Práctica 3

Estructura de Datos

2014-2015

El objetivo de esta práctica es practicar las listas. Vamos a utilizar la implementación vista en clase de teoría de lista enlazada (LinkedList) y vamos a reimplementar algunas de las estructuras como pila y cola con listas enlazadas.

Ejercicio 1 (1 puntos). Definir la clase **LinkedList** (lista enlazada) según el material visto en clase de teoría. Definir un método de test para asegurar que estén bien implementados sus métodos.

Ejercicio 2 (1 puntos). Extender la lista LinkedList a **CircularLinkedList** (lista circular enlazada). Definir un método de test para asegurar que estén bien implementados sus métodos.

Ejercicio 3 (1.5 puntos). Definir la lista de jugadores del juego UNO como una lista enlazada circular. Definir un método de test para asegurar que estén bien implementados sus métodos. Ejecutar el juego para comprobar que la lista enlazada circular esté funcionando correctamente.

Ejercicio 4 (1.5 puntos). Definir la lista de jugadores como lista enlazada doble circular **CircularDoubleLinkedList**. Definir un método de test para asegurar que estén bien implementados sus métodos. Comprobad que funcionen correctamente las cartas de invertir el sentido del juego. Añadir las cartas especiales 'reverse' y 'skip' (si no las tenéis ya implementadas) que invierten el orden de jugar o hacen saltar al siguiente jugador. Estas cartas son comodines y se considera que hay dos cartas de estos tipos por color en el mazo.

Ejercicio 5 (1.5 punto). Implementar la clase Stack como una lista enlazada (**LinkedStack**). Definir una función de test para la clase para comprobar que los métodos están implementados correctamente. Definir la clase Discard_Pile como clase derivada de la clase LinkedStack en el juego ONE.

Ejercicio 6.5 (1 puntos). Implementar la clase Queue como una lista enlazada (**LinkedQueue**). Definir una función de test para la clase para comprobar que los métodos están implementados correctamente. Definir la clase Mazo como clase derivada de la clase LinkedQueue en el juego ONE.

Ejercicio 7 (opcional) (1 punto). Implementar la clase PriorityQueue como una lista enlazada (**LinkedPriorityQueue**). Definir una función de test para la clase para comprobar que los métodos están implementados correctamente. Definir la clase Player como clase derivada de la clase LinkedPriorityQueue en el juego ONE.

Ejercicio 8 (opcional) (0.5 puntos). Añadir las cartas especiales 4+ y 2+ que obliga al siguiente jugador robar 4 cartas. Esta carta es un comodín y se considera que hay dos cartas de este tipo por color en el mazo.

Ejercicio 9 (opcional) (0.5 puntos). Añadir la carta especial que cambie el color. Esta carta también es un comodín y se considera que hay dos cartas de este tipo por color en el mazo.

Ejercicio 10 (1 puntos). Explicar en detalle las implementaciones de las diferentes estructuras y el algoritmo, **la forma de ejecutarlo** y cómo habéis optimizado el código respecto a su eficiencia.

Ejercicio 11 (1 puntos). Comentar qué ventajas y desventajas tiene cada una de las estructuras implementadas con listas enlazadas respecto las implementaciones con las listas de Python.

Entrega: Se ha de entregar en un fichero comprimido con los nombres de los 2 alumnos que contenga: los ficheros con el código en Python, compilados en Wing IDE, correspondientes a los ejercicios, junto con un documento en formato pdf o Word que corresponde a los ejercicios 10 y 11. **La fecha límite de entrega depende del grupo de prácticas (consultar Campus Virtual).** No se aceptarán entregas de la práctica S3 después de esta fecha.

Apéndice

Reglas del juego:

1. Inicialmente todas las cartas están en un mazo.
2. Al principio del juego se reparten un número predefinido (normalmente 7) de cartas a cada jugador, estas cartas proceden del mazo y serán las cartas que tiene el jugador en la mano.
3. Después, se saca aleatoriamente una carta del mazo que servirá como carta inicial de la pila (lo llamaremos discard_pile) y el resto de cartas se dejan en el mazo.
4. Todos los jugadores, por turnos, deben tirar cartas a la pila hasta que puedan, del mismo color o número que la carta visible de la pila. La carta visible será la última que se ha tirado a la pila.
5. Si un jugador no tiene una carta para jugar, debe robar del mazo hasta que obtenga una que le permita jugar. Una vez haya robado una que puede tirar, puede ir tirando mientras pueda.
6. Finalmente, se define un criterio de ganar, por ejemplo, gana el que primer jugador que acabe las sus cartas de la mano.

El apéndice 2 muestra el algoritmo completo del juego del UNO.

Algoritmo principal:

Para implementar el juego, la clase **ONE** está definida de la siguiente forma:

1. La clase del juego (**ONE**) – crea los datos, prepara el juego y empieza a jugar hasta que se cumpla el criterio de acabar

    a. Datos
        i. Lista de jugadores
        ii. Baraja o mazo (Deck)
        iii. Pila (Discard_pile)

    b. Métodos
        i. Crear juego – prepara el juego (prepare_game) y lo pone en marcha (run_game)
        ii. Preparar_juego (prepare_game)
            a. Define los jugadores. Los datos del número de jugadores se introducen desde el teclado.
            b. Crea la baraja para repartir (deck)
            c. Crea la pila (discard_pile)
            d. Reparte las cartas (deal) del deck
            e. Se selecciona aleatoriamente el jugador actual (p.e. el primero)

        iii. Repartir N cartas a cada jugador (deal), p.e. N=7. El valor N se define como una constante al principio del código.

iv. Aplicar criterio para parar (stop_criterion) – Se pueden definir varios criterios, por ejemplo, cuando el primero de los jugadores haya acabado con todas las cartas de su mano, entre otras opciones.

v. Comprobar y felicitar al campeón (announce_champion) – comprueba qué jugador ha cumplido el criterio de parar el juego en primer lugar.

vi. Cambiar de turno (change_turn) – si el jugador actual es p1, pasar a p2, etc.

vii. Función principal (run_game):
    1. Mientras no se cumple el criterio de parar:
        i. Se imprime el estado del juego (visualize_state) - la última carta de la pila y las cartas del jugador actual
        ii. Se comprueba si el jugador actual puede jugar
        iii. En caso que no: el juego saca automáticamente cartas del mazo y se las asigna a la mano del jugador actual. Estas cartas se van imprimiendo por pantalla, hasta que pueda jugar.
        iv. Mientras el jugador actual pueda jugar:
            1. Se pregunta al jugador qué carta desea jugar y se lee por teclado su número
            2. Se selecciona la primera carta que tenga el número indicado por el jugador y sea compatible con la carta mostrada en el discard_pile" (select_card de la clase Player)
            3. El jugador actual juega la carta (play_a_card)
            4. Se visualiza el estado del juego

        v. Si el jugador actual cumple el criterio de parar el juego, se le felicita y se acaba el juego
        vi. Sino, se pasa el turno al siguiente jugador

En la página siguiente tenéis un diagrama que resume el funcionamiento del juego del UNO.

Diagrama del algoritmo