Elementos Esenciales de Programación para Científicos en Formación

Taller Nº 0

Fecha de entrega: Jueves 14 de mayo de 2020, 12:00hs

En el juego `EP Jack` se utiliza un gran mazo de cartas formado a partir de mezclar varios mazos individuales. La dinámica del juego consiste en que cada jugador participante debe tomar cartas de forma consecutiva, sumando sus valores. Si la suma llegase a superar 21, perderá automáticamente. El ganador será aquel jugador que sume entre sus cartas un valor lo más cercano a 21, pero sin pasarse.

El objetivo será desarrollar una estrategia de 'Inteligencia Artificial' altamente sofisticada para jugar al `EP Jack`. Para esto se pide implementar las siguientes funciones en `Python`:

1. **generarMazo(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. Cada mazo contiene cuatro veces los números enteros entre 1 y 13, es decir una repetición por cada palo.
   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 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. Así, 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. Si la lista no llegase a tener elementos, esta función dará un error.

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 que va quedando luego de cada jugada.

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

4. **jugarMiedo(m):** es similar a jugar, pero implementa otra estrategia. En este caso, dejará de pedir cartas si la suma total obtenida es mayor o igual a 19.

5. **jugarBorracho(m):** es similar a jugar, pero implementa otra estrategia. En este caso, cada vez que saca una carta, pide un número aleatorio[2]. Si este valor obtenido es mayor a 0,5, vuelve a pedir una carta, sino deja de pedir. Si la suma de cartas obtenido supera 21, no pide más cartas.

6. (*Optativo*) **jugarSmart(m):** es similar a jugar, pero implementa otra nueva estrategia. Nuevamente se el pedir una carta estará basada en pedir un número aleatorio. En este caso, en lugar de compararlo contra 0,5, se lo comparará contra un valor que sea más alto a medida que la suma de cartas esté más cerca de 21.

   Ayuda: Se admite creatividad para implementar esta función. Una posible implementación podría ser obtener un número aleatorio. Luego, pedir directamente una carta si la suma de cartas es 0, pedir si el número aleatorio es myor que 0,25 si la suma es 5, mayor a 0,5 si la suma es 10, mayor a 0,75 si es 15..., y los valores intermedios podrían completarse proporcionalmente. De esta manera, a medida que se acerca a 21, será menos probable seguir pidiendo cartas.

7. **compararEstrategia(lista_jug):** recibe una lista valores numéricos entre 0 y 3. Donde 0 corresponde a utilizar la función jugar, 1 a jugarMiedo, 2 a jugarBorracho y 3 a jugarSmart. Por cada valor de la lista se debe llamar a la función correspondiente y almacenar el valor obtenido.
   El resultado es una lista que contiene los valores de cada una de las manos para cada jugador habiendo utilizado la estrategia indicada por el parámetro de entrada.
   Por ejemplo, si la función recibiera la lista [0,0,0,1] significa que hay cuatro jugadores, y los tres primeros usan la estrategia implementada en la función jugar, mientras que el cuarto usará jugarMiedo.

Nota: en caso de que existan funciones de `Python` que resuelvan alguno de los ítems pedidos total o parcialmente, no es posible utilizarlas. Ante la duda, primero consultar.

Condiciones de entrega:

- El trabajo es **individual**.
- El archivo fuente deberá tener comentarios.
- Se evaluará la correctitud del código producido y su claridad y legibilidad.
- Enviar el archivo fuente por correo electrónico a la lista de los docentes de la materia: **elemprog-doc@dc.uba.ar**.
- El mail deberá tener el siguiente *asunto*:
  "[Taller 0]: Gonzalez 666/21 o Gonzalez 27423214".
- De tener cursada presencial, debe entregarse **impreso** en la fecha y horario de entrega pautado.

[2]La función random.random() da un número pseudoaleatorio entre 0 y 1.