Actividad 1 - Black Jack

Exactas Programa

Verano 2019

Vamos a escribir un programa `Python` que *juegue* a `Nano Jack`, un juego de cartas basado en el famoso *Black Jack*, pero con reglas un tanto relajadas.

En el juego `Nano Jack` se utiliza un mazo de cartas por cada jugador que participa en el juego todas juntas *mezcladas*. Cada jugador va tomando de a una carta y sumando sus valores. Si logra que las obtenidas *sumen* 21 gana, si se pasa pierde.

El objetivo de esta actividad será realizar un programa en `Python` que *simule* varios jugadores jugando al `Nano Jack`.

En nuestro juego, un mazo de cartas será representado por una lista de números enteros entre 1 y 13, repetidos 4 veces cada uno (uno por cada palo). Vamos a escribir funciones en `Python` que, combinadas, permitan jugar al `Nano Jack`:

1. `generar_mazos(n)`: Esta función recibe como parámetro n, un número entero positivo que representa la cantidad de mazos con los cuales se jugará (que dependerá del número de jugadores participantes). Como resultado, deberá devolver una lista conteniendo los valores de las cartas de los n mazos mezclados de manera aleatoria[1].

2. `jugar(m)`: recibe como parámetro un mazo m (que es una lista de números enteros entre 1 y 13) y realiza la mecánica de retirar cartas una a una hasta que la suma de los valores obtenidos sea mayor o igual a 21. El valor que devuelve esta función es la suma obtenida (aún si ésta fuera mayor a 21). Para que se puedan combinar luego varios jugadores, las cartas sacadas son retiradas del mazo de cartas, es decir, al terminar esta función, el mazo m deberá tener menos cartas (tantas como fueron sacadas para obtener el resultado de la función). Si el mazo no llegara a tener ninguna carta, esta función devolverá la suma obtenida (por ejemplo, devolvería 0 si el mazo no tuviera cartas).

   Ayuda: `una_lista.pop(0)` obtiene el valor del primer elemento de la lista `una_lista`, retirándolo también de esta última.

3. `jugar_varios(m, j)`: toma un mazo de cartas `m` y un número de jugadores `j`. Se trata de usar la función definida en el punto anterior (`jugar`) con cada uno de los jugadores de uno en uno y se almacenará en una lista los valores obtenidos para cada uno de los j jugadores.

   Ayuda: llamar sucesivamente a la función `jugar` con el mazo restante.

4. (*Optativo*) `ver_quien_gano(resultados)`: toma una lista de enteros, `resultados` que corresponde con la que devuelve la función `jugar_varios`. Se considera ganador al jugador que haya sumado 21. Se debe devolver una lista que indique para cada jugador si gano (con un 1), o si no (con un 0). Puede suceder que haya varios ganadores.

5. (*Optativo*) `experimentar(rep,n)`: toma un número entero positivo rep y otro n. Juega `Nano Jack` rep veces con n jugadores. Devuelve una lista con n valores que representan cuántas veces ganó el jugador correspondiente (el elemento 0 guardará cuántas partidas ganó el primer jugador, el elemento 1 guardará cuántas el siguiente y así sucesivamente).

Nota: en caso de que existan funciones de `Python` que resuelvan alguno de los ítems pedidos total o parcialmente, no es posible utilizarlas (salvo que esté explícitamente mencionado).

[1] Se puede utilizar la función de `Python` `random.shuffle(a)` del módulo `random` para mezclar las cartas.