Implementación de Juego de Cartas “Guerra”

Introducción

En este informe, se aborda la implementación de un juego de cartas basado en la mecánica del juego de guerra. La implementación se enfoca en la aplicación de Tipos Abstractos de Datos (TAD), algoritmos de ordenamiento y técnicas de manejo de excepciones en Python. Además, se realiza un análisis de la complejidad de los algoritmos implementados y se incorporan pruebas unitarias para garantizar el correcto funcionamiento del código.

Objetivos

Los objetivos de este proyecto son los siguientes:

Implementación de Tipos Abstractos de Datos (TAD):

Definir y utilizar TAD para representar las cartas, el mazo, los jugadores y el juego en general, siguiendo buenas prácticas de diseño.

Aplicación de Algoritmos de Ordenamiento:

Utilizar algoritmos de ordenamiento para mezclar las cartas en el mazo y garantizar la aleatoriedad del juego.

Análisis de Complejidad de Algoritmos:

Realizar un análisis teórico de la complejidad de los algoritmos de ordenamiento utilizados para mezclar el mazo y comprender su eficiencia.

Manejo de Excepciones para Validación de Datos:

Aplicar manejo de excepciones para validar datos de entrada y garantizar la robustez y seguridad del programa.

Realización de Pruebas Unitarias:

Crear pruebas unitarias para verificar la corrección y funcionalidad de los componentes del juego, asegurando un comportamiento consistente.

Operaciones Principales

Creación y Mezcla del Mazo de Cartas:

Crea un mazo de cartas utilizando la clase Mazo, que a su vez utiliza la clase Carta. Luego, se aplica un algoritmo de mezcla para aleatorizar el orden de las cartas en el mazo.

Repartición de Cartas a los Jugadores:

Reparte las cartas del mazo a los dos jugadores, colocando cierto número de cartas en la mano de cada jugador en cada ronda.

Juego de Rondas:

Realiza una ronda de juego donde los jugadores sacan una carta de su mano, la comparan y resuelven la ronda según el resultado de la comparación. También gestiona los casos de guerra (empates) y actualiza las manos de los jugadores.

Complejidad Algorítmica

A continuación, se presenta un análisis de complejidad para algunas operaciones clave:

Cola (Clase Cola):

encolar: Tiempo O(1) amortizado.

desencolar: Tiempo O(n) en el peor caso.

Mazo (Clase Mazo):

crear_mazo: Tiempo O(1).

mezclar: Tiempo O(n) en el peor caso.

Juego de Guerra (Clase JuegoGuerra):

jugar_ronda: Tiempo O(n) en el peor caso.

jugar: Tiempo O(n) en el peor caso.

Repartición de Cartas:

Creación y mezcla del mazo: Tiempo O(n) en el peor caso.

Repartir cartas: Tiempo O(n) en el peor caso.

Resultados Exitosos

Se lograron alcanzar con éxito varios aspectos cruciales del juego:

Creación, Mezcla y Repartición de Mazos:

Se implementó la creación y mezcla del mazo de cartas, así como su repartición adecuada a los jugadores.

Funcionalidad de Enfrentamiento y Acumulación de Cartas:

Se logró la correcta comparación de cartas, enfrentamiento y la adecuada acumulación de las mismas en el mazo del ganador.

Proceso de Guerra y Gestión de Tributos:

Se implementó exitosamente el proceso de guerra, incluyendo la separación de tributos y la realización de una nueva comparación. Además, se logró visualizar estéticamente las cartas volteadas y los tributos. **Estas operaciones se realizaron correctamente, a pesar de un pequeño problema de visualización.**

Aspectos Pendientes

Visualización Final durante la Guerra:

Aunque se logró visualizar las cartas en la mesa con los tributos, no se pudo concatenar junto a las 2 últimas cartas que finalizan la guerra, mostrando estas últimas solas en la última pantalla. **Este aspecto no pudo ser completamente resuelto.**

Durante la ejecución de las pruebas unitarias, se encontró un error "NameError: name 'Mazo' is not defined" al intentar ejecutarlas. Este problema aún no pudo ser resuelto y es un aspecto pendiente en el proyecto.

Adicionalmente, cabe mencionar que en el proceso de implementación de pruebas unitarias, se realizó una modificación en la línea de código de importación para el archivo de pruebas de testing, cambiando de "from modulos.juego_guerra import JuegoGuerra" a "from JuegoGuerra import JuegoGuerra", debido a una restricción de acceso al código.

Manejo de Excepciones

Se aplicó manejo de excepciones en varias partes del código para garantizar la robustez del programa. Algunos de los manejos de excepciones utilizados incluyen:

Validación de datos de entrada.

Captura de excepciones en operaciones de manejo de cartas y mazos.

Manejo de excepciones en la creación del juego y repartición de cartas.

Conclusiones

En esta implementación del juego de guerra con cartas, he experimentado directamente la utilidad y aplicabilidad de los Tipos Abstractos de Datos (TAD) y algoritmos de ordenamiento. La estructura modular y organizada que proporcionan los TAD ha simplificado la representación y manipulación de las distintas entidades del juego, contribuyendo a un código más limpio y fácil de mantener.

La introducción de algoritmos de ordenamiento ha sido fundamental para lograr la aleatoriedad necesaria en un juego de cartas, asegurando que el mazo se mezcle de manera efectiva antes de repartir las cartas. La consideración de la complejidad algorítmica ha sido esencial para comprender la eficiencia de estos algoritmos, lo cual es crucial para garantizar un rendimiento adecuado a medida que el juego escala en complejidad.

Además, las pruebas unitarias que se realizaron han sido vitales para verificar y validar la funcionalidad correcta del código en diferentes escenarios. Este enfoque ha fortalecido la robustez del proyecto y ha aumentado la confianza en su calidad y fiabilidad.