Informe del Trabajo Práctico N°1

Aplicaciones de Tipos Abstractos de Datos (TADs)

Materia: Algoritmos y Estructuras de Datos - FIUNER

Integrantes:

Flores, Valentina

Aguilar Gonzales, Andrea Fernanda

Lell, Camila Luciana

Fecha de entrega: 2 de mayo del 2025

ACTIVIDAD 3

En esta actividad, necesitamos implementar una clase Mazo que respete y cumpla las necesidades de los test dados por la cátedra y el código del juego. Además de eso, necesitamos el uso de una Lista Doblemente Enlazada (LDE) que para ello decidimos emplear la misma LDE creada del punto 2.

Mazo no es una LDE en sí, sino que usa la LDE creada para modelar su comportamiento y poder funcionar abarcando la mayor cantidad de funciones posibles.

Planteamiento inicial:

Se nos planteó el juego “CARTAS DE GUERRA”, su modo de juego y método de victoria:
-Ambos jugadores empiezan con 26 cartas cada uno, “apuestan” una cada uno y el qué dió el mayor valor gana las dos cartas. Si ambos tiran dos cartas del mismo número, realizan un “botín de guerra” el cual consta de 3 cartas boca abajo cada jugador, vuelven a tirar una carta cada unos hasta que uno vuelva a sacar la carta de mayor valor y se lleve el botín, por lo tanto el ganador es aquel que se queda con la mayor cantidad de cartas.

Objetivo:

La cátedra otorgó para el planteamiento:

- El código con el juego “Cartas de Guerra” (juegoguerra.py)
- Los objetos “Carta” (carta.py)
- Tests para la clase mazo (test_mazo.py) y test para la clase Juego de Guerra (testjuego.py)

El objetivo principal fue crear una clase Mazo para el juego que pueda almacenar los objetos Carta, y que haga uso de una LDE. La clase Mazo debe emplear con las cartas el uso que se hace comúnmente con las mismas (sacar cartas, agregar cartas, saber la cantidad de cartas, etc.)

Solución:

Sabemos que se debe implementar un TAD (Tipo Abstracto de Datos) Lista Doblemente Enlazada, capaz de almacenar cualquier tipo de elemento comparable. Lo que hicimos fue utilizar la LDE que creamos del punto 2, pero reconociendo para qué podría servirle a la clase Mazo que crearemos después.

El mazo de cartas fue implementado utilizando la Lista Doblemente Enlazada. Cada carta en el mazo es un objeto de la clase Carta, que contiene su valor, palo y si está visible o no.

Debíamos tener en cuenta el manejo de excepciones, en este caso el mayor problema que se podría presentar en un juego así es que cuando se intenta sacar una carta de un mazo vacío, por lo tanto se lanza una excepción **DequeEmptyError** en la función: ***sacar_carta_arriba***, y la función siempre sacará en la cabeza del mazo ya que no es lógico en el juego que saque cartas por debajo del mismo. Si del contrario existen cartas en el mazo, entonces se extrae una sola carta por arriba, además implementamos una forma de "voltear" la carta y hacerla visible con ***mostrar=True.***

También debíamos incluir las operaciones: ***poner_carta_arriba*** para agregar cartas arriba del mazo SOLO cuando se reparten las cartas al inicio del juego, al final ***poner_carta_abajo*** propio en el desarrollo de la partida. También se puede obtener la cantidad de cartas en el mazo con el método **tamanio()** o usando la función ***len()*** sobre la instancia, gracias a la implementación de ***__len__.***

Finalmente, la clase incluye dos propiedades ***@property*** llamadas cabeza y cola que permiten consultar, respectivamente, qué carta está en la parte superior e inferior del mazo sin modificarlo. Esto es útil para monitorear el estado del mazo durante una partida o durante pruebas. En conjunto, este módulo permite modelar el comportamiento de un mazo de cartas de forma ordenada y coherente con la dinámica del juego.

Finalmente se realizaron pruebas automáticas (test unitarios) para verificar:

- Funcionamiento del TAD Mazo (tests de poner, sacar cartas y manejar errores).
- Funcionamiento del juego Guerra (tests de partidas simuladas que verifican ganadores y empates).

<u>Conclusiones:</u>

- La estructura de lista doblemente enlazada fue esencial para que las operaciones sobre los mazos se realizaran en tiempo constante (cuando corresponde). la LDE permite la inserción y eliminación en los extremos en O(1), lo que hace que ***poner_carta_arriba*** y ***sacar_carta_arriba*** sean eficientes
- El uso de excepciones permitió manejar de forma controlada situaciones especiales como intentar sacar una carta de un mazo vacío.
- El juego "Guerra" se comporta correctamente incluso en partidas largas, evidenciando que las estructuras y algoritmos utilizados son robustos y eficientes.
- Los tests automáticos confirmaron que todas las clases se comportan como se espera.

Ejemplos de pseudocódigo para conocer cómo mazo utiliza la LDE:

Función poner_carta_abajo(carta):

```
    # carta es un objeto de la clase Carta
    Si el mazo está vacío:
        # La lista está vacía (cabeza y cola son nulos)
        Agregar carta al final de la lista doblemente enlazada
        # Esto implica que la nueva carta se convierte en la cabeza y la cola
    Sino:
        # El mazo no está vacío
        Agregar carta al final de la lista doblemente enlazada
        # Esto implica actualizar la cola de la lista para que sea la nueva carta
    Fin
```

Funcion sacar_carta_arriba(mostrar):

```
    Si el mazo está vacío:
        Lanzar Excepción MazoVacio
    carta = cartas.extraer(0) # Extraer del inicio de la LDE
    Si mostrar es Verdadero:
        carta.visible = Verdadero #se cambia el estado de la carta
    Devolver carta extraída
    Fin
```