Introducción a la computación / Tópicos de Programación

Taller 0

Evaluación: A partir del miércoles 30 de Septiembre de 2021

En el juego de la `Escoba del Quince` se utiliza un mazo de cartas españolas (sin 8s ni 9s) en el que pueden participar entre 2 y 5 jugadores. En cada turno, cada jugador intenta formar un *juego* de cartas que sumen quince usando una de las cartas de su mano y combinándola con una o más cartas de las que están en la *mesa* en ese momento. Si no puede formar un juego porque no encuentra una combinación que sume 15, el jugador debe desechar una de sus cartas y colocarla en la mesa. Se juega por turnos y, cuando se terminan las tres cartas repartidas inicialmente, se vuelven a repartir otras tres cartas a cada jugador usando lo que quedaba del mazo. Cuando se termina la última ronda (ya no quedan cartas para repartir), los jugadores deben contar los puntos que obtuvieron.

La `Escoba del quince` comienza de la siguiente manera:

- Se mezcla el mazo.
- Se reparten tres cartas a cada jugador.
- Se colocan cuatro cartas en la mesa.

Un detalle importante en la escoba de quince es que para hacer un juego (que la suma de cartas dé quince), se usan los valores del número de las cartas salvo en el caso de las figuras a las que se les restan 2. De este modo, las cartas del 1 al 7 aportan al juego el valor que indican, pero el 10 (*la sota*) se cuenta como 8, el 11 (*el caballo*) aporta 9 y el 12 (*el rey*) cuenta como 10.

Como dijimos, cada jugador tiene por objetivo elegir una las cartas que tiene en la mano (inicialmente tres en cada ronda, pero a medida que va jugando, le quedan menos) para hacer un juego combinando con las que hubiera en la mesa.

Cuando se han jugado todas las rondas (no quedan cartas en el mazo para repartir), se cuentan los puntos que obtuvo cada jugador. Para contar puntos, se deben evaluar los juegos hechos por cada jugador y se otorgan puntos a quienes hayan alcanzado los siguientes logros (1 por cada logro).

1. Obtener mayor **cantidad de cartas** que el resto (en caso de empate, ningún jugador obtiene puntaje por este logro).
2. Obtener mayor cantidad de cartas del **palo oro** (en caso de empate, ningún jugador obtiene puntaje por este logro).
3. Haber levantado un juego que incluya al **7 de oro** (o 7 de velos como se lo conoce también).
4. Obtener mayor puntaje en las **70** (en caso de empate, ningún jugador obtiene puntaje por este logro).
5. Haber realizado una **escoba**.

En el caso en que un jugador realice un juego con el que levante todas las cartas de la mesa, se dice que hizo una **escoba** y eso le otorga un punto adicional (indicado en el punto 5 de otorgamiento de puntos). Un jugador puede hacer más de una escoba y se le otorga un punto por cada una.

Luego de una escoba la mesa queda vacía (sin cartas). Por lo tanto el siguiente jugador no podrá hacer un juego y deberá poner una de sus cartas en la mesa.

Es posible que un jugador tenga más de una opción para hacer un juego, en ese caso utilizará alguna estrategia que apunte a mejorar sus chances de obtener alguno de los puntos del 1 al 4 enumerados arriba.

El objetivo de este trabajo será programar este juego y la dinámica que ocurre en cada ronda hasta la última, posteriormente también deberemos programar el otorgamiento de puntos a cada jugador. Vamos a darte una serie de funciones que deberás programar respetando el nombre y los parámetros que reciben. También indicaremos, cuando corresponda, si tenés que usar algunas de las funciones realizadas previamente.

Para modelar el juego, vamos a usar una lista de números positivos (sin el cero) para el mazo, que inicialmente deberemos generar y mezclar. De este mazo iremos sacando las cartas que daremos a cada jugador. Cada jugador tendrá las cartas con las que puede jugar (su mano) que también modelaremos como una lista. Tendremos, también, otra lista de números que corresponderá con la *mesa* que es donde inicialmente se pondrán cuatro cartas para comenzar el juego y donde los jugadores pondrán su carta de descarte cuando no tengan juego para armar. Por otro lado, los juegos realizados por cada jugador también los modelaremos como una lista de números (en este caso, sumarán quince). Por último, deberemos guardar los juegos que hizo cada jugador, para lo que usaremos una lista de juegos, que será ni más ni menos que una lista de listas de números... sí vamos a practicar mucho el uso de listas y ciclos.

> Para simplificar este primer trabajo, vamos a trabajar sin considerar los palos de las cartas (serán representadas solamente por números) y no vamos a considerar las escobas, ni las 70 (las reglas para determinar quién las tiene son bastante complejas). Vamos a considerar que siempre hay cuatro jugadores participando.

1. generarMazo(). Esta función no recibe ningún parámetro (no hace falta) y devuelve una lista conteniendo los valores de las cartas de los n mazos mezclados de manera aleatoria. Se puede utilizar la función de `Python` random.shuffle(a) del módulo random para mezclar las cartas.

2. repartir (mazo, jug1, jug2, jug3, jug4). Esta función modela la repartija de tres cartas para cada uno de los cuatro jugadores. Recibe como parámetro un mazo (una lista que debe tener al menos 12 cartas) y la lista de cartas de cada jugador (al principio vacías). Agrega las cartas correspondientes a las listas, y devuelve un mazo. Naturalmente, el mazo quedará con 12 cartas menos, que figuran en las listas de los jugadores.

   Ayuda: una_lista.pop(0) obtiene el valor del primer elemento de la lista una_lista, retirándolo también de esta última. Si la lista no tiene elementos, esta función dará un error.

3. iniciarMesa(mazo, mesa). Esta función modela la acción de poner las cuatro cartas iniciales en la mesa. Recibe un mazo, que debe tener por lo menos cuatro cartas, las agrega a las mesa retirándolas del mazo. Esta función no devuelve ningún resultado, sino que modifica sus parámetros de entrada.

4. juegosPosibles(jug, mesa): recibe la lista de cartas que están en la mesa y la lista de cartas que tiene el jugador en ese momento y debe generar la lista de todos los juegos posibles que puede realizar el jugador. Es decir, su resultado será una lista de listas de números. Cada juego deberá sumar 15 y se pondrá en el primer lugar de la lista la carta del jugador seguida de las utilizadas de la mesa. Es posible que no hubiera ningún juego para hacer, en ese caso se devolverá una lista vacía.

> Observar que la siguiente función genera todas las sublistas de una lista dada. La pueden incorporar para ayudar a resolver esta función.

```
def subListas(lista):
    """
    arma una lista de todas las sublistas de la lista dada
    """
    resultado = []
    for k in range(1, len(lista) + 1):
        for listita in combinations(lista, k):
            resultado.append(list(listita))
    return resultado
```

5. elegirMejor(juegos): recibe como entrada una lista como la generada en la función anterior (los posibles juegos disponibles para un jugador) y elige cuál es el *mejor* entre las opciones posibles. En base a las simplificaciones que hemos realizado, el criterio para elegir un juego por sobre otro corresponde únicamente con la cantidad de cartas involucradas. En caso de empate, pueden elegir cualquiera. Imaginamos que a esta altura podrás ver que, como está armado el diseño del programa, en esta función es donde se incluye la estrategia de juego de un jugador y es donde podríamos incluir una distinta si hubiéramos incluido todas las reglas de puntuación.

6. jugar(mesa, jug, basa): recibe la lista de cartas que están en la mesa, la lista de cartas que tiene un jugador en ese momento (que debe ser no vacía, es decir, aun tiene que tener cartas por jugar) y la lista de juegos realizados por el jugador hasta el momento (llamaremos a estas carta, la basa del jugador). Esta función debe usar la función juegosPosibles para obtener la lista de juegos posibles, luego usar esta lista como argumento de elegirMejor (siempre y cuando hubiera al menos un juego posible para hacer) con lo que sabrá cuál es la *mejor* opción. En base a eso, retira la carta seleccionada de la mano del jugador (que será la primera del juego) y las de la mesa para armar el juego y agregarlo a la basa del jugador (que será una lista de listas de números). Si no hay juego posible, deja una carta en la mesa (pueden elegir la primera que tenga el jugador en la mano o usar algún criterio más astuto).

7. jugarRonda(mesa, jug1, basa1, jug2, basa2, jug3, basa3, jug4, basa4): recibe la lista de cartas que están en la mesa y utiliza la función jugar con cada uno de los jugadores hasta completar la ronda. Es decir, que cada uno de los jugadores haya jugado sus tres cartas.

8. sumaPuntos(basa1, basa2, basa3, basa4): en esta función se reciben la lista de juegos que pudo hacer cada jugador y devuelve una lista con cuatro elementos que corresponde a la cantidad de puntos que hizo cada jugador. Como en nuestra versión reducida del juego solo consideramos la cantidad de cartas, a lo sumo va a ocurrir que uno solo de los jugadores tenga un punto, pero como también existe la posibilidad de empate, es posible que ninguno obtenga puntos.

9. **Programa principal**: el programa principal deberá definir las variables necesarias para el programa (las distintas listas necesarias) y luego repartir y llamar a la función jugarRonda hasta que no queden cartas en el mazo. Una vez terminado esta etapa, deberá usar la función quienGano para determinar el jugador que ganó. Deberá mostrar en pantalla las basas finales de cada jugador, así como cuál de ellos ganó.

10. **Optativo**: Supongamos que queremos saber si la posición en la lista de jugadores (qué lugar relativo de juego te toca) influye o no en ganar, ¿cómo haríamos con el programa que hicimos para poder responder esa pregunta?

Importante: en caso de que existan funciones de `Python` que resuelvan alguno de los ítems pedidos total o parcialmente (salvo aquellas mencionadas en el enunciado explícitamente), no es posible utilizarlas. Ante la duda, primero consultar.