UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN
ALGORITMOS Y ESTRUCTURAS DE DATOS
PROYECTO #1

LA PACIFICACIÓN DE VALTRYA

ALUMNOS:

CI: 30.347.186 YOSEANNI BERMUDEZ
CI: 30.035.269 DANIEL MOROS

Sobre la pacificación de Valtrya.

En el universo de los videojuegos de rol y estrategia, la conformación de un equipo óptimo es, a menudo, la clave para superar desafíos complejos. El presente programa en C++ aborda esta problemática al simular combates en una "torre" y determinar la combinación ideal de "Numoris" para vencerla. Actúa como una herramienta de optimización que explora sistemáticamente todas las posibles formaciones de un equipo, evaluando su rendimiento contra enemigos predefinidos para identificar la estrategia más eficiente. Este enfoque algorítmico no solo demuestra principios fundamentales de programación como la gestión de memoria dinámica y la lectura de archivos, sino que también implementa una técnica de backtracking para la búsqueda de soluciones.

Funcionalidad Detallada del Programa

El programa se estructura en varias funciones interconectadas, cada una con un propósito específico que contribuye al objetivo final: encontrar el mejor equipo de Numoris para conquistar una torre.

1. Se crea la clase Numor: que encapsula las características esenciales de cada criatura:

`id`: Un identificador único para cada Numor, tipo entero.
`nombre`: El nombre de la criatura, tipo string.
`tipo`: Su tipo elemental (Agua, Fuego, Tierra, Aire), crucial para las ventajas y desventajas elementales, tipo string.
`dmg`: El valor de daño base que inflige, tipo entero.
`life`: La cantidad de puntos de vida, tipo entero.

Esta estructura permite manejar de forma organizada la información de cada combatiente, tanto del equipo del jugador como de los adversarios de la torre (numoris oscuros).

2. Se cargan los datos: `llenarArreglo()` y `leerTorre()`: Antes de iniciar cualquier simulación, el programa necesita cargar la información de los Numoris y la estructura de la torre a enfrentar.

`llenarArreglo(const string& nombreArchivo, Numor* numoris, int N)`: Esta función se encarga de poblar un arreglo dinámico de objetos `Numor` a partir de un archivo de texto (`NumorisDB.in`). Lee la cantidad total de Numoris (`N`) de la primera línea del archivo y luego itera para extraer los `id`, `nombre`, `tipo`, `dmg` y `life` de cada uno. Es fundamental para tener acceso a la base de datos completa de Numoris disponibles para la formación del equipo.

`leerTorre(string& filename, int **Torre, int pisos)`: Esta función lee la configuración de una torre específica (por ejemplo, "Torre1.in", "Torre2.in", etc.). El nombre

del archivo se construye dinámicamente en función de la entrada del usuario (X). La función extrae el número de pisos de la torre y, para cada piso, lee hasta 10 IDs de Numoris, que representan los enemigos a enfrentar en ese nivel. Se utiliza un arreglo dinámico bidimensional (`int** Torre`) para almacenar estos IDs, con un manejo cuidadoso de la lectura de la línea para separar los números correctamente.

3. Se implementan los combates: `calcularVentaja()`, `atacar()` y `simularCombatePiso()`:

`calcularVentaja(const string& tipo1, const string& tipo2)`: Implementa un sistema de ventaja elemental simple. Si un tipo es fuerte contra otro (por ejemplo: Agua contra Fuego), el daño se duplica (factor 2.0). Si es débil, el daño se reduce a la mitad (factor 0.5). En cualquier otro caso, el daño es neutral (factor 1.0). Esta función es importante para la dinámica de los combates, en base al tipo de Numor.

`atacar(Numor& atacante, Numor& defensor)`: Simula un único ataque. Calcula el daño ocasionado por el atacante al defensor, aplicando el factor de ventaja elemental. Luego, reduce la vida del defensor en la cantidad de daño calculada. Retorna “true” si el `defensor` es derrotado (su vida llega a cero o menos), y false en caso contrario.

`simularCombatePiso(Numor (&equipo)[6], Numor oscuros[], int cantidadOscuros)`: Esta es la función central para simular un combate en un único piso de la torre. Crea copias de los Numoris del equipo y de los enemigos del piso para no alterar sus estados originales. Luego, simula un turno a turno donde el Numor del equipo actual y el Numor oscuro actual se atacan mutuamente hasta que uno de ellos es derrotado. Registra las bajas sufridas por el equipo del jugador y el daño total recibido. Retorna una estructura ResultadoCombate que indica si el piso fue victorioso, cuántas bajas se sufrieron y cuánto daño total se recibió.

4. Se recorre la Torre y se preparan de enemigos: `simularCombateTorre()` y `prepararNumorisOscuros()`: Estas funciones coordinan la simulación a través de múltiples pisos y la preparación de los enemigos para el combate.

****prepararNumorisOscuros(int** Torre, int pisos, int* cantidadPorPiso, Numor* numorisDB, int cantidadNumoris):** Transforma los IDs de Numoris leídos de la Torre en objetos Numor completos, listos para la simulación. Recorre cada piso de la Torre, cuenta los Numoris presentes y busca sus datos completos en la numorisDB.

****`simularCombateTorre(Numor equipo[6], Numor** numorisOscuros, int* cantidadPorPiso, int pisos)`:** Esta función simula el progreso del equipo del jugador a través de todos los pisos de la torre. Mantiene un estado equipoActual para reflejar las

bajas sufridas. Itera a través de cada piso, llama a `simularCombatePiso` y acumula el daño total y las bajas. Si el equipo pierde en algún piso, la simulación de la torre se detiene, y se marca como una derrota total.

5. Optimización del Equipo: `encontrarMejorEquipoBT()` y `encontrarMejorEquipo()`: Aquí es donde el algoritmo de optimización entra en juego, utilizando una estrategia de backtracking para explorar el gran espacio de soluciones.

`**encontrarMejorEquipoBT(Nu.mor* numorisDB, int cantidadNumoris, int tamPer, Numor equipoActual[6], bool usado[], int numEle, Numor** numorisOscuros, int* cantidadPorPiso, int pisos, int mejorEquipoIds[6], int idsActuales[6], int equipoSize, int& menorDanio, int& menorBajas, int& menorSumaIds)`: Esta es la función recursiva de backtracking.

Caso Base: Cuando `numEle` (elementos actuales en el equipo) alcanza `tamPer` (tamaño deseado del equipo, que es 6), significa que se ha formado una permutación completa. En este punto:

Se llama a `simularCombateTorre` para evaluar el rendimiento de este equipo contra la torre. Luego, se compara el `resultado` de esta simulación con el `menorBajas`, `menorDanio` y `menorSumaIds` registrados hasta el momento. Entonces, si el `resultado` actual es una victoria y mejora los criterios (menos bajas, luego menos daño total, luego menor suma de IDs para desempatar, y finalmente orden lexicográfico de IDs), se actualiza el `mejorEquipoIds` y las variables de control (`menorBajas`, `menorDanio`, `menorSumaIds`), imprimiendo un mensaje de "Nuevo mejor equipo encontrado".

Paso Recursivo: El bucle `for` itera sobre todos los `Numoris` disponibles en `numorisDB`. Para cada `Numor` que no ha sido `usado` aún en la permutación actual:

Primero se marca como `usado`, se agrega al `equipoActual` en la posición `numEle`. Luego, se realiza una llamada recursiva a `encontrarMejorEquipoBT` con “numEle + 1” para conseguir el siguiente elemento del equipo. Después de la llamada recursiva (backtracking), se marca el `Numor` como no `usado` para permitir que sea parte de otras permutaciones.

`**encontrarMejorEquipo(Numor* numorisDB, int cantidadNumoris, Numor** numorisOscuros, int* cantidadPorPiso, int pisos, int mejorEquipoIds[6])`: Esta función inicializa las variables para el backtracking (bajas mínimas, daño mínimo, etc.) y realiza la primera llamada a encontrarMejorEquipoBT() para

comenzar el proceso de búsqueda.

6. Función Principal (`main`):

Abre y lee la cantidad de Numoris de NumorisDB.in. Crea dinámicamente el arreglo numorisDB y lo llena. Solicita al usuario el número de la torre a enfrentar (X). Construye el nombre del archivo de la torre (TorreX.in). Abre y lee la información de la torre para obtener el número de pisos. Crea y llena la estructura Torre con los IDs de los Numoris oscuros. Prepara los objetos Numor completos para los enemigos de la torre en numorisOscuros. Llama a encontrarMejorEquipo para iniciar la búsqueda del equipo óptimo. Finalmente, imprime el ID del mejorEquipoIds encontrado. Se encarga de la liberación de toda la memoria dinámica asignada, lo cual es crucial para evitar fugas de memoria.

Este programa es un ejemplo robusto de cómo se pueden aplicar principios de programación algorítmica y estructuras de datos para resolver problemas de optimización en contextos simulados. A través de la combinación de la carga eficiente de datos, una simulación de combate detallada y un algoritmo de backtracking exhaustivo, es capaz de explorar todas las posibilidades para identificar la configuración de equipo más ventajosa. Si bien el costo computacional puede ser significativo para conjuntos grandes de Numoris debido a la naturaleza factorial de las permutaciones, su diseño modular y claro facilita tanto su comprensión como su posible expansión y optimización futura. Es una solución completa y bien estructurada para el desafío de la torre de Numoris.