Estructuras de Datos en el Proyecto UNO

Introducción:

En la implementación del juego UNO en C + +, se han seleccionado tres estructuras de datos lineales fundamentales que se alinean perfectamente con las necesidades específicas del juego. Cada estructura responde a requisitos particulares de eficiencia, funcionalidad y simplicidad conceptual.

1. ARREGLOS DINÁMICOS (`std::vector`)

Aplicaciones Identificadas:

- **Baraja Inicial:** Construcción de las 100 cartas mediante bucles anidados, fue implementada como un arreglo dada la facilidad con la cual se puede iterar sobre sus elementos.
- **Manos de Jugadores:** Gestión dinámica de cartas en cada jugador, dado que debemos poder acceder a cualquier carta que pueda estar en la mano y que el tamaño de la misma se modifica constantemente, un arreglo dinámico fue la mejor respuesta.

Los arreglos dinámicos mediante `std::vector` son ideales para representar las manos de los jugadores y la baraja inicial porque ofrecen un balance perfecto entre flexibilidad y eficiencia. A diferencia de los arreglos estáticos, los vectores permiten que el tamaño de la mano crezca o reduzca dinámicamente según los jugadores roben o jueguen cartas, eliminando el desperdicio de memoria.

El acceso aleatorio en O(1) mediante índices facilita la selección directa de cartas por parte del usuario, mientras que las operaciones de inserción y eliminación al final mantienen una eficiencia constante. Aunque la búsqueda de cartas jugables requiere O(n), el tamaño limitado de las manos (generalmente menos de 20 cartas) hace que este costo sea aceptable, proporcionando una solución óptima para gestionar colecciones de tamaño variable con necesidad de acceso posicional inmediato.

En el caso de la baraja, el vector permite una construcción sencilla de las 100 cartas mediante bucles anidados, y luego su mezcla aleatoria. La capacidad de acceder a cualquier carta por índice es crucial para el método `shuffle`.

2. PILAS (`std::stack`)

Aplicación:

- **Pila principal:** Toma secuencial de cartas (Comportamiento LIFO). Tras aplicar la función shuffle a la baraja del juego, los elementos de esta son agregados uno por uno en la pila, desde la cual se van a formar las manos de los jugadores y se va a tomar cartas cuando sea necesario.
- **Pila de Descarte:** Acumulación de Cartas Jugadas.

La estructura de pila (`std::stack`) es la elección natural para el mazo de robo y la pila de descarte debido a su comportamiento LIFO (Last In, First Out), que se ajusta perfectamente a la mecánica natural del juego. En el mazo de robo, las cartas deben ser tomadas desde la parte superior, y en el descarte, las cartas se juegan y se colocan en la parte superior.

Esta estructura garantiza que siempre se acceda a la carta más reciente, con operaciones O(1) para insertar (`push`) y extraer (`pop`). La pila de descarte, en particular, necesita ser eficiente en la inserción de cartas jugadas y en la consulta de la carta superior para determinar las jugadas válidas. Por otro lado, el mazo de robo se va reduciendo a medida que los jugadores roban cartas. La simplicidad de la pila asegura un código claro y eficiente, sin operaciones superfluas que compliquen la lógica del juego.

3. LISTA DOBLEMENTE ENCADENADA Y CIRCULAR (`std::list`)

Aplicación:

- **Lista de jugadores (manejo de turnos):** La lista contiene la totalidad de jugadores e itera sobre ellos para pasar de turno ya sea en un sentido o su contrario, dada esta condición por una posible carta reverse.

La estructura de lista (**`std::list`)** fue ideal para el manejo de los turnos dentro del juego dado que es una lista doblemente encadenada y mediante los operadores (++) o (--) podemos usar los apuntadores para avanzar o retroceder en los elementos de la lista respectivamente. Para nuestra conveniencia efectivamente es circular, o sea que el apuntador previo de la cabeza de la lista apunta al último elemento y el apuntador siguiente del último elemento apunta a la cabeza nuevamente.