# JUEGO DE SCRABBLE

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 Scarbble**, 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 Scrabble** 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 Scrabble** 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 SERVICIOS CRITICOS: 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.

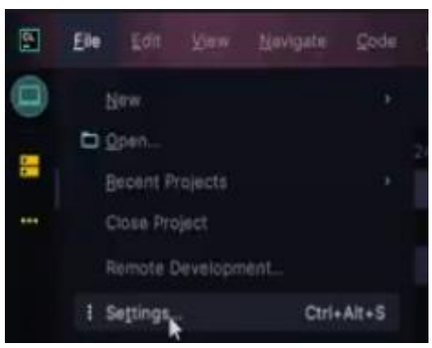
- Instalación

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.

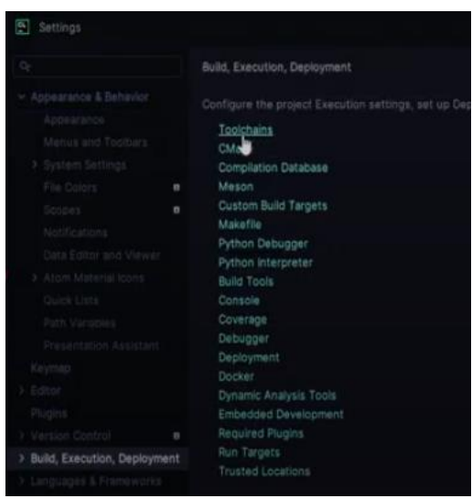
- Configuración

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.

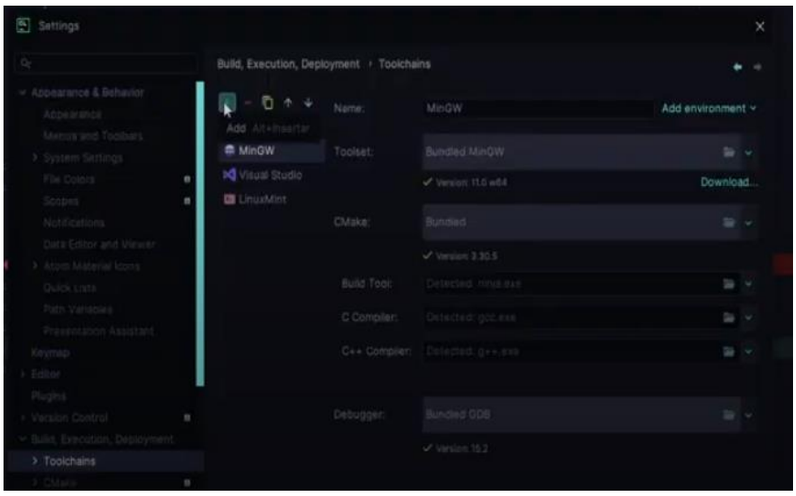
Dirígete a: Settings…:



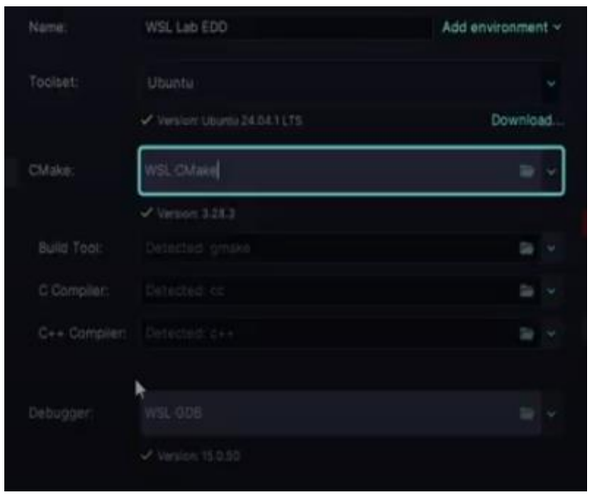

Dirígete a: Build, Execution, Deployment -> Toolchains



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



**COMPLEJIDAD ALGORITMICA DE SERICIOS CRITICOS**

**Ingreso de Jugadores y Fichas** O(1)

Se añaden los jugadores y sus fichas a la lista enlazada al inicio del juego

```
void insertar(T *value) {
    auto *nuevoNodo = new Node<T>(value);
    if (this->raiz == nullptr) {
        this->raiz = nuevoNodo;
        ++this->size;
        return;
    }
    Node<T> *aux = this->raiz;
    while (aux->getNext() != nullptr) aux = aux->getNext();

    aux->setNext(nuevoNodo);
    ++this->size;
}
```

**Gestión de Turnos con Cola** O(1)

Se utiliza una cola para administrar los turnos de los jugadores, facilitando su rotación eficiente

```
void encolar2(T dato) {
    auto *nuevoNodo = new Node<T>(dato);
    if (this->fin == nullptr) {
        this->inicio = this->fin = nuevoNodo;
        return;
    }
    this->fin->setNext(nuevoNodo);
    this->fin = nuevoNodo;
}
```

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

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

```
T desencolar2() {
    if (this->inicio == nullptr) return T();

    Node<T> *aux = this->inicio;
    T *dato = this->inicio->getData();
    this->inicio = this->inicio->getNext();
    if (this->inicio == nullptr) this->fin = nullptr;

    delete aux;
    return *dato;
}
```

O(1)

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

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

**Inserción y Eliminación de Fichas en Lista Enlazada** O(n)

Se gestionan las fichas disponibles del jugador mediante una lista enlazada para agregar y quitar letras

```
void insertar(T *value) {
    auto *nuevoNodo = new Node<T>(value);
    if (this->raiz == nullptr) {
        this->raiz = nuevoNodo;
        ++this->size;
        return;
    }
    Node<T> *aux = this->raiz;
    while (aux->getNext() != nullptr) aux = aux->getNext();

    aux->setNext(nuevoNodo);
    ++this->size;
}
```

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

```
T *eliminar(T *value) {
    Node<T> *aux = this->raiz;
    Node<T> *previo = nullptr;
    while (aux != nullptr && aux->getData() != value) {
        previo = aux;
        aux = aux->getNext();
    }
    if (aux == nullptr) return nullptr;

    if (previo != nullptr) {
        previo->setNext(aux->getNext());
    } else {
        this->raiz = aux->getNext();
    }
    aux->setNext(nullptr);
    --this->size;
    return aux->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)**

**Ordenación de Fichas por Puntación** O(n log n)

Se utiliza un algoritmo de ordenación eficiente para mostrar las fichas en orden de mayor a menor puntuación

```
// Inserta una ficha en la lista enlazada ordenada de mayor a menor valor.
// Complejidad: O(n log n)
void Jugador::insertarFichaOrdenada(Node<Ficha> *nodoNuevo) {
    if (this->listaFichas->isEmpty() || nodoNuevo->getData()->getValor() > this->listaFichas->getRaiz()->getData()->getValor()) {
        nodoNuevo->setNext(this->listaFichas->getRaiz());
        this->listaFichas->setRaiz(nodoNuevo);
    } else {
        Node<Ficha> *aux = this->listaFichas->getRaiz();
        while (aux->getNext() != nullptr
            && aux->getNext()->getData()->getValor() >= nodoNuevo->getData()->getValor()) {
            aux = aux->getNext();
            }
        nodoNuevo->setNext(aux->getNext());
        aux->setNext(nodoNuevo);
    }
    this->listaFichas->setSize(this->listaFichas->getSize() + 1);
}
```

**Cálculo y Ordenación de Puntuación** $O(n^2)$

Se calcula la puntuación de cada jugador y se ordena la lisa de puntuaciones al final del juego

```
// Ordena alfabéticamente el arreglo de palabras (algoritmo de burbuja).
// Complejidad: O(n^2)
void ReportsController::jugadoresOrdenadoPuntaje(LinkedList<Jugador> *jugadores) {
    LinkedList<Jugador> *aux = jugadores;                                                                   O(1)
    for (int i = 0; i < jugadores->getSize() - 1; i++) {                                                    O(n)
        for (int j = i + 1; j < jugadores->getSize(); j++) {                                                O(n)
            if (aux->getElement(i)->getData()->getPuntos() < aux->getElement(j)->getData()->getPuntos()) {  O(1)
                Jugador *jugadorAux = aux->getElement(i)->getData();                                        O(1)
                aux->getElement(i)->setData( value: aux->getElement(j)->getData());                         O(1)
                aux->getElement(j)->setData(jugadorAux);                                                    O(1)
            }
        }
    }
    Node<Jugador> *nodoAux = aux->getRaiz();                                                                O(1)
    int i = 1;                                                                                              O(1)
    while (nodoAux) {                                                                                       O(n)
        std::cout << i << ". " << nodoAux->getData()->getNombre() << " -> " << nodoAux->getData()->getPuntos() << " puntos" << std::endl;  O(1)
        nodoAux = nodoAux->getNext();                                                                       O(1)
        i++;                                                                                                O(1)
    }
    std::cout << std::endl;                                                                                 O(1)
}
```

**TOTAL: $O(n^2)$**

**Ordenación de Palabras Clave** $O(n^2)$

Antes de iniciar la partida deberá el programa ordenar estas palabras alfabéticamente

```cpp
// Ordena alfabéticamente el arreglo de palabras (algoritmo de burbuja).
// Complejidad: O(n^2)
void PalabrasController::ordenarPalabras(LinkedList<std::string> *palabrasExtraidas) {
    for (int i = 0; i < palabrasExtraidas->getSize() - 1; i++) {                          // O(n)
        for (int j = i + 1; j < palabrasExtraidas->getSize(); j++) {                      // O(n)
            if (palabrasExtraidas->getElement(i)->getValue() > palabrasExtraidas->getElement(j)->getValue()) {  // O(1)
                std::string aux = palabrasExtraidas->getElement(i)->getValue();           // O(1)
                palabrasExtraidas->getElement(i)->setValue(palabrasExtraidas->getElement(j)->getValue());  // O(1)
                palabrasExtraidas->getElement(j)->setValue( aux);                         // O(1)
            }
        }
    }
    std::cout << "---------- Palabras Registradas ----------" << std::endl;              // O(1)
    for (int i = 0; i < palabrasExtraidas->getSize(); ++i) {                              // O(n)
        std::cout << palabrasExtraidas->getElement(i)->getValue() << " - ";           // O(1)
    }
    std::cout << std::endl;                                                           // O(1)
    std::cout << std::endl;                                                           // O(1)
}
```

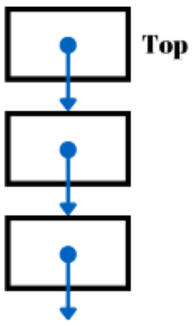
**TOTAL: $O(n^2)$**

## DESCRIPCION DE LAS ESTRUCTURAS DE DATOS

En el software se usaron las Estructuras De Datos tales como: Pilas (Stack), Cola y Listas (Linked List). A continuación se incluye una visualización grafica de dichas estructuras, así como una breve explicación de su funcionamiento para entender los algoritmos de implementación

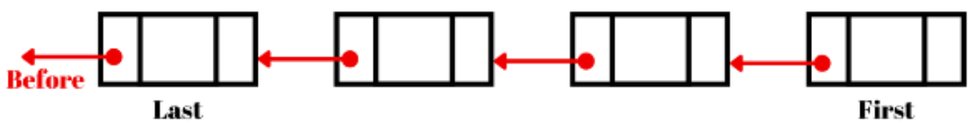
### Pila (Stack)

En una Pila se ingresan elementos hasta el tope y solo pueden sacarse de ahí, sigue la regla **"El último que entra es el primero en salir – Last Int First Out - LIFO"**




### Cola (Queue)

En una Cola se ingresan elementos hasta el top y solo puede sacarse de ahí el primer elemento, sigue la regla de **"El Primero que entra es el primero en salir – First Int First Out - FIFO"**




### Lista Enlazada Simple (Linked List)

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