

**República Bolivariana de Venezuela
Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Asignatura: Algoritmos y Estructura de Datos**

INFORME DEL PROYECTO #1

Profesor

Tony Briceño

Estudiantes

José Maceiras
C.I. 31.228.863

Leonardo Espinola
C.I. 32.105.059

Caracas, 21 de mayo de 2025

INFORME

Otoño del 350 DF
oficina equipo delta
Oficina sub-regional de Valtrya
Departamento de tecnología y desarrollo

Buenas, se reporta equipo Delta, hemos logrado desarrollar el sistema Techno-Arcano lo más rápido que hemos podido, procedemos con la explicación de su funcionamiento para que puedan aplicarlo de inmediato.

El sistema consta de muchas partes pero las 3 más importantes son: Lectura de la ficha de los Numoris, el backtracking y la representación de combate numer.

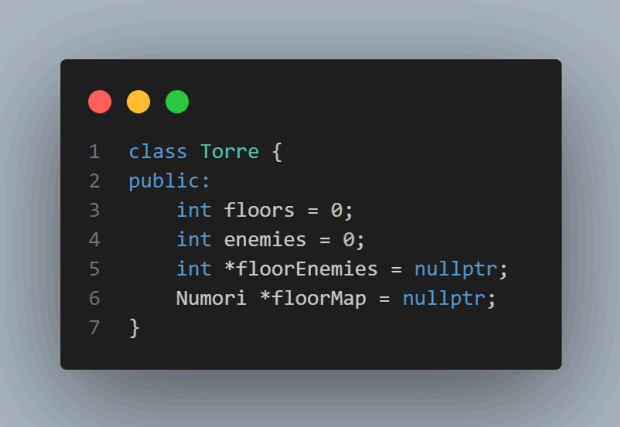
LECTURA

Hemos hecho una representación del numer en la clase llamada "Numori" para poder hacer la lectura de la ficha, esta representación consta de: id, nombre, tipo, daño y vida del numer

```
1  class Numori {  
2  public:  
3      int id = 0;  
4      string name = "";  
5      string type = "";  
6      float attack = 0.0;  
7      float life = 0.0;  
8  };
```

La lectura de la ficha funciona en la clase "ReadNumoris" y consta de 1 lectura de la primera cadena de caracteres que siempre será el número de numoris y toma ese valor guardándolo para crear un arreglo de numoris y un ciclo while que se beneficia de que los datos son siempre 5 y están en la misma posición, de no haber sido así este sistema sería inviable. El ciclo revisa palabra por palabra omitiendo SP (espacios) y guardando cada palabra dependiendo del valor de i (el dato revisado) en el numer "j" que toque, hasta rellenar el arreglo de numoris con sus datos respectivos.

También realizamos una representación de una torre en una clase llamada "Torre" para poder guardar los enemigos de la misma y los demás datos necesarios.



```
1 class Torre {
2 public:
3     int floors = 0;
4     int enemies = 0;
5     int *floorEnemies = nullptr;
6     Numori *floorMap = nullptr;
7 }
```

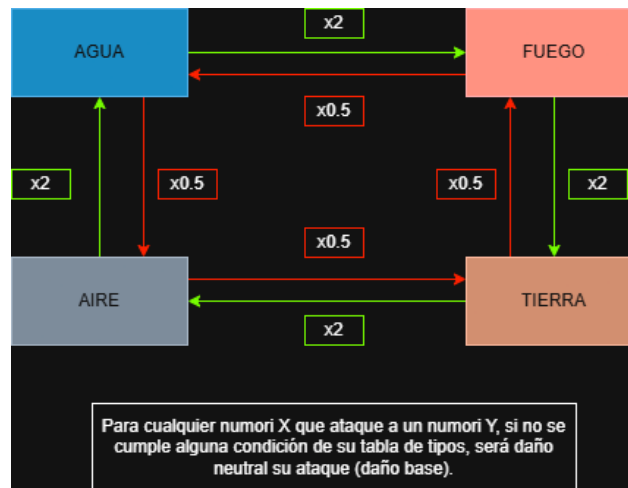
ReadTower guardamos cuantos pisos tiene (en floors), cuantos enemigos hay en total (en enemies), cuantos enemigos hay por piso (en floorEnemies) y todos los enemigos (en floorMap).

La lectura de una torre ReadTower simplemente lee el archivo TorreX.in y guarda los valores del archivo en un objeto Torre.

floorEnemies es un simple arreglo lineal que toma los valores de cuantos numoris hay en cada piso para poder recorrerlo después sin necesidad de usar una matriz. Más adelante, en el combate se explicará con más detalle esta implementación.

COMBATE

Logramos hacer una réplica exacta simulada de los combates que tienen los Numoris en la realidad. Para ello, primero tuvimos que desarrollar en el programa la función `AttackRival()`, que permite que un Numer seleccionado ataque a un Numer objetivo. Esta función se limita a calcular el daño que se le hace al rival, extrayendo el daño del Numer seleccionado y aplicándolo al objetivo, teniendo en cuenta los modificadores de daño correspondientes. Esta parte del combate se puede resumir en este diagrama:



Luego tenemos la función principal de combate: `Combat()`, es una función booleana recursiva que necesita de un equipo de 6 numoris para poder combatir, además de la lista de enemigos de un piso en la torre.

La implementación de la torre simulada, al guardar todos los Numoris rivales de todos los pisos en un único arreglo, necesita de 2 funciones extras para poder obtener el rango de ese arreglo en el piso que queremos combatir.

Para ello, simplemente ideamos 2 funciones, `getFloorInit()` y `getFloorEnd()`, estas funciones, como su nombre lo indica, nos permite obtener la posición inicial o final del arreglo de enemigos en la torre en un determinado piso (esto es posible gracias a que en la lectura de la torre guardamos dentro del objeto un arreglo de cuántos enemigos hay por piso).

Una vez tenemos el rango, el sistema selecciona el primer Numer vivo en nuestro equipo y retorna su posición interna del arreglo. Lo mismo ocurre con los Numoris de la torre. Esto es posible gracias a la función `SearchNumoriAliveAndReturnArrPosition()`. El valor retornado por la función se guardará en `SelectedUser` y `SelectedRival`. En caso de que no haya ninguno vivo, retornara -1, lo que significa que el combate ha terminado en ese piso y retorna true si hemos ganado el combate, o false si lo hemos perdido.

Al ser una simulación del combate, replicamos todas las reglas necesarias para que sea realista, por lo que se cumple el orden de los turnos: el retador siempre comenzará primero (gracias a un booleano llamado `userAlreadyHaveTurn`, que siempre inicia en falso y es un parámetro incluido en la firma de la función). Cuando sea el turno tanto del usuario y del

rival, se realizarán los daños correspondientes y se alternará el valor de `userAlreadyHaveTurn` dependiendo de si fue el turno del usuario o del rival. Esta metodología también nos permite cumplir con la otra norma del orden de los turnos: si el usuario debilita a un Numer, el siguiente turno es del rival y viceversa.

Cuando es el turno del Numer rival, en cada turno se añade el daño realizado a nuestros Numoris y las bajas que nos han provocado. Esta cuenta se lleva en las variables globales `current_damage` y `current_deaths`. Estas estadísticas recopiladas nos permiten ver que combinación es más óptima para superar una torre con el menor daño posible y solo se reinician en cada instancia de un equipo nuevo (es decir, cuando otro conjunto de numoris entra de nuevo a la misma torre, las estadísticas empiezan en 0 nuevamente, para realizar las respectivas comparaciones al final en caso de que el equipo supere la torre).

Una vez se ha terminado el turno, se llama de nuevo a la función `Combat()`, pasando el estado de los equipos del usuario y rival y de quien fue el último turno, para seguir con el siguiente turno del combate hasta que alguno de los dos haya perdido.

LA TORRE

El combate depende de la función `TorreLimpiada()`, esta es la función que se llama desde el `backtracking` y requiere de un arreglo de numoris (el equipo) y la instancia de la Torre retorna un booleano `true` si un equipo de 6 numoris logró completar satisfactoriamente una torre y `false` cuando no.

La función se encarga de llamar a `Combat()` por cada piso que tenga de la torre, en caso de ganar un combate, se suma +1 al contador de victorias, en caso de perder un combate, retorna `false` y significa que ese equipo no pudo completar la torre, por lo que no es una solución válida.

La función retorna `true` (es una solución válida) si y sólo si el número de victorias es igual al número de pisos de la torre.

BACKTRACKING

Esta implementación requirió de 4 funciones auxiliares principales, las cuales son:

`alternativaValida()` esta función recibe el conjunto solución, el paso en el backtracking y un numer como alternativa a evaluar, comprueba que el paso del backtracking sea menor a la cantidad máxima de numoris, en el caso de que en efecto sea menor pasamos a la siguiente condición la cual es que la id del numer evaluado no coincida con ninguna id que ya se encuentre en el conjunto solución en cualquiera de los casos que no se cumpla retornara false

`deshacerAlternativa()` esta función convierte un numer en la posición "paso" del conjunto solución a un numer vacío

`aplicarAlternativa()` esta función recibe el conjunto solución, un numer como alternativa y el paso del backtracking, guardando en el arreglo conjunto solución en la posición paso al numer alternativa

`esMejorSol()` esta función sólo recibe el conjunto solución y en caso de ser la primera solución encontrada reemplaza a solución por conjunto solución y cambia a false el valor de la variable `hasFindedSol`. De no ser la primera solución la función preguntará si la cantidad de muertes es menor que la cantidad de muertes anterior, de serlo se reemplaza la solución. De no serlo se preguntará si la cantidad de muertes es igual y la cantidad de daño es menor que la cantidad de daño anterior, de serlo se reemplaza la solución y de no serlo se preguntará si la suma de todas las id de los numoris es menor que la suma de todas las id de los numoris de la solución anterior, de serlo se reemplaza la solución y de no serlo se preguntará si el número formado por las id de los numoris es menor que el número formado por las id de los numoris de la solución anterior, de serlo se reemplaza la solución, de no serlo significa que este equipo de numoris es inferior al anterior y se mantendrá el anterior

El backtracking funciona recorriendo la cantidad de numoris máxima en el arreglo del equipo la cual son 6, recibe el paso(el cual será 0 la primera llamada), recibe el arreglo de los numoris totales(alternativas) y recibe un arreglo `conjunto_solucion` el cual habrá que llenar. En cada iteración del ciclo while se preguntará si la alternativa "i" es válida llamando a `alternativaValida()` en caso de no serlo continúa iterando hasta encontrar una alternativa válida, al encontrarla aplicará esta alternativa al arreglo `conjunto_solucion` y preguntará si este equipo puede derrotar la torre mediante la función `TorreLimpiada()`, si no fueron capaces de derrotar a la torre el backtracking se volverá a llamar a sí mismo pasando el `paso+1`, conjunto alternativas, instancia de la torre y conjunto solución. En caso de haber derrotado la torre se llamará a la función `esMejorSol()`, en caso de serlo se reemplaza esta solución por la anterior. Una vez terminado eso deshará la última alternativa para poder seguir buscando otras soluciones

- Att. Equipo Delta.