Práctica 2

de Estructura de Datos

2014-2015

El objetivo de esta práctica es conocer y practicar los tipos de datos abstractos: pila, cola, cola con prioridad así como las clases, el polimorfismo, el encapsulamiento y la herencia en Python.

Ejercicio 1. Implementar las clases **Stack**, **Queue** y **PriorityQueue** vistas en clase de Teoría. Definir una función de test para cada clase para comprobar que las clases estén bien implementadas. Utilizar la sobrecarga en los métodos de comparación que necesitaréis en la clase de PriorityQueue.

Ejercicio 2. Redefinir las clases **Deck**, **Discard_Pile** y **Player** como clases derivadas de una de las tres clases del Ejercicio anterior. Separar bien qué métodos y datos deben ir en las clases bases y cuáles en las clases derivadas.

¿Cómo quedan en esta nueva definición las clases?

Definir funciones de test de cada clase: Deck, Discard_Pile y Player para asegurarse que estén bien implementadas.

Ejercicio 3. Implementar la clase **One** que implemente el algoritmo del juego según el algoritmo y el diagrama en el apéndice.

Ejercicio 3' (opcional): Añadir la carta que invierte el orden de jugar de los jugadores. Esta carta es un comodín y se considera que hay dos cartas de este tipo por color en el mazo. Los comodines sólo tienen el color y la función que realizan (en este caso se invierte el orden de jugar de los jugadores).

Ejercicio 3'' (opcional): Añadir la carta que hace saltar el siguiente jugador. Esta carta también es un comodín y se considera que hay dos cartas de este tipo por color en el mazo.

Ejercicio 3''' (opcional): Implementar el juego One que permita jugar a cualquier número de jugadores (no solamente a 4). El número de jugadores se introduce por teclado al iniciarse el juego. Posteriormente, se introducen los datos por teclado de todos los jugadores (tantos jugadores como se haya definido por teclado al iniciarse el juego).

Ejercicio 4. Explicar en detalle las implementaciones de las diferentes estructuras y el algoritmo, la forma de ejecutar y cómo habéis optimizado el código respecto su eficiencia.

Ejercicio 5. Comentar en qué consiste el polimorfismo, el encapsulamiento y la herencia en esta implementación.

Entrega: Se ha de entregar en un fichero comprimido con los nombres de los 2 alumnos que contenga: el fichero con el código en python, compilado en Wing IDE, correspondientes a los ejercicios 1, 2 y 3, junto con un documento en formato pdf o Word que corresponde a los ejercicios 4 y 5. **La fecha límite de entrega depende del grupo de prácticas (consultar Campus Virtual).** No se aceptarán entregas de la práctica S2 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 (al que llamaremos ONE a partir de ahora) y que acabaremos de implementar en la segunda práctica.

Algoritmo principal:

Para implementar el juego, definiremos la clase **ONE** 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 (p.e. consideramos que hay 4 jugadores). Los datos del número de jugadores se leen desde 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. Éste valor de 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 que se desea implementar en la segunda práctica de Estructura de Datos.

Diagrama del algoritmo