

**Informe taller**

**Videojuego**

**Kight**

****

**Integrantes:** Javier Martinez Araya

Alonso Sepúlveda Rivera

**Docente:** Jose Luis Veas

**Ayudantes:** Matias Roa

Sebastian Pleticosic

Kight, hace referencia a “Knife”(Cuchillo) y “Fight”(Pelea), estas palabras son parte del núcleo del juego ya que se basa en lo mencionado anteriormente. Los personajes principales se encuentran en una disputa, ambos están cursando Estructura de Datos y estaban desarrollando un taller juntos, tras no poder resolver sus diferencias con respecto al taller, una discusión se agravó y los llevó al punto en el que se está en el juego. El ganador será quien tenga la razón.  
El juego es de pelea por turnos en formato 2D, desarrollado en C++ junto a la librería SFML 2.6.2

**Diseño Visual**

Se optó por un diseño 2D retro y sprites simples, inspirado en los antiguos juegos de pelea. Buscando un enfoque más cómico se decidió que fuera desarrollado en un entorno similar al bloque G5 de la Universidad Católica del Norte, ya que en este se cursa una clase de Estructura de Datos.

**Arena de juego:** El juego se desarrolla en una de las salas ubicada en el G5 como se puede apreciar en la siguiente imagen



**Personajes:** Los personajes que protagonizan la pelea son Deivid y Paulo

**Jugador Deivid Jugador Paulo**

**Ataque**  **Ataque**

**Bloqueo** **Bloqueo** 

Ambos personajes tienen 100 puntos de vida, los movimientos que pueden realizar son los mismos: Moverse tanto a la izquierda como a la derecha, Atacar y Bloquear.

**Jugabilidad**

Kight es un juego de pelea por turnos, el objetivo de este es “eliminar” a tu oponente para demostrar tu razón. Hay diferentes acciones que se pueden realizar para poder llegar al objetivo:  
 **Ataque:** El jugador podrá realizar una estocada hacia adelante la cual restará 20 puntos de vida al oponente, tras 5 golpes el jugador oponente será eliminado del área de combate.

**Bloqueo:** Al ser realizado este denegará todo el daño infligido por el jugador rival, no obstante para que este sea efectivo, tiene que ser realizado antes que el ataque del opuesto, si no el daño será aplicado.

**Movimiento:** Esta acción permite que el jugador pueda moverse una cantidad estática de espacio, es una alternativa al bloqueo aunque sigue la misma lógica, ya que para evitar el daño el jugador tendrá que alejarse del oponente antes que este haga su ataque. Puede ser tanto como para alejarse del oponente como para acercarse al mismo.

Todas las acciones mencionadas anteriormente toman un turno y para poder llegar a la eliminación del rival se tendrá que predecir parcialmente qué es lo que este hará.

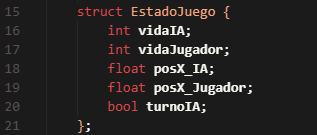
**Funcionamiento de la IA**

La IA tiene una estructura que se llama EstadoJuego que contiene la Vida del jugador como de la IA, la posición en la que se encuentran ambos y de quien es el turno.

*/\*\**

*\* @brief Estructura que representa el estado del juego para la IA*

*\*/*



Cuando la IA tiene que actuar llama a decidirAccion, en el cual se generan y simulan todas las acciones posibles a realizar, moverse hacia derecha, moverse hacia la izquierda, atacar y bloquear.

*/\*\**

*\* @brief Decide la mejor acción para la IA usando Minimax*

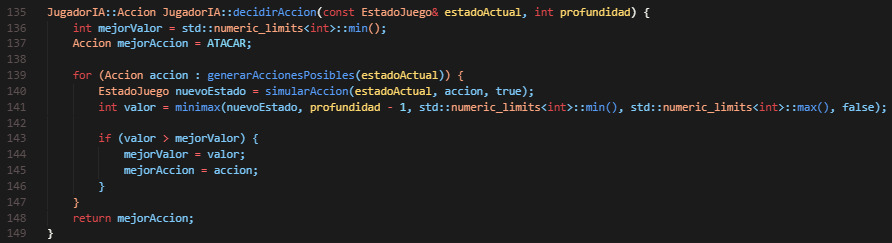
*\* @param estadoActual Estado actual del juego*

*\* @param profundidad Profundidad maxima de búsqueda*

*\* @return Accion optima para la IA*

*\*/*

Accion decidirAccion(const EstadoJuego& estadoActual, int profundidad);



Lo cual para cada acción se simula y evalúa cada estado usando el algoritmo minimax para que se explore el árbol.

*/\*\**

*\* @brief Algoritmo Minimax con poda alfa-beta*

*\* @param estado Estado actual*

*\* @param profundidad Profundidad restante*

*\* @param alpha Valor alfa para poda*

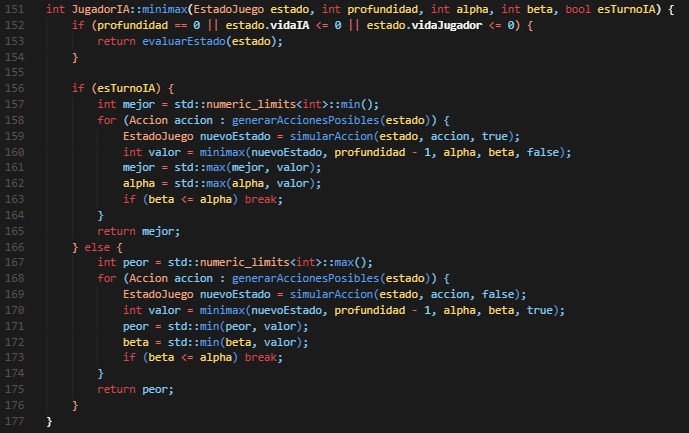
*\* @param beta Valor beta para poda*

*\* @param esTurnoIA Indica si es el turno de la IA*

*\* @return Valor heuristico del mejor movimiento*

*\*/*

*int minimax(EstadoJuego estado, int profundidad, int alpha, int beta, bool esTurnoIA);*



*/\*\**

*\* @brief Simula el resultado de realizar una acción en el estado actual*

*\* @param estado Estado actual*

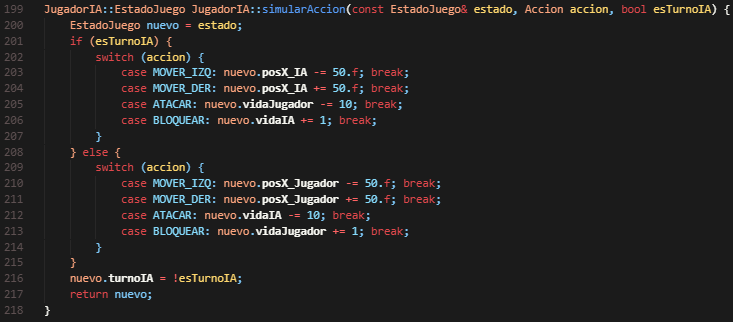
*\* @param accion Acción a simular*

*\* @param esTurnoIA Indica si es el turno de la IA*

*\* @return Nuevo estado tras aplicar la accion*

*\*/*

*EstadoJuego simularAccion(const EstadoJuego& estado, Accion accion, bool esTurnoIA);*



La función evaluarEstado en el que se calcula un puntaje para cada estado en base a la vida y distancia para que luego la IA escoja la mejor acción a realizar.

*/\*\**

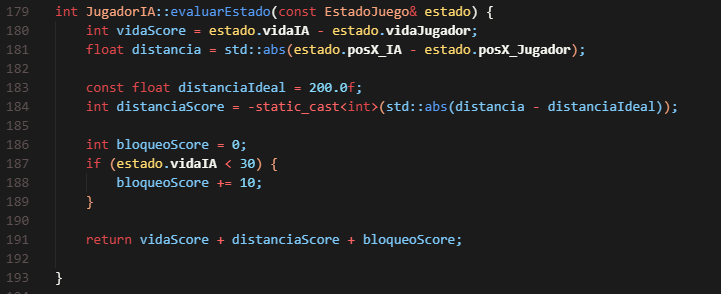
*\* @brief Evalua el estado actual del juego (heuristica)*

*\* @param estado Estado a evaluar*

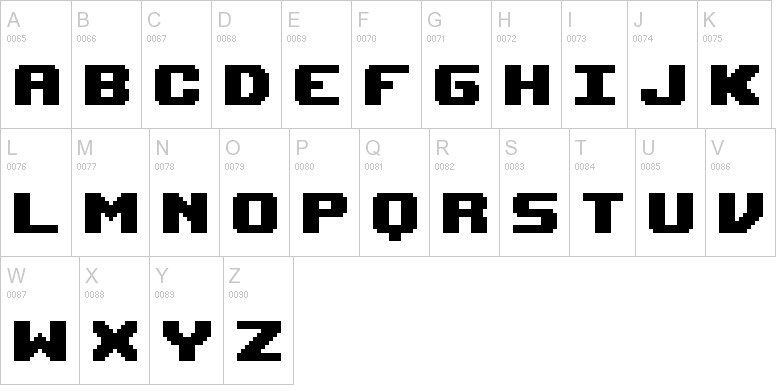
*\* @return Valor heuristico del estado*

*\*/*

int evaluarEstado(const EstadoJuego& estado);

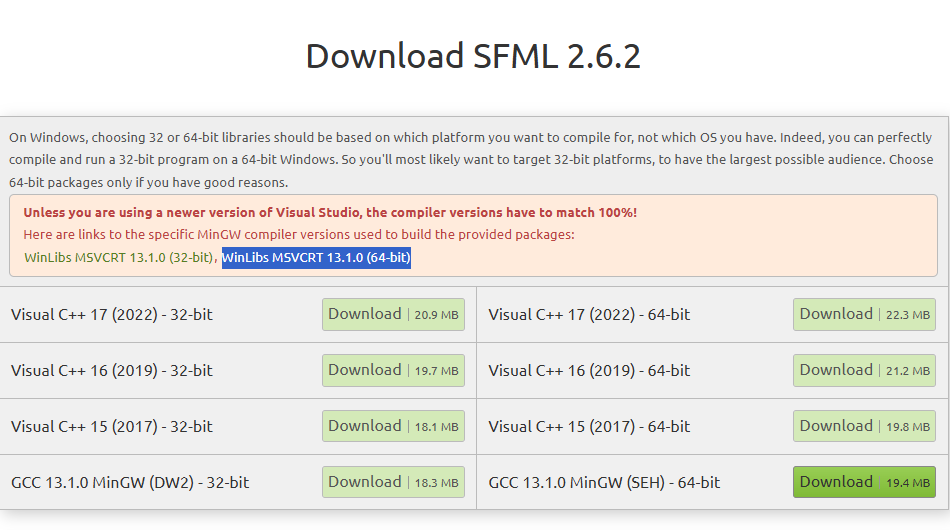


**Tipo de letra usada**

El tipo de letra que fue usado en todo el juego fue la letra “Upheaval” como se puede apreciar a continuación:****

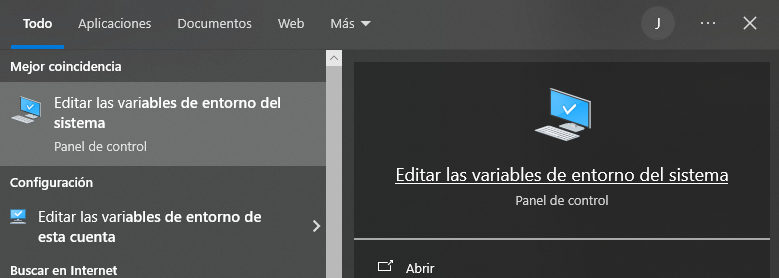
**Ejecución del juego**

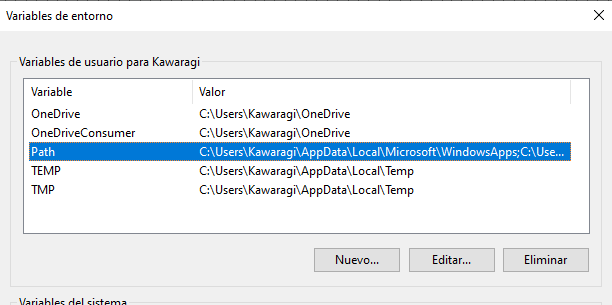
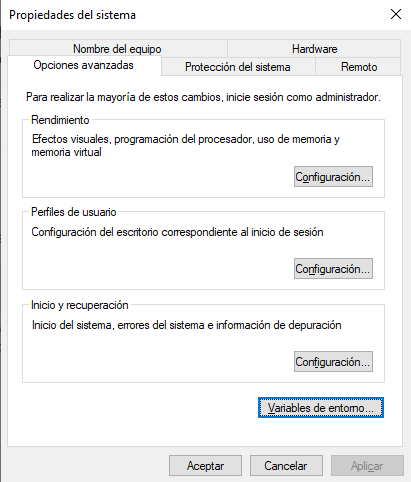
**Requisitos:** Debes tener instalado SFML 2.6.2 , específicamente la versión **GCC 13.1.0 MinGW (SEH) - 64-bit** y [WinLibs MSVCRT 13.1.0 (64-bit)](https://github.com/brechtsanders/winlibs_mingw/releases/download/13.1.0-16.0.5-11.0.0-msvcrt-r5/winlibs-x86_64-posix-seh-gcc-13.1.0-mingw-w64msvcrt-11.0.0-r5.7z) como compilador.

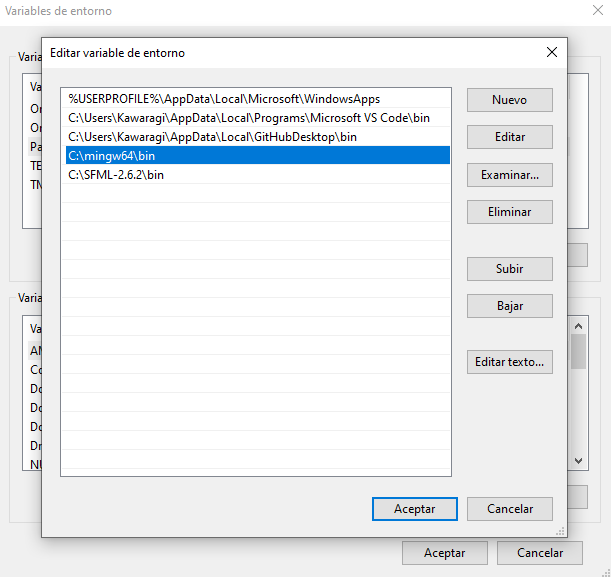


**Ubicación:** Ambos .zip deben ser extraídos y dejar su respectiva carpeta extraida en el disco local C.

**Variables de entorno:** En las propiedades del sistema donde se encuentran la variables de entorno, en Path, debemos agregar la ruta donde se encuentran tanto SFML como mingw las rutas bin y luego aceptar.







**Compilación:** En la terminal debes ejecutar el comando g++ src/\*.cpp -o juego.exe -Iinclude -IC:/SFML-2.6.2/include -LC:/SFML-2.6.2/lib -lsfml-graphics -lsfml-window -lsfml-system

**Ejecución:** Una vez compilado correctamente, el juego se ejecuta con .\juego.exe

