecución del programa.
- **tablaOriginal:** Tabla original del juego (copia de seguridad).

Métodos:

- **Función de hash:** función de hash recuperada de la librería Boost de C++.
- **Constructores:** Constructor vacío y constructor con parámetro para inicializar la tabla de solución.
- **printMovimientosSolucion(unsigned int estadoDelHash, string nombreNivel):** Imprime los movimientos de la solución en la terminal mostrando también la tabla en dicho estado.
- **Solucionador():** Función principal para encontrar la solución.
- **buscarSolucion(unsigned int& ultimoHash, OrdenDeMovimiento& ultimoOrden):** Función recursiva utilizando DFS y Backtracking para encontrar la solución.
- **rotacionDeDireccion(unsigned int num):** Realiza una rotación de dirección cada vez que se llama a la función dando una solución de Klotski consistente.

Algoritmo de Solución en la Clase Klotski

1. Inicialización de la Tabla y la Memoria:

- Se crea una instancia de la clase `Klotski` con la tabla de juego proporcionada como argumento.
- La memoria (`unordered_map`) se inicializa con el hash del estado inicial de la tabla y se marca como "en progreso" (`EN_PROGRESO`).

2. Búsqueda de Solución (DFS con Backtracking):

- Se realiza una iteración sobre todas las piezas en el tablero para explorar posibles movimientos y configuraciones en el ciclo `for`.
- Para cada pieza, se selecciona una dirección inicial con `rotacionDeDireccion` y se intenta mover la pieza en esa dirección.
- Si el movimiento es válido y lleva a un nuevo estado del tablero, se calcula el hash de ese nuevo estado y se procede a explorar desde ese nuevo estado llamando recursivamente a la función `buscarSolucion`.
- Se almacena la información del nuevo estado en la memoria, marcándolo como "en progreso" y registrando la profundidad, el último hash y el último movimiento realizado.
- Si se encuentra una solución (por ejemplo, si la pieza singular llega a la posición objetivo), se registra como "solución encontrada" (`SOLUCION_ENCONTRADA`) y se devuelve el hash del estado actual.
- Si no se encuentra una solución desde el estado actual, se utiliza la técnica de Backtracking: se retrocede al estado anterior y se continúa explorando desde ese punto.
- Para definir si no existe solución se toma en cuenta el límite de (`LIMITE_DE_PROFUNDIDAD`) y también se considera si no se pueden mover las piezas.

3. Impresión de la Solución:

- Una vez encontrada la solución, se utiliza la información almacenada en la memoria para reconstruir la secuencia de movimientos realizados desde el estado inicial hasta el estado final.
- Se crea una tabla para imprimir los tableros con respecto a los movimientos.
- Se utiliza una pila (stack) para almacenar los movimientos en orden inverso (desde el final hasta el principio).
- Se imprime cada tablero moviendo la pieza correspondiente a la pila.
- Se imprime cada movimiento, incluyendo la pieza movida, la dirección y el paso en el que se realizó considerando que no se haya movido la misma pieza en el paso anterior.

En resumen, el algoritmo realiza una búsqueda exhaustiva del espacio de estados, utilizando la técnica DFS para explorar en profundidad y el Backtracking para retroceder cuando se alcanzan situaciones sin solución desde un estado particular. La memoria se utiliza para evitar explorar estados ya visitados, mejorando la eficiencia del algoritmo.

Temas investigados

Klotski

El nombre "Klotski" en sí mismo proviene de la palabra polaca que significa "bloque pequeño", este rompecabezas, consta de una cuadrícula rectangular que presenta un bloque grande, y varios bloques más pequeños, cada uno de los cuales se puede deslizar en una dirección específica para liberar el bloque más grande a la salida, intentando hacerlo en el menor número de movimientos posibles.

El rompecabezas consta de un número predeterminado de bloques, cada uno con un tamaño, forma y colores únicos. Entre los bloques, se encuentran uno grande de 2×2 unidades, cuatro rectángulos medianos de 2×1 unidades y cuatro cuadrados pequeños de 1×1 unidad.

Compatibilidad de Windows y Sistemas basados en Unix

Es crucial lograr compatibilidad entre el código de Windows y sistemas basados en Unix compatibles con estándares POSIX como lo sería Linux y Mac Os X. Esto con la finalidad de obtener un código portable que funcione en casi cualquier sistema operativo y no solo el sistema Operativo de Microsoft, Windows.

Esto se logra mediante el uso de directivas del compilador GCC para detectar el sistema operativo y se usan caracteres de escape ANSI de C para la manipulación de caracteres en la terminal, aunque en Windows no está activada directamente la dicha compatibilidad, pero usando la librería windows.h y sus funciones API de sistema, se activa una "consola virtual" para poder soportar estos caracteres ANSI y también se pueden activar la compatibilidad con caracteres Unicode con codificación UTF8, en la consola de Windows, CMD.

Para la entrada del usuario, se optó por la función getch() de conio.h y la función getch() de ncurses.h, adaptada para trabajar como la de conio.h en Windows.

Archivos

Los archivos en C++ son fundamentales para leer y escribir datos en dispositivos de almacenamiento como disco duro, memoria USB, etc. Puedes usar la biblioteca de entrada/salida (iostream) de C++ junto con las clases ifstream y ofstream para trabajar con archivos de entrada (lectura) y salida (escritura), respectivamente. La biblioteca fstream proporciona las clases ofstream y ifstream para escribir y leer archivos respectivamente. Los archivos en C++ son útiles para almacenar datos persistentes, realizar operaciones de lectura/escritura de configuraciones, guardar resultados de cálculos, entre muchas otras aplicaciones. Son una parte esencial en el manejo de datos fuera de la memoria volátil.

Arboles

Los árboles son una forma de estructura de datos no lineal que consiste en nodos conectados por aristas. La jerarquía inherente de los árboles los hace ideales para representar relaciones parentales, organización de datos y algoritmos de búsqueda eficientes.

La Estructura Básica de estos es:

Un árbol consta de nodos, donde uno de ellos se designa como nodo raíz. Cada nodo puede tener cero o más nodos secundarios, formando subárboles. Los nodos sin hijos se llaman hojas. La conexión entre nodos se denomina arista o borde.

Existen diferentes tipos de Árboles los cuales son:

Árboles Binarios: Cada nodo tiene como máximo dos hijos: uno izquierdo y otro derecho. Utilizados en estructuras de búsqueda como Árboles Binarios de Búsqueda (BST).

Árbol n-ario: los árboles n-arios son aquellos arboles donde el número máximo de hijos por nodo es de N , en la figura 7 podemos apreciar dos árboles con grado 2 y grado 3, estos dos árboles también los podemos definir como Árbol n-ario con $n = 2$ y $n=3$ respectivamente.

Árboles AVL: Árboles binarios balanceados automáticamente para garantizar operaciones eficientes.

Árboles B: Estructuras de datos utilizadas en sistemas de archivos y bases de datos para la organización eficiente de grandes conjuntos de datos.

Operaciones Comunes:

Inserción: Agregar un nuevo nodo al árbol.

Búsqueda: Encontrar un valor específico en el árbol.

Eliminación: Eliminar un nodo con un valor dado del árbol.

Recorridos: Explorar y procesar nodos en un orden específico (inorden, preorden, postorden).

Hash

En C++, una función hash es una función que toma una entrada y produce una salida de tamaño fijo, generalmente utilizada para mapear datos de tamaño variable a datos de tamaño fijo. Estas funciones son fundamentales en diversas aplicaciones, desde tablas hash para almacenar datos hasta encriptación y verificación de integridad de datos.

Uso de funciones hash.

Estructuras de datos: Las tablas hash son estructuras de datos comunes que utilizan funciones hash para indexar datos y permitir un acceso rápido. En C++, puedes implementar tablas hash utilizando contenedores como `std::unordered_map` o `std::unordered_set`

Criptografía: Las funciones hash criptográficas se utilizan para garantizar la integridad de los datos y en la generación de firmas digitales. C++ proporciona implementaciones de funciones hash criptográficas como SHA-1, SHA-256, SHA-512, etc., a través de bibliotecas como OpenSSL o funciones estándar como `std::hash` para tipos de datos estándar.

Implementación en C++.

En C++, puedes implementar funciones hash personalizadas para tipos de datos personalizados o especializados.

Consideraciones:

Colisiones: Las funciones hash pueden producir colisiones, donde dos entradas diferentes generan el mismo valor hash. Para aplicaciones críticas, se necesitan funciones hash que minimicen estas colisiones.

Seguridad: Las funciones hash criptográficas deben ser resistentes a los intentos de inversión y colisiones deliberadas para garantizar la seguridad de los datos.

Bibliotecas en C++ para funciones hash

Biblioteca estándar: C++ proporciona `std::hash` para tipos de datos estándar y contenedores de la STL.

Bibliotecas externas: Para funciones hash criptográficas más avanzadas, se pueden utilizar bibliotecas como OpenSSL, Crypto++, etc.

Las funciones hash son herramientas poderosas en el desarrollo de software que se utilizan para múltiples propósitos, desde la optimización del acceso a datos hasta la seguridad en la manipulación de información sensible, etc.

unordered_map de std c++

¿Qué es "std::unordered_map"?

Estructura de datos: Es una estructura de contenedor que almacena elementos como pares de clave-valor.

implementación basada en tablas hash: Utiliza una función hash para calcular el índice donde se almacena cada elemento. Esto permite un acceso rápido a los elementos según su clave.

Características principales:

Rápido acceso: La complejidad promedio para búsqueda, inserción y eliminación es $O(1)$ en el mejor caso, aunque puede llegar a ser $O(n)$ en el peor caso si hay colisiones.

Claves únicas: Cada clave en el es única; no puede haber duplicados de claves."std::unordered_map"

Iteradores: Se pueden utilizar iteradores para recorrer los elementos del mapa, ya sea en orden o de forma aleatoria.

Consideraciones importantes:

Función hash: Para tipos de datos personalizados como estructuras, es necesario proporcionar una función hash personalizada.

Colisiones: A pesar de su eficiencia, las colisiones pueden degradar el rendimiento. Una buena función hash y un manejo adecuado de la capacidad pueden reducir la posibilidad de colisiones.

Cuándo usar:" std::unordered_map".

Búsqueda eficiente: Ideal para aplicaciones donde se necesitan búsquedas rápidas y se pueden tolerar variaciones en el tiempo de acceso.

No se requiere orden específico: A diferencia de, no garantiza un orden específico en sus elementos." std::map" "std::unordered_map"

"std::unordered_map" es una herramienta poderosa en C++ para almacenar datos asociativos de manera eficiente y proporcionar acceso rápido a los elementos a través de claves únicas. Su implementación basada en tablas hash lo hace ideal para una amplia gama de aplicaciones donde se requiere acceso eficiente a los datos.

Árbol-Rojo-Negro

Un árbol rojo-negro es una estructura de datos en forma de árbol binario de búsqueda que cumple ciertas reglas adicionales que garantizan un balance relativo. Es una variante de los árboles binarios de búsqueda que se utiliza comúnmente en la implementación de estructuras de datos en muchos lenguajes de programación, incluido C++.

Características principales:

Estructura de árbol binario: Cada nodo tiene a lo sumo dos hijos: un hijo izquierdo y un hijo derecho.

Reglas de equilibrio: Los árboles rojo-negro se mantienen balanceados para asegurar que la altura del árbol sea relativamente pequeña en comparación con otros árboles de búsqueda. Esto ayuda a garantizar un tiempo de búsqueda eficiente.

Nodos coloreados: Cada nodo del árbol está coloreado con rojo o negro según ciertas reglas que garantizan el balance. Las reglas específicas sobre la coloración de los nodos incluyen:

Todos los nodos son rojos o negros.

La raíz siempre es negra.

Las hojas (nodos nulos) son consideradas negras.

No puede haber dos nodos rojos adyacentes (un nodo rojo no puede tener un hijo rojo).

Todos los caminos desde cualquier nodo hasta sus hojas descendientes tienen el mismo número de nodos negros.

Operaciones eficientes: Proporciona operaciones de inserción, eliminación y búsqueda con tiempos de ejecución garantizados en el peor de los casos, lo que lo hace útil en aplicaciones donde la eficiencia es crucial.

Usos:

Los árboles rojo-negro se utilizan en implementaciones de bases de datos, sistemas de archivos, compiladores y en diversas áreas donde se requiere una estructura de datos que ofrezca una eficiencia predecible en las operaciones de inserción, eliminación y búsqueda.

Ventajas:

Garantizan tiempos de búsqueda, inserción y eliminación relativamente eficientes. Proporcionan un balanceamiento automático, lo que asegura una altura del árbol logarítmica en función del número de nodos.

Son útiles en aplicaciones donde se necesita un rendimiento predecible incluso en casos de peor escenario.

Desventajas:

La implementación de un árbol rojo-negro puede ser más compleja que otras estructuras de datos debido a las reglas de balanceo y coloración de nodos.

Requieren un mantenimiento adicional para garantizar que siempre cumplan con las reglas de equilibrio.

En C++, existen bibliotecas y estructuras de datos implementadas que incluyen árboles rojo-negro para facilitar su uso en aplicaciones que requieran estas características de balance y eficiencia.

Funciones Booleanas

Las funciones booleanas en C++ son piezas de código que evalúan expresiones lógicas y devuelven un valor booleano (verdadero o falso) como resultado. Estas funciones son esenciales para controlar el flujo del programa, realizar comparaciones y tomar decisiones basadas en condiciones lógicas.

¿Qué son las funciones booleanas en C++?

Definición: Son funciones que devuelven un valor booleano después de evaluar una expresión lógica.

Evaluación lógica: Pueden involucrar operadores de comparación (>, <, ==, !=) y operadores lógicos (&&, ||, !) para comparar valores o expresiones y producir un resultado booleano.

Utilidad y aplicaciones:

Control de flujo: Se usan en declaraciones condicionales (if, else, ifelse) para ejecutar o no un bloque de código basado en una condición booleana.

Funciones de validación: Se emplean para verificar condiciones y validar datos antes de realizar operaciones.

Algoritmos lógicos: Son fundamentales para la implementación de algoritmos y estructuras de datos que requieren toma de decisiones basada en lógica.

Características principales:

Devuelven valores booleanos: El resultado de una función booleana es verdadero (true) o falso (false).

Utilizan expresiones lógicas: Pueden contener operadores de comparación y lógicos para evaluar condiciones.

Pueden ser funciones personalizadas: Además de usar operadores de comparación directamente, se pueden crear funciones propias que realicen evaluaciones más complejas y retornen un resultado booleano.

Iteradores de c++

Los iteradores en C++ son herramientas que permiten recorrer y acceder a elementos dentro de contenedores, como arrays, listas, mapas, etc. Son similares a los punteros en su funcionalidad, ya que permiten desplazarse entre elementos, pero proporcionan una interfaz más abstracta y segura para trabajar con estructuras de datos.

¿Qué son los iteradores en C++?

Definición: Los iteradores son objetos que actúan como una abstracción para acceder secuencialmente a los elementos dentro de un contenedor.

Abstracción sobre estructuras de datos: Permiten recorrer los elementos de una colección (array, lista, mapa, etc.) sin exponer los detalles internos de la implementación.

Utilidad y aplicaciones:

Recorrido de contenedores: Permiten acceder a los elementos almacenados en una estructura de datos de manera secuencial.

Manipulación de datos: Facilitan la inserción, eliminación o modificación de elementos dentro de un contenedor de forma segura.

Algoritmos estándar de la STL: Muchos algoritmos de la Biblioteca Estándar de Plantillas (STL) de C++, como, entre otros, utilizan iteradores para trabajar con contenedores. `std::sort` `std::find`

Características principales:

Abstracción de acceso: Proporcionan una interfaz uniforme para acceder a los elementos, independientemente del tipo de contenedor.

Sintaxis similar a punteros: Se utilizan operaciones similares a las de los punteros, como, (operador de desreferencia), etc.

Iteradores de finalización: Los contenedores suelen proporcionar un iterador especial para indicar el final de la colección, conocido como. `end()`

Tipos de iteradores en C++:

Iteradores de entrada: Solo permiten lectura secuencial de elementos.

Iteradores de salida: Solo permiten la escritura secuencial de elementos.

Iteradores bidireccionales: Permiten recorrer los elementos en ambas direcciones (adelante y atrás).

Iteradores aleatorios: Permiten el acceso aleatorio a los elementos (como los arrays).

Los iteradores son una parte esencial de C++, ya que proporcionan una forma versátil y segura de acceder a los elementos dentro de las estructuras de datos,

permitiendo realizar operaciones como recorrer, modificar o eliminar elementos de manera eficiente.

Paradigma de fuerza bruta

La fuerza bruta es una técnica de resolución de problemas que implica probar sistemáticamente todas las posibles soluciones hasta encontrar la correcta. En informática, se usa en algoritmos de búsqueda exhaustiva para encontrar soluciones a problemas complejos.

En C++, el paradigma de fuerza bruta se puede implementar para resolver una variedad de problemas, como encontrar la combinación correcta para un problema de optimización, calcular todas las posibles permutaciones o combinaciones de un conjunto de elementos, entre otros.

Características clave de un algoritmo de fuerza bruta en C++ podrían ser:

Exploración exhaustiva: Evalúa todas las posibles soluciones sin depender de estrategias heurísticas.

Eficiencia y tiempo de ejecución: Puede ser lento para problemas grandes debido a la evaluación de todas las posibles soluciones.

Implementación directa: Suele ser relativamente sencillo de implementar y comprender.

Aplicabilidad general: Puede aplicarse a una amplia gama de problemas, pero no siempre es la opción óptima.

Un ejemplo sencillo podría ser un algoritmo para encontrar la suma de dos números que sea igual a un valor dado. Aquí, podrías iterar a través de todos los pares posibles de números y verificar si su suma coincide con el valor objetivo. Aunque es una solución válida, no es la más eficiente para problemas más grandes debido a la cantidad de iteraciones necesarias.

Tipo unsigned int

En C++, es un tipo de dato que representa enteros sin signo. Mientras que el tipo puede representar tanto números positivos como negativos, solo puede representar números enteros no negativos (positivos y el valor 0). "unsigned int" "int unsigned" "int"

¿Qué es unsigned int?

Representación sin signo: Este tipo de dato no permite almacenar números negativos. Su rango se extiende hasta el doble del tamaño del tipo (en la mayoría de las implementaciones). "int"

¿Para qué se utiliza unsigned int?

Representación de valores no negativos: Cuando se necesita almacenar únicamente valores positivos o enteros no negativos, es útil. Por ejemplo, para contadores, índices de bucles o cualquier variable donde se sepa que el valor nunca será negativo. "unsigned int!"

Características clave:

Rango mayor para números positivos: A diferencia del, que puede almacenar números positivos y negativos en un rango simétrico, tiene un rango que comienza en 0 y llega a un valor máximo mayor (dependiendo de la arquitectura del sistema). "int unsigned"

Uso en bucles y estructuras de control: Se usa a menudo para índices de bucles y para almacenar valores que nunca deben ser negativos, como tamaños de arrays o conteos. "for"

DFS (Depth-First Search o Búsqueda en Profundidad)

En nuestro caso se utilizó este método en lugar de BFS, ya que después de investigar un poco nos dimos cuenta de que le mejor, más rápido y fácil de implementar era este.

Es un algoritmo fundamental en la teoría de grafos y se utiliza para recorrer o buscar en estructuras de datos como grafos o árboles. Se basa en un enfoque de exploración en profundidad, es decir, sigue un camino lo más profundo posible en la estructura antes de retroceder.

¿Qué es DFS?

Algoritmo de búsqueda: DFS es un algoritmo de búsqueda no informada que se utiliza para recorrer o buscar en estructuras de datos, especialmente grafos y árboles.

¿Para qué se utiliza DFS?

Recorrido de grafos y árboles: DFS se utiliza para recorrer todos los nodos o vértices de un grafo o árbol de manera sistemática y exhaustiva.

Características clave:

Profundidad primero: DFS sigue un enfoque de exploración en profundidad, donde avanza tan lejos como sea posible a lo largo de un camino antes de retroceder.

Implementación recursiva o con pila: Puede implementarse de manera recursiva o utilizando una estructura de datos tipo pila (stack) para llevar un seguimiento de los nodos por visitar.

Aplicaciones diversas: Se utiliza en numerosos problemas como la búsqueda de caminos, la detección de ciclos, la determinación de componentes conectados y más.

Ejemplo conceptual:

Supongamos un grafo no dirigido con varios nodos conectados por aristas. Al aplicar DFS en este grafo, el algoritmo comenzaría en un nodo inicial, exploraría uno de sus vecinos, y luego seguiría avanzando hasta llegar al final de esa rama antes de retroceder y explorar otras ramas.

Divide y Conquista

El enfoque de "divide y conquista" es una estrategia algorítmica que consiste en resolver un problema dividiéndolo en subproblemas más pequeños, resolviendo cada subproblema de forma independiente y luego combinando las soluciones para obtener la solución al problema original.

En C++, este enfoque se implementa comúnmente mediante funciones recursivas.

Back Tracking

El backtracking es una técnica algorítmica utilizada para encontrar soluciones a problemas computacionales, especialmente aquellos que implican la búsqueda sistemática de todas las posibles combinaciones para encontrar la solución óptima. Se basa en la idea de probar secuencialmente diferentes opciones, retrocediendo (backtracking) cuando se alcanza una solución incorrecta para explorar otras posibilidades.

¿Qué es el backtracking?

Técnica de búsqueda exhaustiva: Backtracking es un enfoque exhaustivo que prueba sistemáticamente todas las posibles soluciones para un problema.

¿Para qué se utiliza el backtracking?

Resolución de problemas combinatorios: Se utiliza para resolver problemas que involucran la generación y búsqueda de todas las combinaciones posibles para encontrar la solución óptima.

Características clave:

Exploración recursiva: Se implementa típicamente mediante funciones recursivas que prueban diferentes combinaciones y retroceden cuando una solución no es viable.

Uso de poda: Se emplea la poda para descartar ramas o caminos que no llevan a una solución válida, lo que ayuda a reducir la búsqueda.

Aplicaciones diversas: Se utiliza en problemas de búsqueda de caminos, problemas de optimización, acertijos, rompecabezas, entre otros.

Ejemplo conceptual:

Imagina un laberinto en el que intentas encontrar el camino más corto para llegar de un punto A a un punto B. Backtracking probaría secuencialmente diferentes rutas, retrocediendo cuando se alcanza un callejón sin salida, y explorando otras opciones hasta encontrar el camino correcto.

Continue c++

En C++, continue es una palabra clave que se utiliza dentro de bucles (como for, while o do-while) para saltar a la siguiente iteración del bucle sin ejecutar el resto del código dentro del bucle en la iteración actual. Cuando se encuentra la instrucción continue, el programa pasa directamente a la siguiente iteración del bucle, omitiendo cualquier código que esté después de esa declaración dentro de la misma iteración.

Algunas características y usos clave de continue en C++ son:

Control de flujo: Ayuda a controlar el flujo de ejecución dentro de un bucle. Por ejemplo, puedes usar continue para saltar ciertas iteraciones basadas en alguna condición específica.

Evitar la ejecución de cierto código: Cuando se alcanza la instrucción continue, el código que sigue inmediatamente después de esta instrucción dentro del bucle no se ejecuta en esa iteración.

Std::find

`std::find` es una función en C++ que se encuentra en la biblioteca estándar (`<algorithm>`) y se utiliza para buscar un valor específico dentro de un rango de elementos en contenedores como vectores, listas, arreglos, entre otros. Esta función devuelve un iterador apuntando al primer elemento encontrado que coincide con el valor buscado o al final del rango si no se encuentra.

Características y uso:

Búsqueda de valores: `std::find` se utiliza para buscar un valor específico dentro de un rango de elementos. Por ejemplo, puedes buscar un número, una cadena o cualquier otro objeto comparable.

Flexibilidad: Puede ser utilizado con una variedad de contenedores de la biblioteca estándar, como vectores, listas, arreglos, entre otros. Esto proporciona una manera genérica de buscar elementos en diferentes estructuras de datos.

Iteradores: Devuelve un iterador al primer elemento que coincide con el valor buscado dentro del rango especificado. Si no se encuentra, devuelve un iterador al final del rango.

Sintaxis simple: La sintaxis de `std::find` es sencilla y fácil de usar, lo que la hace conveniente para realizar operaciones de búsqueda sin la necesidad de escribir bucles de búsqueda personalizados.

Beneficios:

Proporciona una forma conveniente y eficiente de buscar elementos dentro de contenedores de la biblioteca estándar.

Ayuda a simplificar el código al evitar la implementación manual de algoritmos de búsqueda.

`std::find` es una herramienta poderosa y útil para realizar operaciones de búsqueda dentro de contenedores en C++, contribuyendo a escribir un código más limpio y legible al facilitar la búsqueda de elementos en colecciones de datos.

Goto

goto es una instrucción en C++ que permite transferir el control de ejecución a una etiqueta específica dentro de la misma función, ignorando la secuencia normal de ejecución del programa. A pesar de estar disponible en el lenguaje, su uso está desaconsejado en la mayoría de los casos debido a sus implicaciones en la legibilidad y mantenibilidad del código.

Características y uso:

Transferencia de control: goto se utiliza para transferir el control de ejecución a una etiqueta específica dentro de una función.

Etiquetas: Se coloca una etiqueta seguida de dos puntos en un punto determinado del código al cual se desea saltar usando goto.

Consideraciones:

El uso excesivo de goto puede hacer que el código sea más difícil de entender y depurar, ya que puede generar saltos inesperados en la lógica del programa.

Su uso incontrolado puede llevar a problemas de legibilidad y mantenibilidad, especialmente en programas grandes y complejos.

Recomendaciones:

Es preferible utilizar estructuras de control más estructuradas como bucles y condicionales (for, while, if-else) en lugar de goto.

El estándar de codificación y buenas prácticas generalmente desaconsejan el uso de goto excepto en casos muy específicos y controlados.

En resumen, aunque goto es parte del lenguaje C++ y puede ser usado, su uso está desaconsejado en la mayoría de los casos debido a sus implicaciones en la legibilidad y mantenibilidad del código. Se recomienda preferir estructuras de control más estructuradas y legibles para facilitar la comprensión y el mantenimiento del código.

Makefile (make de gnu)

Un Makefile es un archivo utilizado por el programa, una herramienta de construcción de software en entornos UNIX y similares. Está especialmente diseñado para compilar programas, gestionar dependencias y automatizar el proceso de construcción en proyectos de software, especialmente en C y C++, aunque puede ser utilizado en otros lenguajes también.

¿Qué es un Makefile?

Un Makefile es un archivo de texto que contiene reglas, objetivos y dependencias que describen cómo se debe compilar un programa o proyecto. Está compuesto de varias secciones:

Objetivos: Representan los archivos o acciones que se deben generar o ejecutar.

Reglas: Definen cómo se deben generar los objetivos a partir de otros archivos.

Dependencias: Indican las relaciones entre los archivos y objetivos, especificando qué archivos deben estar actualizados antes de construir un objetivo particular.

Funciones y características principales:

Automatización de compilación: Facilita la compilación de programas complejos al encargarse de reconstruir solo los componentes que han cambiado desde la última compilación.

Gestión de dependencias: Permite definir dependencias entre archivos fuente, encargándose de reconstruir automáticamente los archivos afectados por cambios en dependencias.

Portabilidad: Al ser compatible con sistemas UNIX y similares, hace que los proyectos sean más portables entre diferentes plataformas.

Flexibilidad: Permite a los desarrolladores definir sus propias reglas y acciones personalizadas para la compilación y construcción de un proyecto.

CC: Define el compilador a utilizar.

CFLAGS: Opciones del compilador.

TARGET: Nombre del programa final.

SRC: Lista de archivos fuente.

OBJ: Archivos objeto generados a partir de los archivos fuente.

\$(TARGET): Es el objetivo principal, depende de para su construcción. **(OBJ)%**.
o: **%.cpp:** Regla para compilar cada archivo fuente a su correspondiente archivo objeto.

clean: Regla para limpiar los archivos generados en la compilación.

En resumen, el Makefile y la herramienta simplifican y automatizan el proceso de compilación de programas, gestionando eficientemente las dependencias y optimizando el tiempo de desarrollo en proyectos de software.

Conclusiones

Llegada a la conclusión de este proyecto puedo decir que para llegar la solución de un problema complejo como lo es Klotski no solo se ocupa lo aprendido en el curso sino buscar de manera personal temas de análisis y solución de problemas usando distintos paradigmas de programación competitiva, donde el uso del tiempo es vital no solo para programar sino para aprender temas nuevos y necesarios no solo para este proyecto sino para mi vida útil como programador, en lo que destaco como lo más importante el proceso para identificar nuevos temas y procesarlos para adaptarlos a la solución del problema específico de Klotski.

Este proyecto también me ayudo a utilizar herramientas de control de versiones y trabajo en equipo como lo sería el uso de Git y GitHub donde mis compañeros y yo editábamos y comparábamos el código de uno con el otro implementando las ideas más funcionales del proyecto para lograr un proyecto unificado verificado por todos.

No solo el uso de esta herramienta me enseñó a trabajar en equipo en un proyecto de programación, sino que también ayudo mucho a mis habilidades comunicativas, de liderazgo y de trabajar en un límite de tiempo. Ya que al dictar que cosas tenía que hacer cada integrante del equipo en que tiempos determinados tenía que pedirlo de manera amable y exigir resultados requeridos para los límites de tiempo, con esto se logro un flujo de trabajo optimo con el cual pudimos terminar el proyecto en un lapso de poco más de treinta días.

En conclusión, los proyectos de programación compleja requieren no solo de conocimientos previos sino de conocimientos nuevos, junto con una mente abierta que esté dispuesta a trabajar con una constante presión de tiempo mientras que a la vez tus compañeros de equipo realizan tareas similares para al final unir dichos trabajos en un proyecto finalizado de manera correcta.

Referencias consultadas


- Alaraph (2021) - solving-the-klotski-puzzle-in-scala Recuperado de: <https://alaraph.com/2021/09/10/solving-the-klotski-puzzle-in-scala/>
- Boost c++ libraries (2005-2008 Daniel James) - Boost.ContainerHash. Recuperado de: https://www.boost.org/doc/libs/1_83_0/libs/container_hash/doc/html/hash.html#ref_hash_combine
- Brandeis (2004) - Recuperado de: <https://www.cs.brandeis.edu/~storer/JimPuzzles/ZPAGES/zzzCenturyPuzzles.html>
- Codeforces Búsqueda binaria y otros métodos de reducción a la mitad. Recuperado de: <https://codeforces.com/blog/entry/96699>
- Codeforces Recuperado de:
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press. Recuperado de: <https://monet.en.kku.ac.th/courses/EN812303/book/Introduction.to.Algorithms.4th.Edition.pdf>
- Cplusplus (2017) - reference/algorithm/find. Recuperado de: <https://cplusplus.com/reference/algorithm/find/>
- Cppreference (2022) - types/size_t. Recuperado de: https://en.cppreference.com/w/c/types/size_t
- Cppreference (2023) - cpp/language/types. Recuperado de: <https://en.cppreference.com/w/cpp/language/types>
- Delftstack (2021) - howto/cpp/makefile-in-cpp/. Recuperado de: <https://www.delftstack.com/es/howto/cpp/makefile-in-cpp/>
- DelftStack (2023) - continue-cpp. Recuperado de: <https://www.delftstack.com/es/howto/cpp/break-vs-continue-cpp/#:~:text=La%20sentencia%20continue%20es%20la%20caracter%C3%A4stica%20del%20lenguaje,en%20los%20bucles%20for%2C%20while%20o%20do%20while.>
- DelftStack (2023) - cpp-goto-line. Recuperado de: <https://www.delftstack.com/es/howto/cpp/cpp-goto-line/>
- Encora. (2020). DFS vs BFS. Recuperado de: <https://www.encora.com/es/blog/dfs-vs-bfs>
- Estructura de datos – Árboles. (2014). Recuperado de: <https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/>
- Estructuras de datos (2012) – Osvaldo Cairo. Recuperado de: [file:///C:/Users/alang/AppData/Local/Microsoft/Windows/INetCache/IE/TFOAL5KI/Estructuras%20de%20Datos%20Osvaldo%20Cair%C3%B3\[1\].pdf](file:///C:/Users/alang/AppData/Local/Microsoft/Windows/INetCache/IE/TFOAL5KI/Estructuras%20de%20Datos%20Osvaldo%20Cair%C3%B3[1].pdf)
- Gao, A. (2019). CS 486/686 Assignment 1 (121 marks in total). Recuperado de: https://cs.uwaterloo.ca/~a23gao/cs486686_s19/assignments/assign1.pdf
- Geeksforgeeks (2023) - backtracking-algorithms. Recuperado de: <https://www.geeksforgeeks.org/backtracking-algorithms/>

- Geeksforgeeks (2023) - difference-between-unsigned-int-and-signed-int-in-c. Recuperado de: <https://www.geeksforgeeks.org/difference-between-unsigned-int-and-signed-int-in-c/>
- Geeksforgeeks (2023) - size_t-data-type-c-language. Recuperado de: https://www.geeksforgeeks.org/size_t-data-type-c-language/
- Geeksforgeeks (2023) - std-find-in-cpp. Recuperado de: <https://www.geeksforgeeks.org/std-find-in-cpp/>
- Geeksforgeeks (2023) - unordered_map en C++ STL. Recuperado de: https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/
- Github (2023) - Apuntadores-Iteradores.pdf Recuperado de: <https://github.com/Chisrra/GALLOS/blob/main/G1/Presentaciones/008-Apuntadores-Iteradores.pdf>
- Github (2023) - maps-hash.pdf. Recuperado de: <https://github.com/Chisrra/GALLOS/blob/main/G1/Presentaciones/010-maps-hash.pdf>
- Gracie. (2023). *The Ultimate Guide To A Klotski Puzzle*. Recuperado de: <https://puzzlemechanics.com/the-ultimate-guide-to-a-klotski-puzzle/>
- Halim, S. Halim, F. (2013). *Competitive Programming 3*. Recuperado de: <https://www.inf.ufpr.br/andre/textos-CI1355-CI355/HalimHalim.pdf>
<https://codeforces.com/blog/entry/18898>
- Jjpeleato (2019) - algoritmia/backtracking. Recuperado de: <https://docs.jjpeleato.com/algoritmia/backtracking>
- Learn Microsoft (2023) - data-type-ranges. Recuperado de: <https://learn.microsoft.com/es-es/cpp/cpp/data-type-ranges?view=msvc-170>
- LearnC++ (2021) - why-you-should-know-about-brute-force-methods-in-c. Recuperado de: <https://learnplusplus.org/why-you-should-know-about-brute-force-methods-in-c/>
- Microsoft (2023) - cpp/goto-statement. Recuperado de: <https://learn.microsoft.com/es-es/cpp/cpp/goto-statement-cpp?view=msvc-170>
- Microsoft (2023) - file-types-created-for-visual-cpp-projects. Recuperado de: <https://learn.microsoft.com/es-es/cpp/build/reference/file-types-created-for-visual-cpp-projects?view=msvc-170>
- Microsoft. (2021). *SetConsoleMode function*. Recuperado de: <https://learn.microsoft.com/en-us/windows/console/setconsolemode>
- Microsoft. (2022). *Sleep function*. Recuperado de: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep>
- Microsoft. (2022). *Virtual Terminal Sequences*. Recuperado de: <https://learn.microsoft.com/en-us/windows/console/console-virtual-terminal-sequences>
- Microsoft. (2022.). *Console and Port I/O*. Recuperado de: <https://learn.microsoft.com/en-us/cpp/c-runtime-library/console-and-port-i-o?view=msvc-170&redirectedfrom=MSDN>
- Microsoft. (2023). *Operador condicional*. Recuperado de: <https://learn.microsoft.com/es-es/cpp/cpp/conditional-operator-q?view=msvc-170>
- Nandakumar. (2014). *Write Your Own conio.h*. Recuperado de: <https://www.opensourceforu.com/2014/03/write-conio-h-gnulinux/>

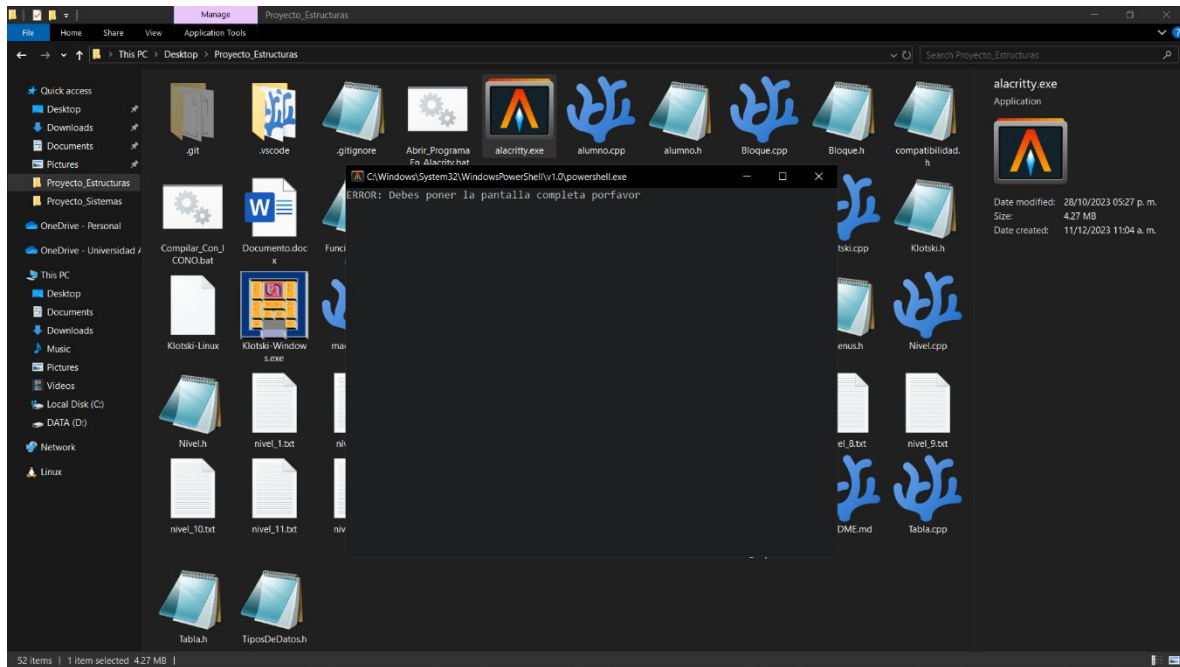
- Petersen, C. (2021). *ANSI Escape Sequences*. Recuperado de: <https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797>
- Programmerclick (2019) – Fuerza bruta. Recuperado de: <https://programmerclick.com/article/86801470856/>
- Sandroid. (2004). *Turbo C Entities*. Recuperado de: <https://www.sandroid.org/TurboC/functionlist.html#Entitites>
- Silisteanu, P. (2019). *ansi-escape-codes-windows-posix-terminals-c-programming-examples*. GitHub. Recuperado de: <https://github.com/sol-prog/ansi-escape-codes-windows-posix-terminals-c-programming-examples>
- Spaans, R. (2009). Solving sliding-block puzzles. Recuperado de: <https://www.pvv.ntnu.no/~spaans/spec-cs.pdf>
- Stackoverflow (2009) - how-do-i-add-an-icon-to-a-mingw-gcc-compiled-executable. Recuperado de: <https://stackoverflow.com/questions/708238/how-do-i-add-an-icon-to-a-mingw-gcc-compiled-executable>
- Stackoverflow (2016) – c why is boosthash combine the best way to combine hash values. Recuperado de: <https://stackoverflow.com/questions/35985960/c-why-is-boosthash-combine-the-best-way-to-combine-hash-values>
- Stackoverflow (2016) - what-is-an-unsigned-char. Recuperado de: <https://stackoverflow.com/questions/75191/what-is-an-unsigned-char>
- Stackoverflow (2018) - algoritmo-que-calcula-el-máximo-y-mínimo-de-un-vector-técnica-divide-y-vencerás. Recuperado de: <https://es.stackoverflow.com/questions/166047/algoritmo-que-calcula-el-m%C3%A1ximo-y-m%C3%Adnimo-de-un-vector-t%C3%A9cnica-divide-y-vencer%C3%A1s>
- Stackoverflow (2018) – how do i combine hash values in c0x. Recuperado de: <https://stackoverflow.com/questions/2590677/how-do-i-combine-hash-values-in-c0x>
- Stackoverflow (2019) – how stdunordered map is implemented. Recuperado de : <https://stackoverflow.com/questions/31112852/how-stdunordered-map-is-implemented>
- Stackoverflow (2019) - how-to-make-a-simple-c-makefile. Recuperado de: <https://stackoverflow.com/questions/2481269/how-to-make-a-simple-c-makefile>
- Stackoverflow (2021) - magic number in boosthash combine. Recuperado de: <https://stackoverflow.com/questions/4948780/magic-number-in-boosthash-combine>
- Techiedelight (2023) - depth-first-search. Recuperado de: <https://www.techiedelight.com/es/depth-first-search/>
- The Linux Programming Interface. (2023). *console_codes*. Recuperado de: https://man7.org/linux/man-pages/man4/console_codes.4.html
- unicode.org (2023). *The Unicode Standard, Version 15.1* . Recuperado de: <https://www.unicode.org/Public/UCD/latest/charts/CodeCharts.pdf>
- user2019716. (2011). *How can I use ANSI escape codes for outputting colored text in C and C++?* Stack Overflow. Recuperado de: <https://stackoverflow.com/questions/7414983/how-can-i-use-ansi-escape-codes-for-outputting-colored-text-in-c-and-c>

- Vidal, P. J. (2002). *UConio*. Recuperado de: <https://web.archive.org/web/20021016233403/http://crazylovetrain.hypermart.net/projects.htm>
- Wen, X., Zhi, L., & Zhen, R. (2015). *Huarong Dao Puzzle Solution (Search)*. Recuperado de: <https://wendy-xiao.github.io/files/huarongdao.pdf>
- Wikipedia (2021) - Búsqueda_en_profundidad. Recuperado de: https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad

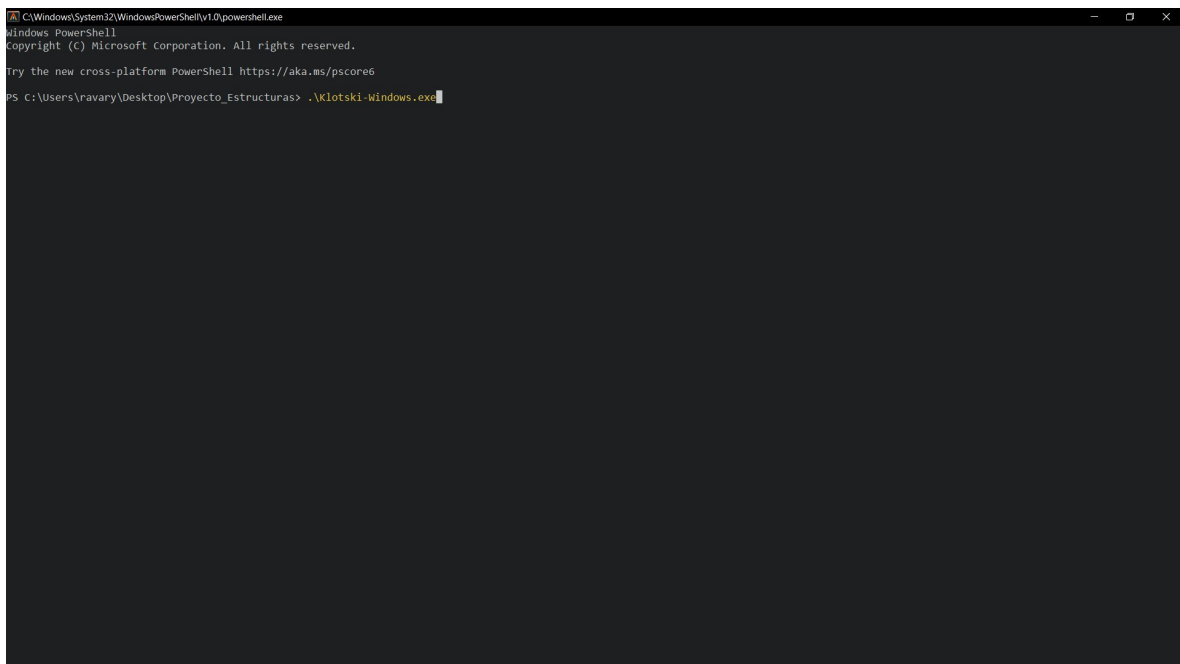
Manual de usuario

Para Ejecutar el programa se debe abrir el emulador de terminal recomendado (Alacrity ) , abrir la ruta especifica al programa y ejecutar el programa: "Klotski-Windows.exe" en el sistema operativo de Microsoft: Windows o si se está en un sistema operativo similar UNIX como lo seria Linux el programa será: "Klotski-Linux"

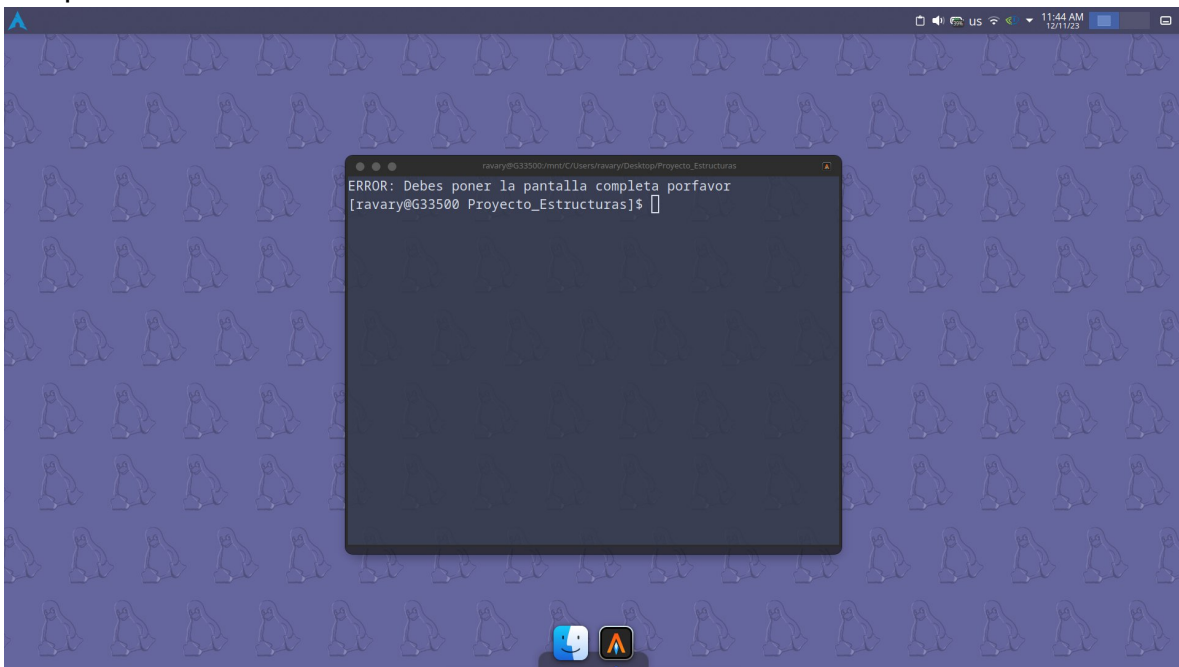
Si el emulador de terminal no se abre en pantalla completa mostrara un error:



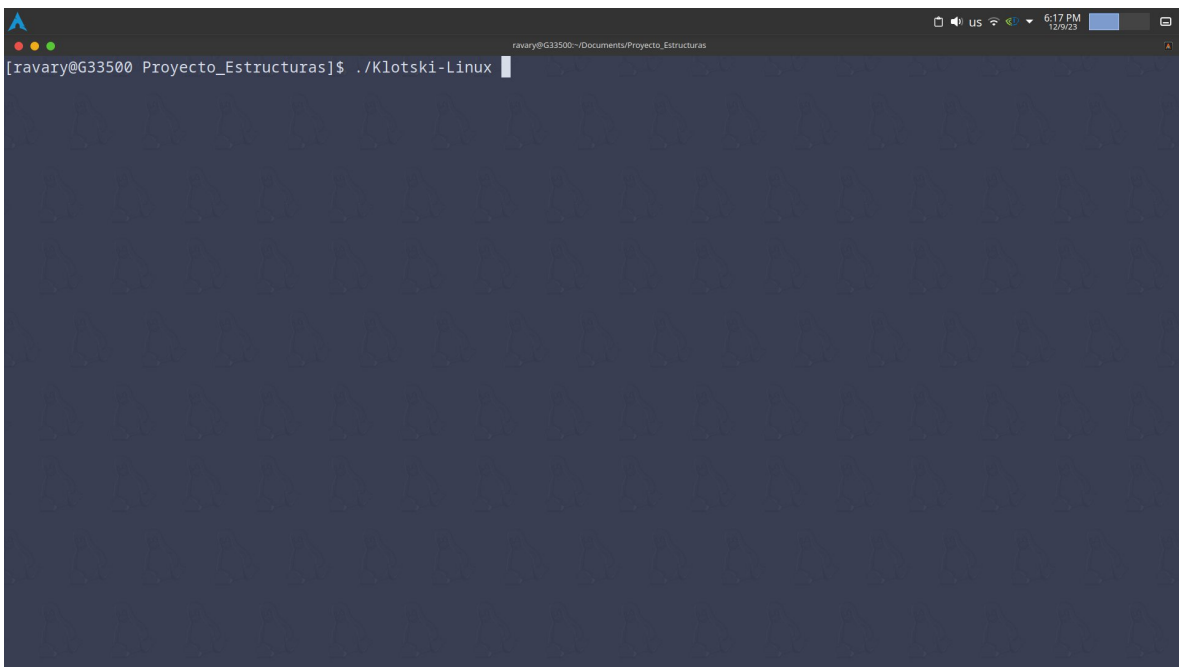
De lo contrario si se esta en pantalla completa el programa correra con normalidad.



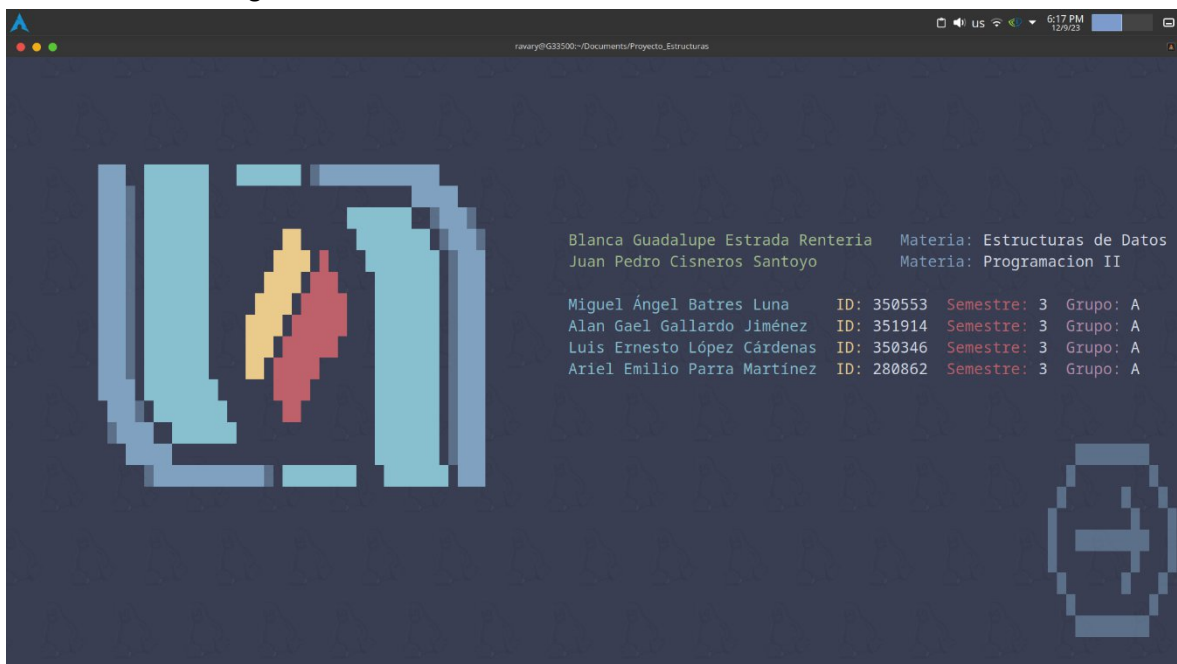
Igualmente, en Linux soltara un mensaje de error si no se está en pantalla completa



De lo contrario si se esta en pantalla completa el programa correra con normalidad.



Al Abrir el programa se ve la primera pantalla la cual muestra los nombres de los integrantes del equipo y los nombres de los profesores, para cambiar a la segunda pantalla se debe presionar la tecla “con la flecha derecha”, la letra “d” o la letra “D” de un teclado regular.



Al presionar alguna de estas teclas se ve el titulo del programa, para regresar a la primera pantalla se debe presionar la tecla “con la flecha izquierda”, la letra “a” o la letra “A” de un teclado regular; para ir a la tercera pantalla se debe presionar la tecla “con la flecha derecha”, la letra “d” o la letra “D” de un teclado regular.

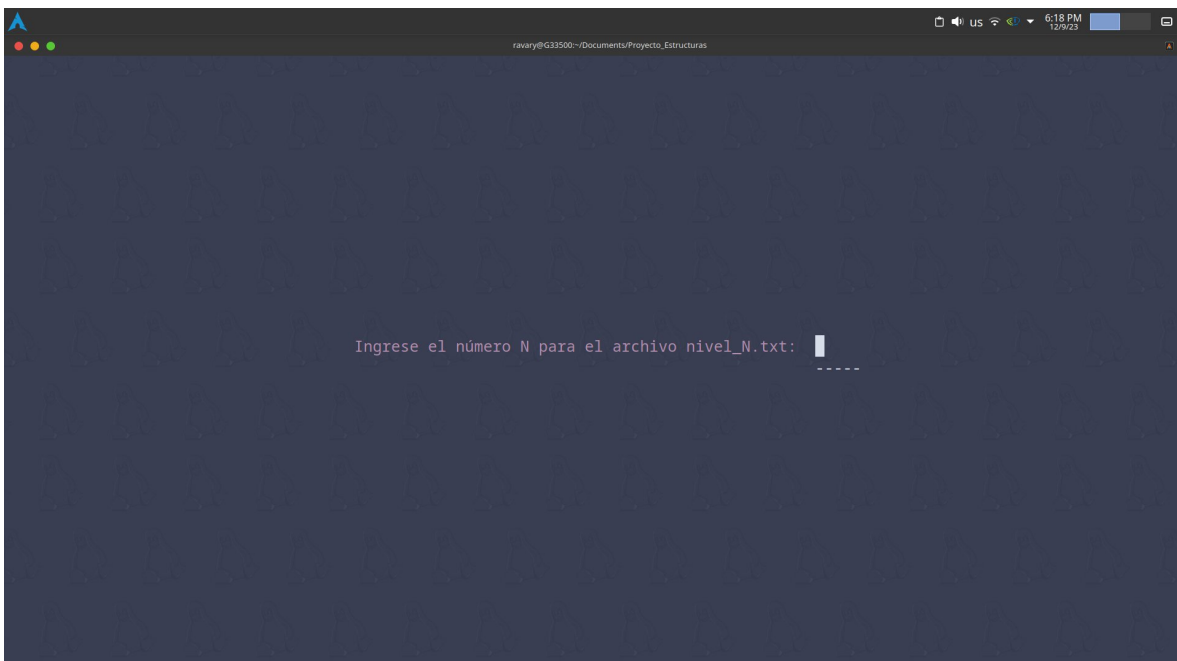


En la tercera pantalla tenemos la opción de regresarnos a la segunda pantalla con la tecla “con la flecha izquierda”, la letra “a” o la letra “A” de un teclado regular. Donde podemos navegar entre las cuatro opciones, hacia arriba con la tecla “con la flecha arriba”, con la tecla “w” o la tecla “W”, o hacia abajo con tecla “con la flecha abajo”, con la tecla “s” o la tecla “S”.



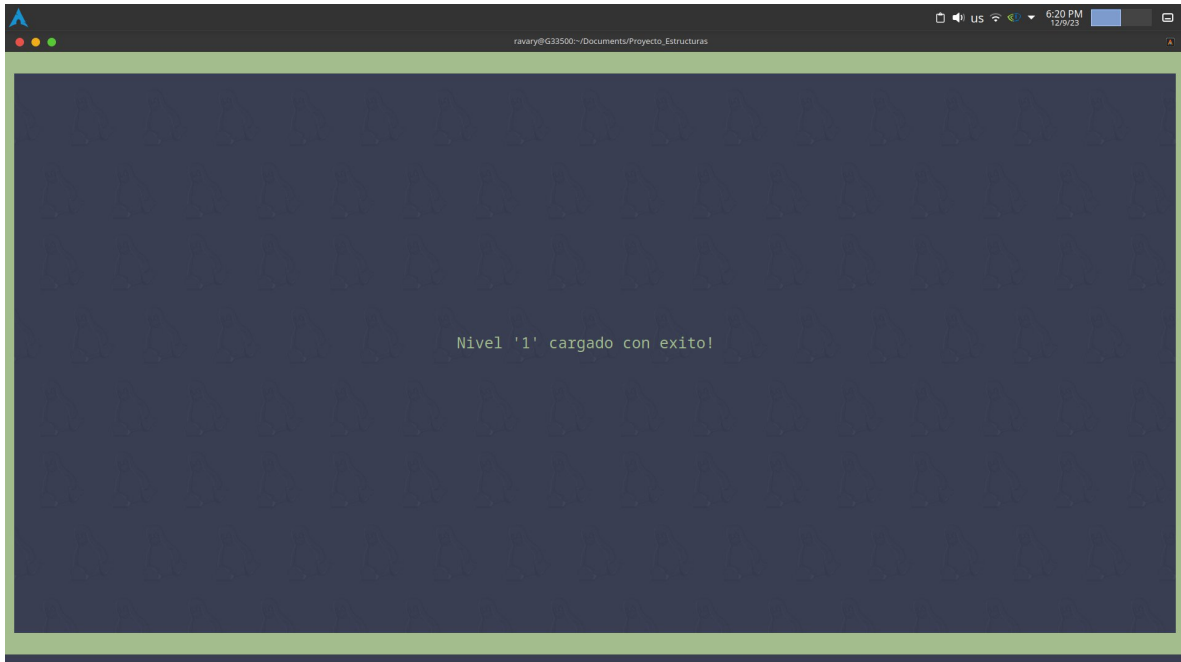
Pero también se cuenta con las opciones del programa, la primera opción en color cian es “CARGAR” con el cual podemos cargar los niveles de Klotski requeridos por el programa, para acceder a la opción presionamos la tecla “espaciadora” o la tecla que dice “ENTER” de un teclado regular.

Al entrar en la opción “CARGAR” se debe ingresar un número del 0 al 99999 y presionar la tecla “ENTER”, por ejemplo, si ingresamos el número 1 este representa

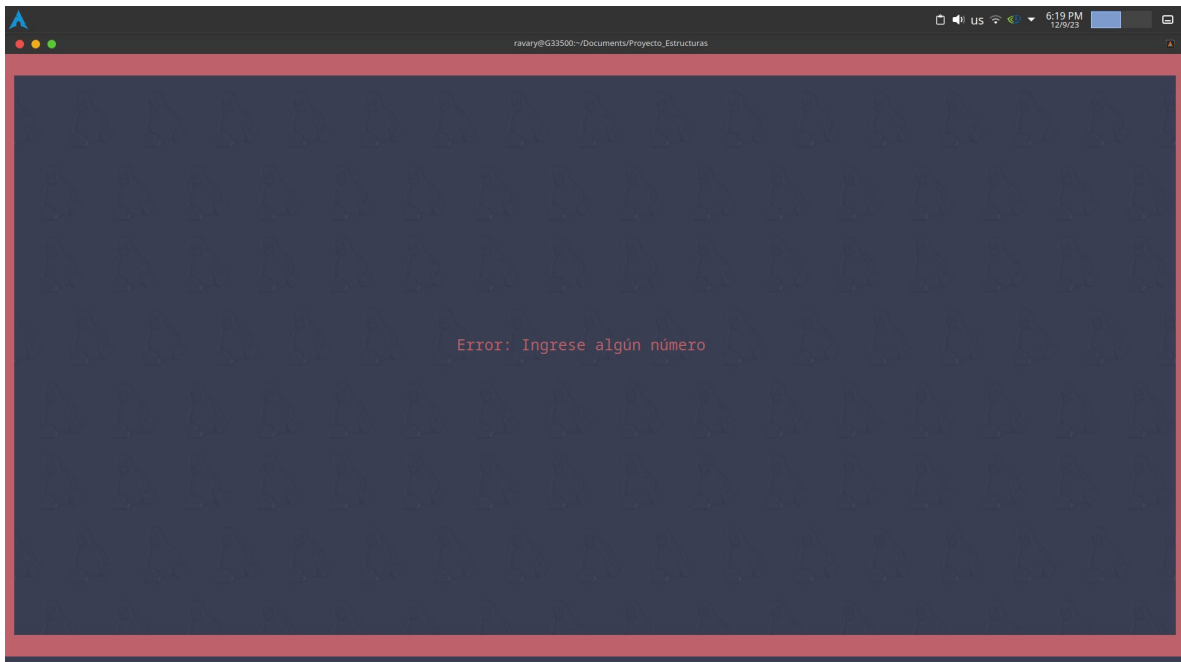


al archivo con el nombre de “nivel_1.txt” el cual contiene un título, las dimensiones del tablero y el tablero en cuestión.

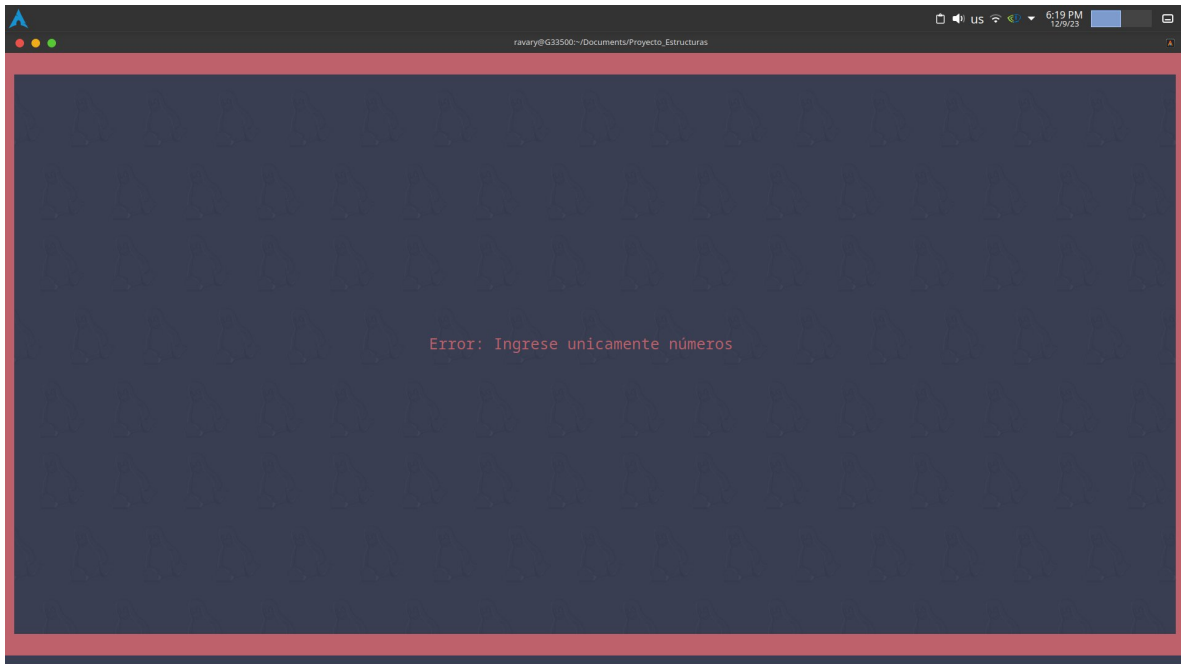
Si el nivel se cargó con éxito se mostrará una pantalla indicándolo, de lo contrario se mostrarán mensajes de error.



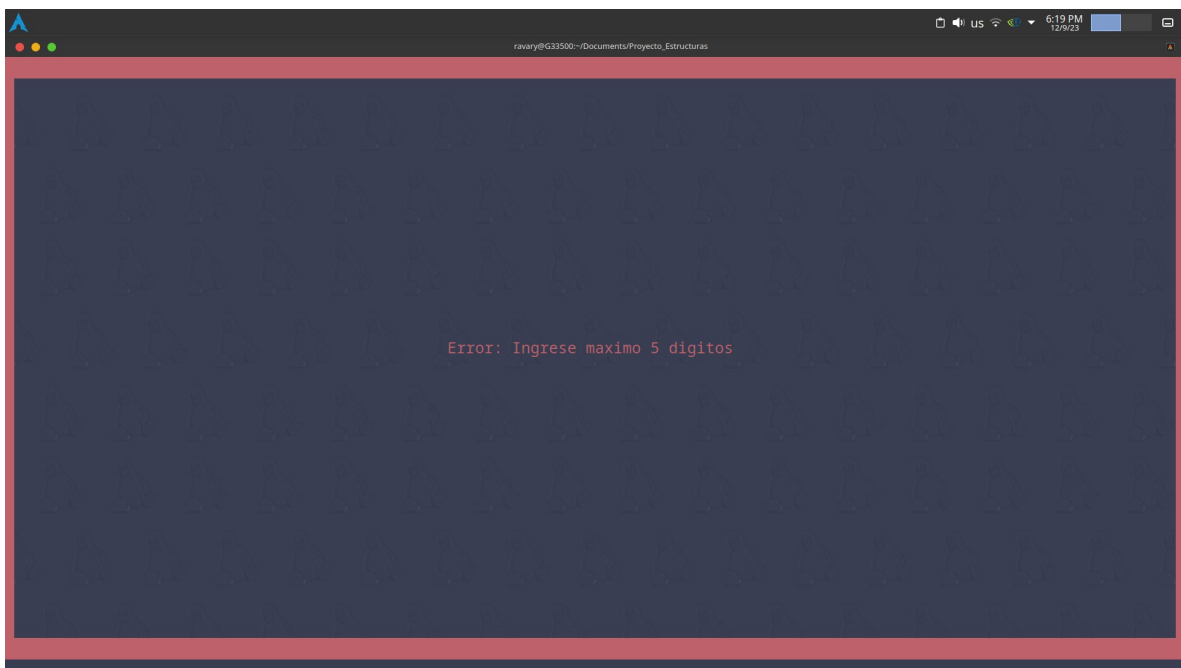
Si no se ingresa un número y solo se presiona la tecla “ENTER” dará este error:



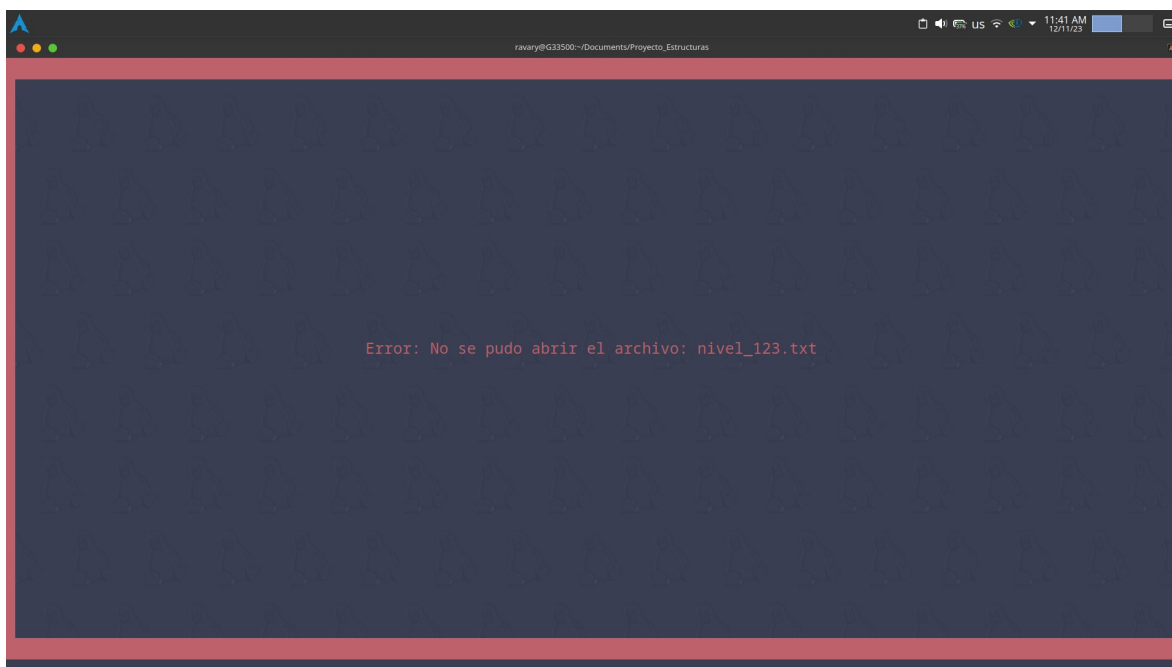
Si se ingresa algún carácter no numérico mostrara este error:



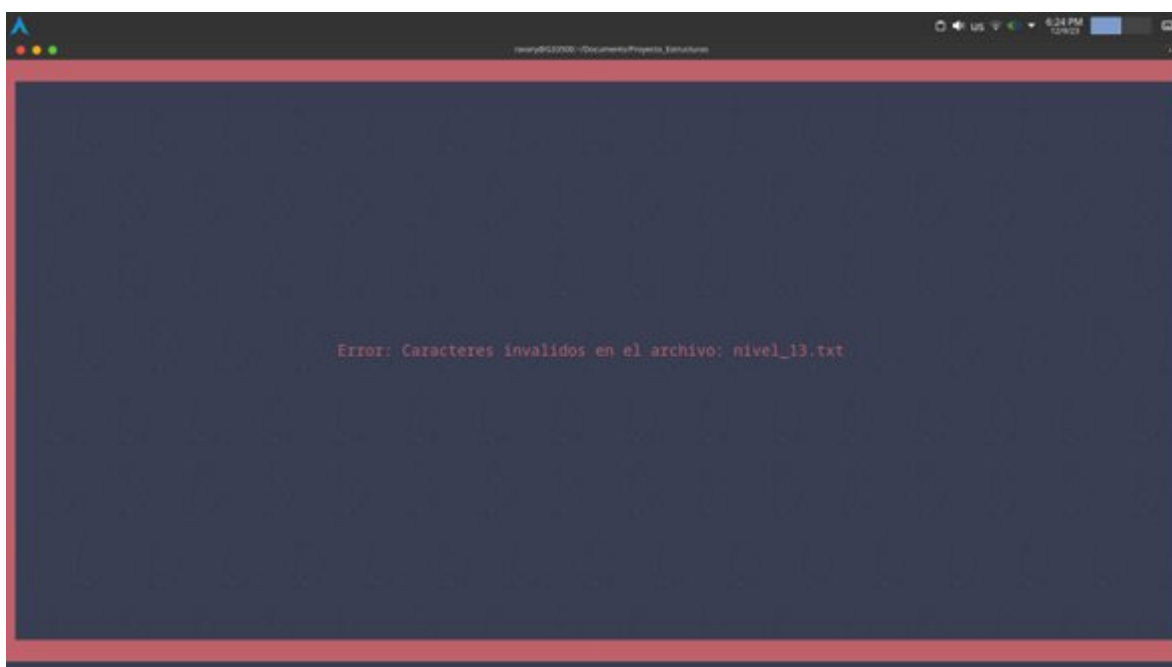
Si se ingresa un número mayor a 99999 mostrara este error:



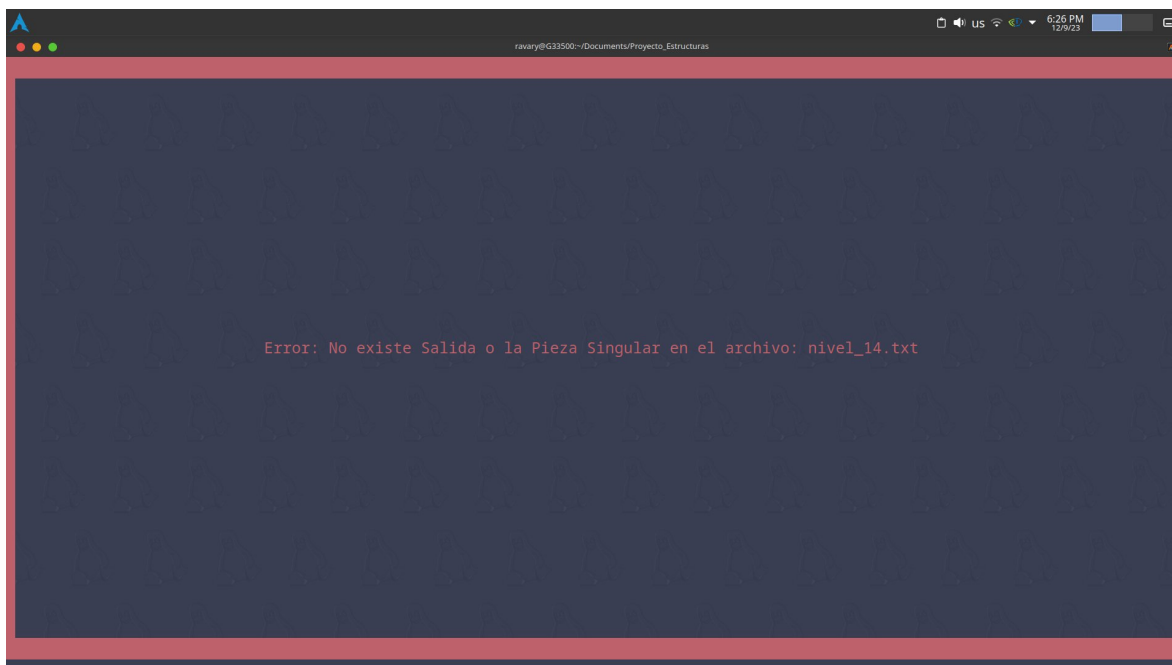
Si el archivo ingresado no existe o hubo un error de permisos para abrirlo, se mostrará este error con el nombre del archivo no existente



Y si el archivo contiene caracteres inválidos (todos los que no sean letras ASCII de la "a" a la "z" o estos caracteres " ", "&", ".", "*", "#") se mostrará este error:



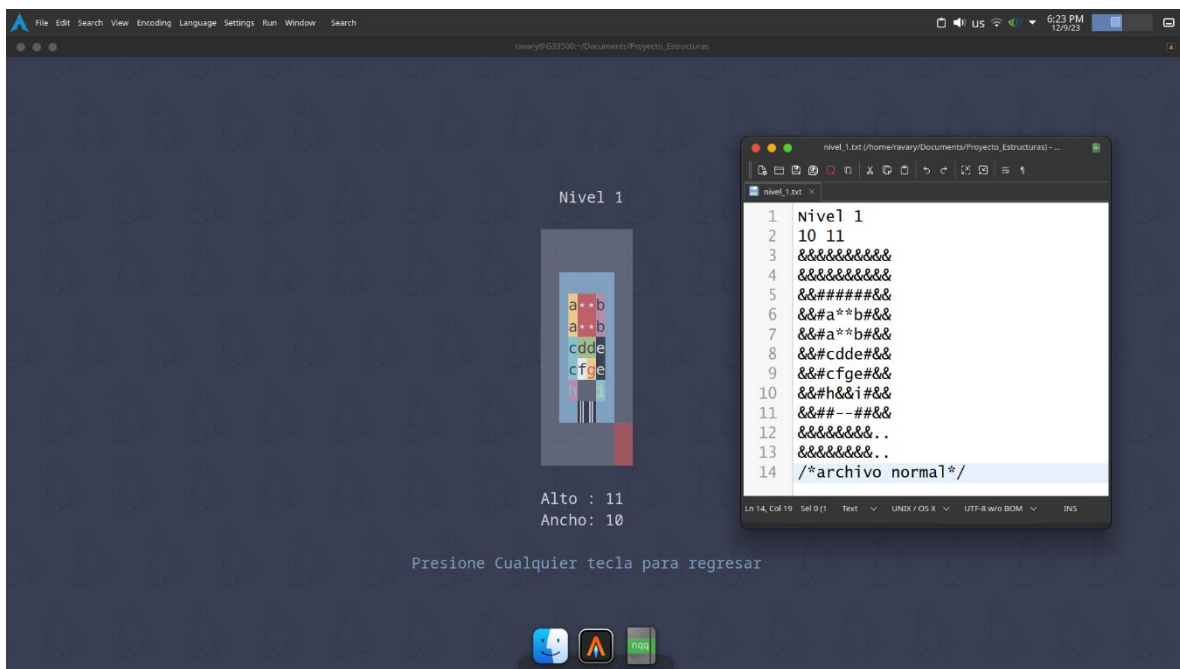
Por último, si no existe una Pieza singular o el Objetivo en el archivo, se mostrará este error:



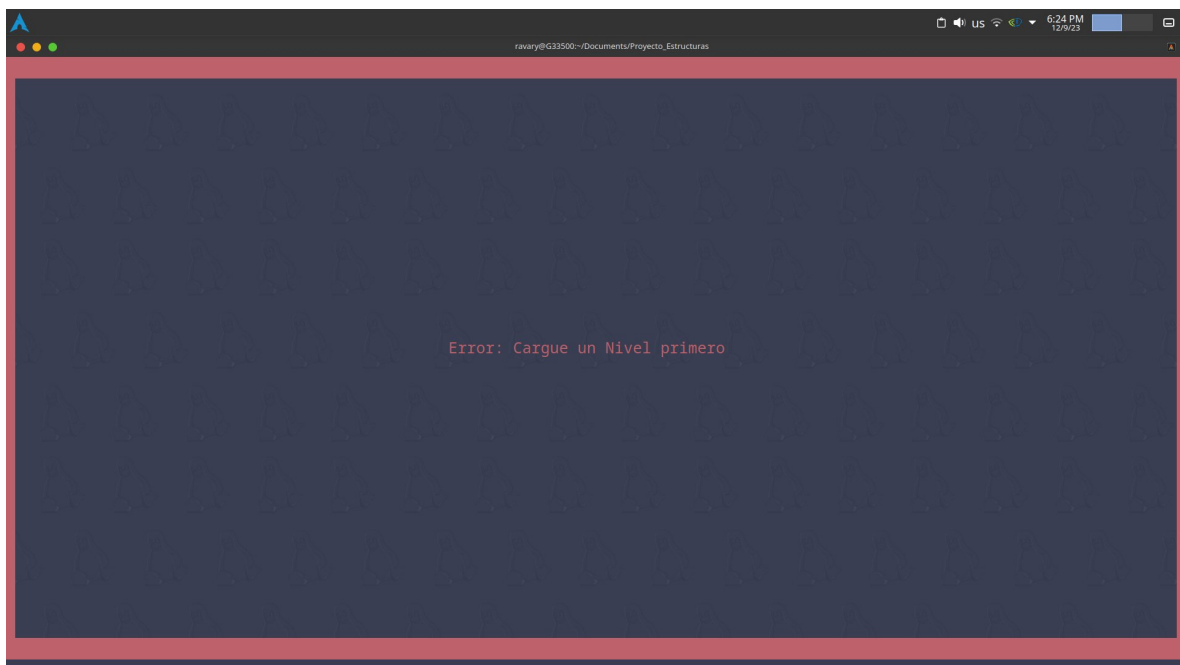
De vuelta el tercer Pantalla La segunda opción en color magenta es “TABLERO” con el cual podemos ver el Tablero de Klotski del Nivel elegido previamente.



para acceder a la opción presionamos la tecla “espaciadora” o la tecla que dice “ENTER” de un teclado regular.



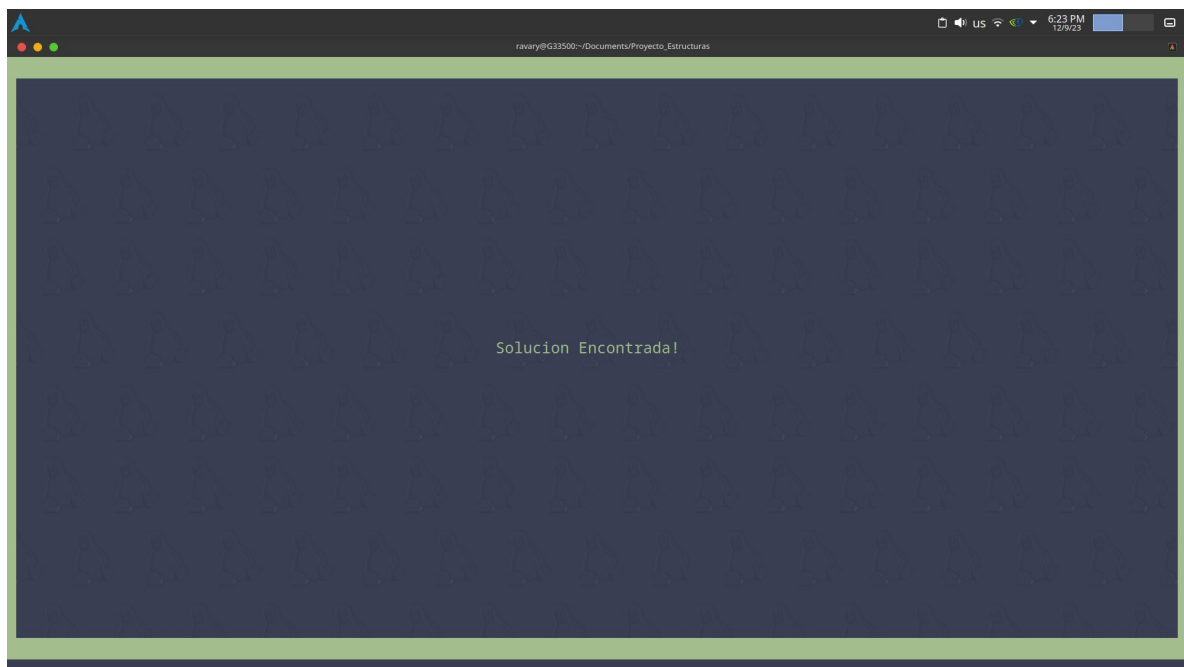
Si no se eligió anteriormente un nivel mostrara el siguiente mensaje de error:



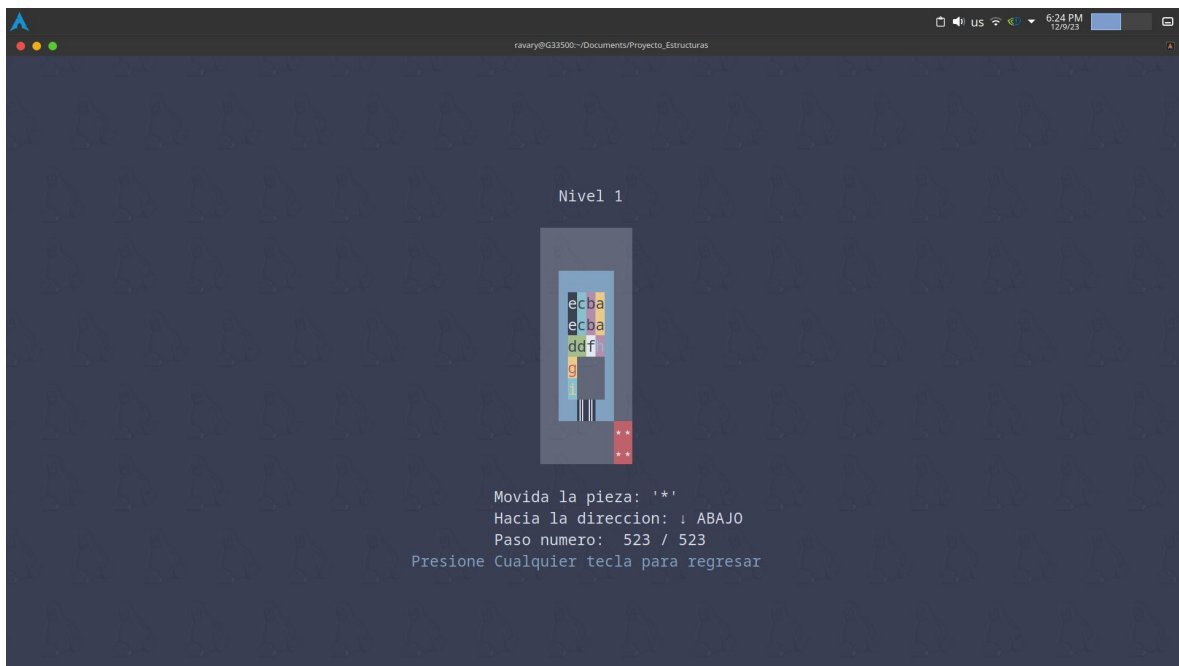
La tercera opción en color azul es “SOLUCION” donde se nos muestra los pasos para la solución con un lapso de 100ms aproximadamente por cada movimiento de la solución mientras se nos indica que movimiento se realizó y cuantos movimientos faltan para que la “Pieza Singular” llegue al “Objetivo”, para acceder a la opción presionamos la tecla “espaciadora” o la tecla que dice “ENTER” de un teclado regular.



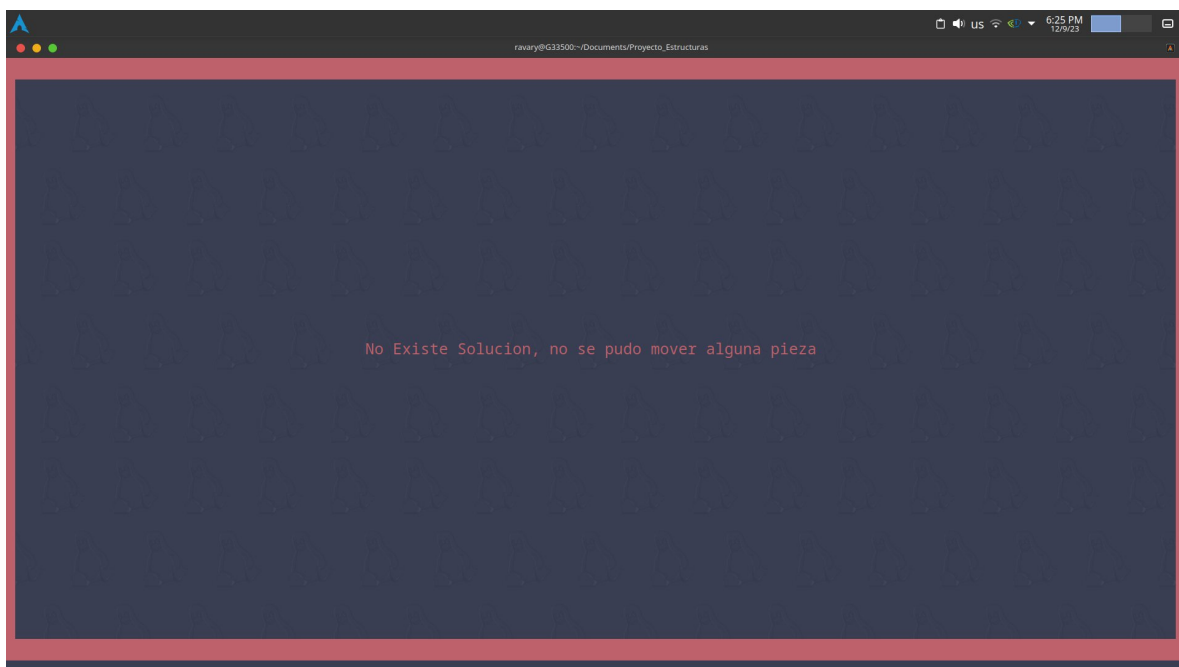
Si se encontro solucion se mostrara una pantalla indicándolo, de lo contrario se mostrarán dos posibles mensajes de error diciendo lo que ocurrió.



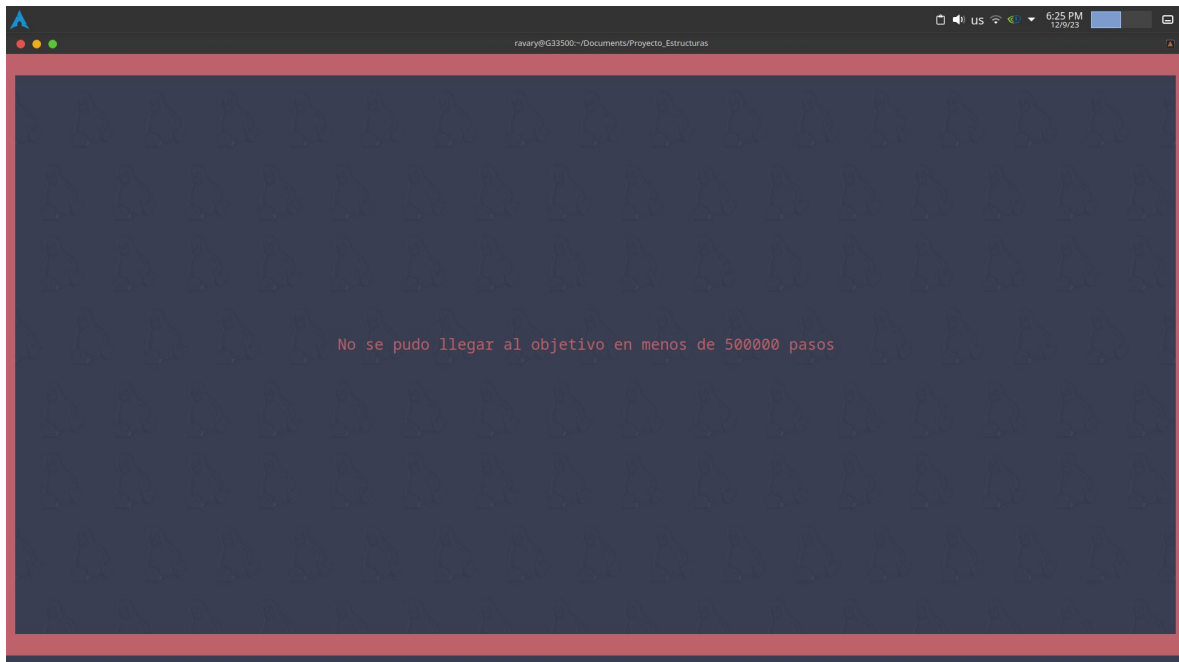
Si se encontró se mostrarán los pasos para la solución y al último se pide presionar alguna tecla para regresar a la tercer pantalla.



Si el tablero contenía un tablero sin solución ya que una pieza no se podía mover, se mostrará el siguiente error:



Si en el tablero el “Objetivo” o la “Pieza Singular” están bloqueados por “Paredes”; o si el tablero ocupa más de 500000 movimientos para su solución se mostrará el siguiente error:



Por último, la última opción de la tercera pantalla es “SALIR” con el cual se regresa el cursor a la terminal y se reinician los colores en pantalla, esta es la opción preferente para salir del programa de lo contrario se podría regresar a la terminal sin cursor y con colores modificados.

