JUEGO DE BUSCAR EL TESORO

DOCUMENTACION TECNICA

Presentación

La siguiente documentación ha sido desarrollada con la finalidad de dar a conocer la información necesaria para realizar el mantenimiento, instalación y exploración del software del **Juego de Buscar el Tesoro**, el cual consta de diferentes funcionalidades. El manual ofrece la información necesaria de ¿Cómo está desarrollado el software? Para que la persona (Desarrollador en C++) que quiera editar el software lo haga de una manera apropiada, dando a conocer la estructura del desarrollo del aplicativo.

Resumen

El manual detalla los aspectos técnicos e informativos del software del **Juego de Buscar el Tesoro** con la finalidad de explicar la estructura del aplicativo al personal que quiera administrarlo, editarlo o configurarlo. La siguiente guía se encuentra dividida en las herramientas que se usaron para el desarrollo del software con una breve explicación paso a paso. El aplicativo maneja diferentes funcionalidades los cuales se explicara que funcionamiento realiza cada uno de ellos, dando sugerencias para el debido uso del sistema informático.

Introducción

El manual se realiza con el fin de detallar el software para el **Juego de Buscar el Tesoro** en términos técnicos para aquella persona que vaya a administrar, editar o configurar el aplicativo lo haga de una manera apropiada. El documento se encuentra dividido en las siguientes secciones:

- ENTORNO DE DESARROLLO: Se darán a conocer las herramientas de desarrollo que se utilizaron para el software, así como se darán breves explicaciones de instalaciones y configuraciones necesarias de las herramientas anteriormente expuestas
- COMPLEJIDAD ALGORITMICA DE CADA ALGORITMO IMPLEMENTADO EN EL PROYECTO: Se describirá la complejidad algorítmica de los métodos que fueron implementados dentro del código fuente de la aplicación y que tienen una gran relevancia en Estructura De Datos
- DESCRIPCION DE LAS ESTRUCTURAS DE DATOS: Se explicaran las estructuras de datos que se utilizaron para el desarrollo de la aplicación

ENTORNO DE DESARROLLO

El aplicativo del **Juego de Scrabble** tiene la finalidad de simular el juego original de mesa llamado scrabble. Se recomienda que el siguiente manual sea manipulado únicamente por la persona que quiera administrar, editar o configurar el software para el **Juego de Scrabble** para velar por la seguridad de los datos que se almacenan. A continuación se presenta la instalación, configuración y descripción de las herramientas utilizadas para el desarrollo de la aplicación:

WSL (Windows Subsistem Linux)

- Descripción

Subsistema de Windows para Linux (WSL) es una característica de Windows que permite ejecutar un entorno Linux en la máquina de Windows, sin necesidad de una máquina virtual independiente ni de arranque dual. WSL está diseñado para proporcionar una experiencia perfecta y productiva para los desarrolladores que quieren usar Windows y Linux al mismo tiempo.

WSL 2 es el tipo de distribución predeterminada al instalar una distribución de Linux. WSL 2 usa tecnología de virtualización ligera. Las distribuciones de Linux se ejecutan como contenedores aislados dentro de la máquina virtual administradas de WSL 2. WSL 2 aumenta el rendimiento del sistema de archivos y agrega compatibilidad completa de llamadas del sistema en comparación con la arquitectura WSL 1.

- Instalación

Si tienes Linux o macOS no es necesario la instalación de WSL. Para la instalación de WSL en Windows desde la página oficial de Microsoft dirígete al siguiente enlace: https://learn.microsoft.com/es-es/windows/wsl/install en donde tendrás la guía completa de como descargar e instalar WSL en tu equipo de Windows.

- Configuración

Después de haber instalado WSL se tendrán que ejecutar los siguientes comandos:

sudo apt update: Con este comando se buscaran actualizaciones para los paquetes de la distribución de Linux instalada

sudo apt upgrade: Con este comando se aplicaran las actualizaciones (si los hay) de los paquetes que anteriormente se encontraron. Si en el comando anterior no se encontraron paquetes por actualizar no es necesario ejecutar este comando.

sudo apt install cmake gcc clang gdb build-essential: Con este comando se instalaran las herramientas, administradores, compiladores y debugers necesarios para el manejo de C++

CLion

- Descripción

IDE que viene de la mano de JetBrains una empresa de desarrollo de software bastante conocida por la creación de varias herramientas y lenguajes de programación Kotlin. CLion es un IDE enfocado para el desarrollo en los lenguajes de programación C y C++, CLion es un IDE multiplataforma por lo que puede ser utilizado en Linux, macOS y Windows integrado con el sistema de compilación CMake.

Además de C y C++, CLion admite otros lenguajes directamente o mediante complementos: Kotlin, Python, Rust, Swift y otros. CLion al igual que muchos IDE cuanta con la función de completar el código fácilmente, con lo cual puede ayudarte a ahorrar bastante tiempo en completar las sintaxis de tu código que estés escribiendo en él.

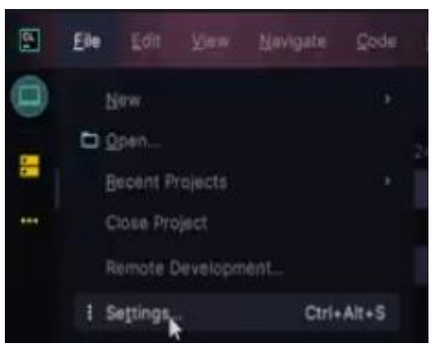
- Instalación

Para la instalación de CLion desde la página oficial de JetBrains dirígete al siguiente enlace: https://www.jetbrains.com/es-es/clion/ en donde podrás descargar CLion para su posterior instalación.

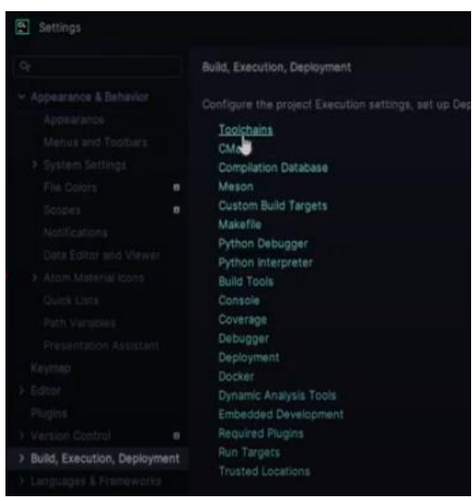
- Configuración

Si te encuentras en Linux o macOS, no es necesario hacer esta configuración. Si te encuentras en Windows debes hacer la siguiente configuración a CLion.

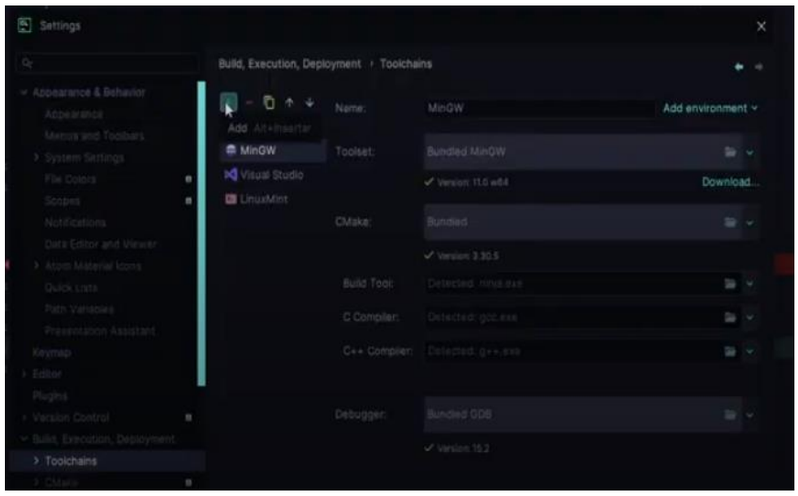
Dirígete a: Settings…:



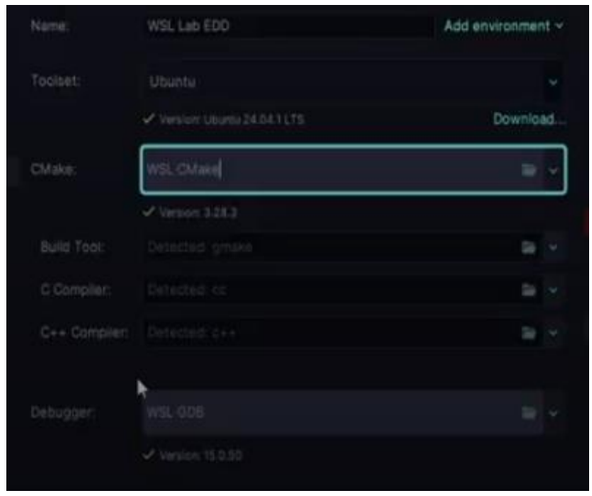
Dirígete a: Build, Execution, Deployment -> Toolchains



Luego en Add -> WSL:



De las siguientes configuraciones la más importante en tener es la de Toolset, en donde debes seleccionar la distribución de Linux que instalaste en WSL



Graphviz

- Descripción

Graphviz es un software de visualización de gráficos de código abierto. La visualización de gráficos es una forma de representar información estructural como diagramas de redes y gráficos abstractos. Tiene importantes aplicaciones en redes, bioinformática, ingeniería de software, diseño de base de datos y sitios web, aprendizaje automático y en interfaces visuales para otros dominios técnicos.

Los programas de diseño de Graphviz toman descripciones de gráficos en un lenguaje de texto simple y crean diagramas en formato útiles, como imágenes y SVG para páginas web; PDF Postscript para incluir en otros documentos; o visualizar en un navegador concreto, como opción de colores, fuentes, diseños de nodos tabulares, estilos de líneas, hipervínculos y formas personalizadas.

- Instalación

Para los que instalaron WSL, es recomendable que instalen Graphviz en los dos sistemas operativos por si acaso uno de los dos fallara (no debería) tener el otro de reserva, de igual forma este software es ligero en descarga e instalación.

Linux: ejecutar los siguientes comandos en la consola/terminal de WSL o Linux directamente.

sudo apt update: Con este comando se buscaran actualizaciones para los paquetes de la distribución de Linux instalada

sudo apt upgrade: Con este comando se aplicaran las actualizaciones (si los hay) de los paquetes que anteriormente se encontraron. Si en el comando anterior no se encontraron paquetes por actualizar no es necesario ejecutar este comando.

sudo apt install graphviz: Este comando instalara todo lo necesario para tener Graphviz

Windows: Para la instalación de Graphviz desde su página oficial dirígete al siguiente enlace: https://graphviz.org/download/ en donde podrás descargar el ejecutable para Windows y así hacer su instalación.

COMPLEJIDAD ALGORITMICA DE CADA ALGORITMO IMPLEMENTADO EN EL PROYECTO

A continuación se presenta la estructuración del proyecto junto con el análisis algorítmico de cada algoritmo implementado en el proyecto:

- Utilities.cpp

```
void Utilities::verifyNumericEntry(int &value, std::string message) {
    std::string input;
    while (true) {
        try {
            std::cout << message;
            getline([&]std::cin, [&]input);
            std::stringstream ss(input);
            if (!(ss >> value) || !(ss.eof())) {
                throw std::runtime_error("Debe Ingresar un Valor Numerico");
            }
            break;
        } catch (const std::exception &e) {
            std::cout <<">> Error!!!: " << e.what() << std::endl;
        }
    }
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)

TOTAL: O(1)

- Structs
  - List
    - LinkedList.h

```
NodeList<T> *getElementAt(int index) {
    if (index < 0 || index >= this->size) return nullptr;

    if (index == 0) return this->head;

    int counter = 0;
    NodeList<T> *current = this->head;
    while (counter < index) {
        current = current->getNext();
        counter++;
    }
    return current;
}
```

O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)

TOTAL: O(n)

```
void addElementAt(T *element) {
    auto *newNode = new NodeList<T>(element);
    if (this->isEmpty()) {
        this->head = newNode;
        ++this->size;
        return;
    }
    NodeList<T> *current = this->head;
    while (current->getNext() != nullptr) current = current->getNext();

    current->setNext(newNode);
    ++this->size;
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)

O(1)
O(1)
TOTAL: O(n)

```
T *removeElementAt(int index) {
    NodeList<T> *current = this->head;
    NodeList<T> *prev = nullptr;
    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->getNext();
    }
    if (prev != nullptr) {
        prev->setNext(current->getNext());
    } else {
        this->head = current->getNext();
    }
    current->setNext(nullptr);
    --this->size;
    return current->getData();
}
```

O(1)
O(1)
O(n)
O(1)
O(1)

O(1)
O(1)

O(1)

O(1)
O(1)
O(1)
TOTAL: O(n)

```
T *removeElements(T *element) {
    NodeList<T> *current = this->head;
    NodeList<T> *prev = nullptr;
    while (current != nullptr && current->getData() != element) {
        prev = current;
        current = current->getNext();
    }
    if (current == nullptr) return nullptr;

    if (prev != nullptr) {
        prev->setNext(current->getNext());
    } else {
        this->head = current->getNext();
    }
    current->setNext(nullptr);
    --this->size;
    return current->getData();
}
```

O(1)
O(1)
O(n)
O(1)
O(1)

O(1)

O(1)
O(1)

O(1)

O(1)
O(1)
O(1)
TOTAL: O(n)

- NodeList.h
- Matrix
  - NodeMatrix.h
  - OrthogonalMatrix.h

```
void createOrthogonalMatrix() {
    NodeMatrix<T> *aux = this->root;
    for (int i = 0; i < this->dimensionX - 1; i++) {
        this->connectNodes(aux, i);
        aux = this->getNode( x: i, y: 0, z: 0);
        auto path = new Path();
        path->setImage( " ");
        auto *newNode = new NodeMatrix<T>(path, x: i + 1, y: 0, z: 0);
        aux->setBottom(newNode);
        newNode->setTop(aux);
        aux = newNode;
        this->connectNodes(aux, i + 1);
    }
    std::cout << "TABLERO TRIDIMENSIONAL CREADO CON EXITO!!!" << std::endl;
}
```

O(1)
O(x)
O(y * z)
O(x + 0 + 0)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(y * z)

O(1)
TOTAL: O(x*y*z)

```
void connectNodes(NodeMatrix<T> *aux, int i, int j) {
    for (int k = 0; k < this->dimensionZ - 1; k++) {
        auto path = new Path();
        path->setImage( " ");
        auto *newNode = new NodeMatrix<T>(path, x: i, y: j, z: k + 1);
        aux->setBack(newNode);
        newNode->setFront(aux);
        if (j > 0) {
            newNode->setPrev(aux->getPrev()->getBack());
            aux->getPrev()->getBack()->setNext(newNode);
        }
        if (i > 0) {
            newNode->setTop(aux->getTop()->getBack());
            aux->getTop()->getBack()->setBottom(newNode);
        }
        aux = newNode;
    }
}
```

O(z)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)

O(1)
O(1)
O(1)

O(1)

TOTAL: O(z)

```
void connectNodes(NodeMatrix<T> *aux, int i) {
    for (int j = 0; j < this->dimensionY - 1; j++) {
        this->connectNodes(aux, i, j);
        aux = this->getNode( x: i, y: j, z: 0);
        auto path = new Path();
        path->setImage( " ");
        auto *newNode = new NodeMatrix<T>(path, x: i, y: j + 1, z: 0);
        aux->setNext(newNode);
        newNode->setPrev(aux);
        if (i > 0) {
            newNode->setTop(aux->getTop()->getNext());
            aux->getTop()->getNext()->setBottom(newNode);
        }
        aux = newNode;
        this->connectNodes(aux, i, j + 1);
    }
}
```

O(y)
O(z)
O(x + y + 0)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(z)

TOTAL: O(y * z)

```
NodeMatrix<T> *getNode(const int x, const int y, const int z) {
    NodeMatrix<T> *aux = this->root;
    for (int i = 0; i < x; ++i) aux = aux->getBottom();

    for (int i = 0; i < y; ++i) aux = aux->getNext();

    for (int i = 0; i < z; ++i) aux = aux->getBack();

    return aux;
}
```

O(1)
O(x)

O(y)

O(z)

O(1)

TOTAL: O(x + y + z)

- Tree
  - NodeTree.h

- TreeBB.h

```
void insert(NodeTree<T> *value, NodeTree<T> *&root, const bool isInverse) {
    if (root == nullptr) {                                          O(1)
        root = value;                                               O(1)
        return;                                                     O(1)
    }
    if (isInverse) {    // ARBOL DE TRAMPAS                         O(1)
        if (value->getSize() < root->getSize()) {                   O(1)
            insert(value, root: [&]root->getRight2(), isInverse);   O(n)
        } else {
            insert(value, root: [&]root->getLeft2(), isInverse);    O(n)
        }
    } else {    // ARBOL DE ENEMIGOS
        if (value->getSize() < root->getSize()) {                   O(1)
            insert(value, root: [&]root->getLeft2(), isInverse);    O(n)
        } else {
            insert(value, root: [&]root->getRight2(), isInverse);   O(n)
        }
    }
}
```

TOTAL: O(n)

```
void remove(NodeTree<T> *value, NodeTree<T> *&root) {
    if (value->getSize() == root->getSize()) {                      O(1)
        if (this->isLeaf(root)) {                                   O(1)
            root = nullptr;                                         O(1)
            return;                                                 O(1)
        }
        if (root->getLeft() == nullptr) {                           O(1)
            root = root->getRight();                                O(1)
            return;                                                 O(1)
        }
        if (root->getRight() == nullptr) {                          O(1)
            root = root->getLeft();                                 O(1)
            return;                                                 O(1)
        }
        root->getRight()->setLeft(root->getLeft()->getRight());     O(1)
        root->getLeft()->setRight(root->getRight());                O(1)
        root = root->getLeft();                                     O(1)
        return;                                                     O(1)
    }
    if (value->getSize() < root->getSize()) {                       O(1)
        remove(value, root: [&]root->getLeft());                    O(n)
        return;                                                     O(1)
    }
    if (value->getSize() >= root->getSize()) {                      O(1)
        remove(value, root: [&]root->getRight());                   O(n)
    }
}
```

TOTAL: O(n)

```
T *search(NodeTree<T> *root, int size, const bool isInverse) {
    if (root == nullptr) {
        return nullptr;
    }
    if (size == root->getSize()) {
        return root->getData();
    }
    if (isInverse) {    // ARBOL DE TRAMPAS
        if (size < root->getSize()) {
            return search( root: root->getRight(), size, isInverse);
        }
        if (size >= root->getSize()) {
            return search( root: root->getLeft(), size, isInverse);
        }
    } else {    // ARBOL DE ENEMIGOS
        if (size < root->getSize()) {
            return search( root: root->getLeft(), size, isInverse);
        }
        if (size >= root->getSize()) {
            return search( root: root->getRight(), size, isInverse);
        }
    }
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O()
O(1)
O(n)
O(1)
O(n)

TOTAL: O(n)

```
bool isLeaf(NodeTree<T> *node) {
    return node->getLeft() == nullptr && node->getRight() == nullptr;
}
```

O(1)

TOTAL: O(1)

- Controllers
  - GameController.cpp

```
NodeMatrix<Object> *GameController::verifyMovement(const char movement) {
    switch (movement) {
        case 'W': { return this->board->getPlayerNode()->getTop(); }
        case 'S': { return this->board->getPlayerNode()->getBottom(); }
        case 'A': { return this->board->getPlayerNode()->getPrev(); }
        case 'D': { return this->board->getPlayerNode()->getNext(); }
        case 'Q': { return this->board->getPlayerNode()->getBack(); }
        case 'E': { return this->board->getPlayerNode()->getFront(); }
        default: { return nullptr; }
    }
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)

TOTAL: O(1)

```
void GameController::gameReport() {
    std::cout << "\n--- Reporte de la Partida Actual ---" << std::endl;
    std::cout << "Nombre del Jugador: " << this->currentGame->getPlayerName() << std::endl;
    std::cout << "Tiempo Total de la Partida: " << this->currentGame->getTime() / 1000 << "s" << std::endl;
    std::cout << "Total de Movimientos: " << this->currentGame->getMovements() << std::endl;
    std::cout << "Puntuacion: " << this->currentGame->getScore() << std::endl;
}
```

O(1)
O(1)
O(1)
O(1)
O(1)

TOTAL: O(1)

- HistoryController.cpp

```
LinkedList<Game> *HistoryController::loadHistory(std::string &fileName) {
    std::ifstream file(fileName);
    if (!file.is_open()) {
        std::cerr << "Error al abrir el archivo en la ruta: " << fileName << std::endl;
        return nullptr;
    }
    std::cout << "Archivo Cargado Correctamente" << std::endl;
    auto history = new LinkedList<Game>();
    std::string line;
    while (std::getline( [&] file,  [&] line)) {
        std::istringstream iss(line);
        std::string playerName;
        int score;
        int movements;
        if (getline( [&] iss,  [&] playerName,  delim: ',') && iss >> score && iss.ignore() && iss >> movements) {
            auto game = new Game( playerName, score,  time: 0, movements);
            history->addElementAt(game);
        } else {
            std::cout << "Error al extraer los datos de la linea: " << line << std::endl;
        }
    }
    file.close();
    std::cout << "Archivo Leido!!!" << std::endl;
    std::cout << "Se registraron: " << history->getSize() << " partidas\n" << std::endl;
    return history;
}
```

O(1)
O(1)
O(1)
O(1)

O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)

O(1)

O(1)
O(1)
O(1)
O(1)

TOTAL: O(n)

- ReportsController.cpp

```
void ReportsController::treasureLocationReport(NodeMatrix<Object> *treasureNode, LinkedList<Movement> *playerMovements) {
    std::cout << "--- Ubicacion del Tesoro ---" << std::endl;
    std::cout << "Coordenada X: " << treasureNode->getX() << std::endl;
    std::cout << "Coordenada Y: " << treasureNode->getY() << std::endl;
    std::cout << "Coordenada Z: " << treasureNode->getZ() << std::endl;
    std::cout << "--- Trayectoria del Jugador ---" << std::endl;
    NodeList<Movement> *aux = playerMovements->getHead();
    int i = 1;
    while (aux != nullptr) {
        std::cout << "-> Movimiento: " << i << std::endl;
        std::cout << aux->getData()->convertToString() << std::endl;
        aux = aux->getNext();
        i++;
    }
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)

TOTAL: O(n)

```
void ReportsController::reportEnemiesAndTraps(TreeBB<Enemy> *enemiesTree, TreeBB<Trap> *trapsTree) {
    std::cout << "--- Enemigos Enfrentados ---" << std::endl;
    this->traversePreOrden( nodeEnemy: enemiesTree->getRoot());
    std::cout << "--- Trampas Activadas ---" << std::endl;
    this->traversePreOrden( nodeTrap: trapsTree->getRoot());
}
```

O(1)
O(n)
O(1)
O(n)

TOTAL: O(n)

```
void ReportsController::printLeaderBoard(LinkedList<Game> *gamesHistory) {
    NodeList<Game> *aux = gamesHistory->getHead();
    int i = 1;
    while (aux != nullptr) {
        std::cout << i << ". " << aux->getData()->getPlayerName() << " -> " << aux->getData()->getScore() << " puntos, "
        << aux->getData()->getMovements() << " movimientos -> " << aux->getData()->getTime() / 1000 << " s" << std::endl;
        aux = aux->getNext();
        i++;
    }
}
```

O(1)
O(1)
O(n)
O(1)

O(1)
O(1)

TOTAL: O(n)

- Models
  - Objects
    - Enemy.cpp
    - Object.cpp
    - Path.cpp

- Potion.cpp
- Track.cpp
- Trap.cpp
- Treasure.cpp

- Board.cpp

```
void Board::createBoard(Player *player, TreeBB<Enemy> *enemiesTree, TreeBB<Trap> *trapsTree) {
    this->putTreasure();                                                    O(x + y + z)
    this->putPlayer(player);                                                O(x + y + z)
    srand( seed: time( timer: nullptr));                                    O(1)
    for (int i = 0; i < this->totalNodes - 2; i++) {                        O(n)
        int random = rand() % 100;                                          O(1)
        if (random < 15) {                                                  O(1)
            this->putEnemies(enemiesTree);                                  O(x + y + z)
        } else if (random < 30) {                                           O(1)
            this->putTraps(trapsTree);                                      O(x + y + z)
        } else if (random < 45) {                                           O(1)
            this->putPotions();                                             O(x + y + z)
        } else if (random < 60) {                                           O(1)
            this->putTracks();                                              O(x + y + z)
        }   // else SE QUEDA EL PATH LIBRE
    }
    std::cout << "Enemigos Colocados en el Mapa: " << this->totalEnemies << std::endl;   O(1)
    std::cout << "Trampas Colocadas en el Mapa: " << this->totalTraps << std::endl;      O(1)
    std::cout << "Pociones Colocadas en el Mapa: " << this->totalPotions << std::endl;   O(1)
    std::cout << "Pistas Colocadas en el Mapa: " << this->totalTracks << std::endl;      O(1)
}                                                                           TOTAL: O(n^4)
```

```
void Board::putTreasure() {
    srand( seed: time( timer: nullptr));                                    O(1)
    NodeMatrix<Object> *treasureNode;                                       O(1)
    do {
        int x = rand() % this->orthogonalMatrix->getDimensionX();           O(1)
        int y = rand() % this->orthogonalMatrix->getDimensionY();           O(1)
        int z = rand() % this->orthogonalMatrix->getDimensionZ();           O(1)
        treasureNode = this->orthogonalMatrix->getNode(x, y, z);            O(x + y + z)
    } while (treasureNode->getData() == nullptr                             O(1)
        || dynamic_cast<Path*>(treasureNode->getData()) == nullptr);
    delete treasureNode->getData();                                         O(1)
    auto *treasure = new Treasure();                                        O(1)
    treasure->setImage( path: "$");                                         O(1)
    treasureNode->setData(treasure);                                        O(1)
    this->treasureNode = treasureNode;                                      O(1)
    std::cout << "Tesoro Colocado en el Mapa" << std::endl;                 O(1)
}                                                                           TOTAL: O(x + y + z)
```

```
void Board::putPlayer(Player *player) {
    srand( seed: time( timer: nullptr));
    NodeMatrix<Object> *playerNode;
    do {
        int x = rand() % this->orthogonalMatrix->getDimensionX();
        int y = rand() % this->orthogonalMatrix->getDimensionY();
        int z = rand() % this->orthogonalMatrix->getDimensionZ();
        playerNode = this->orthogonalMatrix->getNode(x, y, z);
    } while (playerNode->getData() == nullptr
        || dynamic_cast<Path*>(playerNode->getData()) == nullptr);
    this->dataBelowThePlayer = playerNode->getData();
    playerNode->setData(player);
    this->playerNode = playerNode;
    std::cout << "Jugador Colocado en el Mapa" << std::endl;
}
```

O(1)
O(1)

O(1)
O(1)
O(1)
O(x + y + z)
O(1)

O(1)
O(1)
O(1)
O(1)
TOTAL: O(x + y + z)

```
void Board::putEnemies(TreeBB<Enemy> *enemiesTree) {
    srand( seed: time( timer: nullptr));
    NodeMatrix<Object> *enemyNode;
    int x, y, z;
    do {
        x = rand() % this->orthogonalMatrix->getDimensionX();
        y = rand() % this->orthogonalMatrix->getDimensionY();
        z = rand() % this->orthogonalMatrix->getDimensionZ();
        enemyNode = this->orthogonalMatrix->getNode(x, y, z);
    } while (enemyNode->getData() == nullptr
        || dynamic_cast<Path*>(enemyNode->getData()) == nullptr);
    delete enemyNode->getData();
    int level = ((x + 1) * 100) + ((y + 1) * 10) + (z + 1);
    int damage = (rand() % 15) + 1;
    auto *enemy = new Enemy(level, damage);
    enemy->setImage( ∴ "!");
    enemyNode->setData(enemy);
    auto *enemyNodeTree = new NodeTree(enemy, level);
    enemiesTree->insert(enemyNodeTree, isInverse: false);
    this->totalEnemies++;
}
```

O(1)
O(1)
O(1)

O(1)
O(1)
O(1)
O(x + y + z)
O(1)

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
TOTAL: O(x + y + z)

```
void Board::putTraps(TreeBB<Trap> *trapsTree) {
    srand( seed: time( timer: nullptr));
    NodeMatrix<Object> *trapNode;
    int x, y, z;
    do {
        x = rand() % this->orthogonalMatrix->getDimensionX();
        y = rand() % this->orthogonalMatrix->getDimensionY();
        z = rand() % this->orthogonalMatrix->getDimensionZ();
        trapNode = this->orthogonalMatrix->getNode(x, y, z);
    } while (trapNode->getData() == nullptr
        || dynamic_cast<Path*>(trapNode->getData()) == nullptr);
    delete trapNode->getData();
    int level = ((z + 1) * 100) + ((y + 1) * 10) + (x + 1);
    int damage = (rand() % 10) + 1;
    auto *trap = new Trap(level, damage);
    trap->setImage( ∴ "#");
    trapNode->setData(trap);
    auto *trapNodeTree = new NodeTree(trap, level);
    trapsTree->insert(trapNodeTree, isInverse: true);
    this->totalTraps++;
}
```

O(1)
O(1)
O(1)

O(1)
O(1)
O(1)
O(x + y + z)
O(1)

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
TOTAL: O(x + y + z)

```
void Board::putPotions() {
    srand( seed: time( timer: nullptr));
    NodeMatrix<Object> *potionNode;
    do {
        int x = rand() % this->orthogonalMatrix->getDimensionX();
        int y = rand() % this->orthogonalMatrix->getDimensionY();
        int z = rand() % this->orthogonalMatrix->getDimensionZ();
        potionNode = this->orthogonalMatrix->getNode(x, y, z);
    } while (potionNode->getData() == nullptr
        || dynamic_cast<Path*>(potionNode->getData()) == nullptr);
    delete potionNode->getData();
    int healing = (rand() % 10) + 1;
    auto *potion = new Potion(healing);
    potion->setImage( ∴ "&");
    potionNode->setData(potion);
    this->totalPotions++;
}
```

O(1)
O(1)

O(1)
O(1)
O(1)
O(x + y + z)
O(1)

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(x + y + z)

```
std::string Board::getTrack(const int distance) {
    std::string type;
    switch (distance) {
        case 1: {
            type = "CALIENTE";
            break;
        }
        case 2: {
            type = "TIBIO";
            break;
        }
        default: {
            type = "FRIO";
            break;
        }
    }
    return type;
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(1)

```
void Board::printBoard(NodeMatrix<Object> *node, NodeMatrix<Object> *root, int z) {
    NodeMatrix<Object> *aux = node;
    if (z >= 0 && node == root) {
        std::cout << "Tablero en Z = " << z << std::endl;
    }
    while (aux != nullptr) {
        std::cout << "|" << aux->getData()->image;
        aux = aux->getNext();
    }
    std::cout << "|" << std::endl;
    if (node->getBottom() != nullptr) {
        printBoard( node: node->getBottom(), root, z);
    } else {
        if (root->getBack() != nullptr) {
            printBoard( node: root->getBack(), root: root->getBack(), z + 1);
        }
    }
}
```

O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(n)
TOTAL: $O(n^3)$

- Game.cpp
- Movement.cpp
- Player.cpp

DESCRIPCION DE LAS ESTRUCTURAS DE DATOS

En el software se usaron las Estructuras de Datos tales como: Listas Enlazadas (Linked List), Árbol Binario de Búsqueda (Tree BB) y Matriz Ortogonal (Orthogonal Matrix). A continuación se incluye una visualización de dichas estructuras, así como una breve explicación de su funcionamiento para entender los algoritmos de implementación.

Lista Enlazada Simple (Linked List)

Esta Lista tiene un solo enlace hacia adelante, tal cual se muestra en la imagen:

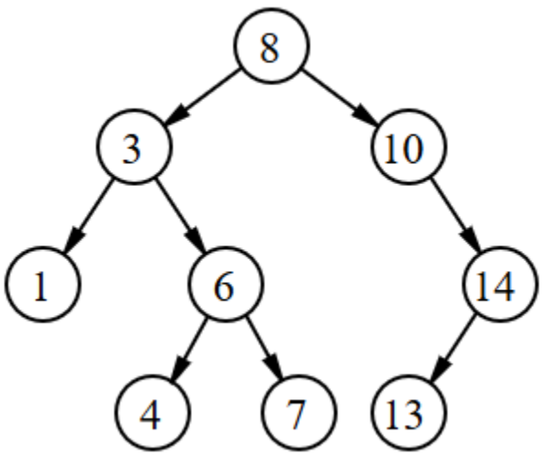
Es una estructura de datos lineal formada por una secuencia de nodos. Cada nodo contiene al menos dos partes fundamentales: un **dato** y un **enlace o referencia** al siguiente nodo de la lista.

Árbol Binario de Búsqueda (Tree BB)

Es un tipo de árbol binario donde:

- Cada nodo tiene un valor único
- Los valores menores al del nodo están en su árbol izquierdo
- Los valores mayores están en el subárbol derecho

Esto permite búsquedas, inserciones y eliminación eficientes en promedio.

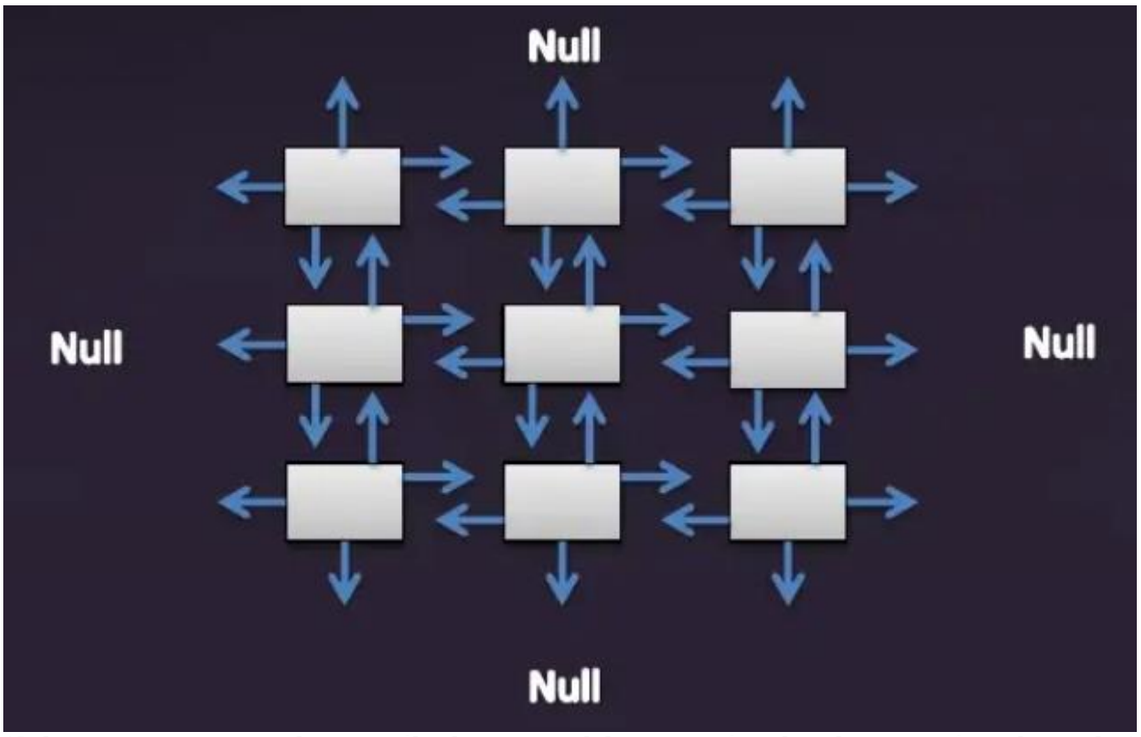


Matriz Ortogonal (Orthogonal Matrix)

Es una estructura de datos que representa una matriz dispersa (con muchos ceros o valores nulos), utilizando listas enlazadas para almacenar solo los elementos no nulos. Se usa para ahorrar espacio en matrices muy grandes con pocos valores significativos.

- Cada fila y columna se representa como una lista enlazada
- Los nodos contiene como mínimo:
  - Valor
  - Referencia a la fila siguiente y anterior
  - Referencia a la columna siguiente y anterior

- Referencia a la capa siguiente y anterior



(Solo una capa. Imaginarlo con múltiples capas adelante y atrás enlazadas entre sí con los nodos)