Algoritmos y Programación I

75.40

Trabajo práctico N°3

Facultad de Ingeniería, Universidad de Buenos Aires

Segundo cuatrimestre 2016

Alumnos: del Mazo, Federico - Silvestri, María Carolina

Legajo: 100029 - 99175

Práctica: Alan

E-mail: delmazofederico@gmail.com - silconito@gmail.com

Correctora: Jennifer Woites

Planteo del problema

En este trabajo tuvimos que realizar un código que nos permitiera simular el juego de cartas Uno, para jugar por consola. El juego cuenta con un mazo de 112 cartas (32 especiales, 76 normales, 4 especiales pensadas por nosotros), entre 2 y 4 jugadores (de los cuales 1 es el usuario y el resto son “maquinas”). Se empieza con 7 cartas, por turno se descarta (si se puede) una de la mano, y el que se queda sin cartas gana el juego.

Resolución

Desde un principio supimos que el problema tendría que ser orientado a objetos y a ser resuelto por métodos y atributos más que por funciones. Decidido esto, dividimos el programa en dos distintos archivos. Por un lado, el archivo de funciones, que haciendo uso de las clases armadas juega el juego mismo, por el otro, un archivo de clases, donde definimos las clases y sus atributos correspondientes.

Funciones

En cuanto al archivo de las funciones mismas, es el más fácil de los dos. Este lo hicimos con ciclos definidos por la cantidad de jugadores, ciclos indefinidos con la condición de que ningún jugador haya ganado, creaciones de instancias de los objetos ya definidos, validaciones de las interacciones del usuario y demás. Con todo esto el concepto principal es el más sencillo, recorrer el “ciclo” de jugadores indefinidamente, pidiéndole a cada jugador que usen del método “jugar”, donde, ya definido en el segundo archivo, se les pedirá que cartas jugar, si quieren robar del mazo, etc.

Objetos

El verdadero problema reside en el archivo de clases. Donde y cuando llamar a cada método y/o atributo, y más importante, que pasar y devolver como parámetros, siendo todo el programa tan entrelazado donde todo lo que pasa en el juego depende de todos los agentes en él; si el mazo esta vacío, tiene que ‘hablar’ con el pozo, cuando un jugador tira una carta especial, el efecto depende del jugador a su derecha, es decir, llama al ciclo, etcétera.

Carta

La primera decisión tomada fue la de no hacer dos clases distintas para una carta especial y para una carta numérica. Una carta especial no deja de ser una carta. Por lo tanto, los atributos de la carta son el número, el color, y la especialidad. Si alguna carta carece de número (las especiales), de color (los comodines) o de especialidad (las numéricas), se pasa el atributo como None.

Además de las cartas del juego se nos propuso implementar cartas propias. Las cartas nuestras son:

- Carta blanca: Una carta trivial, no hace nada. Se puede tirar en cualquier momento y cualquier carta puede apilarse arriba de esta. Estratégicamente, la carta más poderosa del juego. Verdaderamente, la carta más aburrida del juego.
- Carta Guasón: Hartos de que todas las cartas sean a beneficio del jugador, el príncipe del crimen viene directo de ciudad Gótica a hacer preso al jugador de sus propias cartas. El que la tira debe sacar dos cartas del mazo

Métodos de Carta:

- Encaja: Recibe por parámetro la propia carta y una segunda carta (lo que sería, la carta en mi mano que deseo tirar y la carta en el pozo). Devuelve un booleano que, dependiendo de las reglas del juego, confirma si la carta puede ser tirada o no.
- Aplicar especialidad: A nuestro parecer, el método más difícil del programa. Recibe al jugador, al mazo y al iterador actual. Si la carta es Blanca, simplemente imprime un mensaje. Si la carta es Guason, el jugador saca cartas del mazo. Si la carta es Saltear, con el método __next__ del iterador, forzamos al ciclo a saltearse un turno. Si la carta es Invertir Sentido, llamamos al método del iterador del ciclo que se detalla más adelante. Por último, si la carta es de sumarle a otro, el

jugador siguiente, pero sin avanzar al iterador (por lo tanto, se hace uso del módulo copy) saca cartas del mazo, y luego, como en Saltear, se avanza al iterador.

- Es comodín y pedir color: Los comodines son cartas sin color, por ende, cada vez que uno es lanzado hay que pedirle al jugador que color declarará. Para esto pedir color recibe como parámetro al jugador, para confirmar si es un jugador usuario (pregunta interactuando) o un jugador maquina (con 'choice' del módulo 'random' se elige un color al azar)

Mazo de cartas

Heredado de una Lista Enlazada (y no una nativa de Python, por eficiencia) con métodos de insert y pop se implementa esta clase donde los métodos son de agregar carta (simplemente un renombramiento del append de la clase base, por comodidad y unicidad), sacar carta (con el módulo random, usa el pop con una carta al azar) y remezclar, que es llamada cuando no hay más cartas para sacar y, recibiendo el pozo, mantiene la carta del tope del pozo y vuelve a agregar todas las cartas de este.

Pozo

Heredado de una Pila, ya que solo necesita poder apilar cartas arriba, sin importar lo del medio, solo el 'tope'. Implementamos esta clase con solo un método agregado que es el de la carta inicial, que saca una carta del mazo y la apila en el pozo.

Jugador

Al igual que con las cartas, decidimos no dividir en dos la clase y hacer que todo jugador, maquina o usuario, sea jugador. El jugador, pensado análogamente como un nodo de la lista enlazada, tiene como atributos el nombre, la mano de cartas (lista de Python), el jugador a la derecha y el jugador a la izquierda. Un último atributo es el de comprobar si el jugador esta 'vivo' o no.

Métodos principales de Jugador:

- Ver mano: Para imprimir las manos de los perdedores una vez terminado el juego
- Gano: devuelve por booleano si la mano tiene cartas o no, es decir, si el jugador gano la partida o no.
- Jugar, pedir acción y jugar carta: El núcleo del juego dividido en 3 métodos distintos para mejor lectura. Primero, si el jugador es el usuario, se le pide, imprimiéndole todas sus cartas y dándole la opción de robar del mazo, que carta tirar. De lo contrario, si el jugador es una máquina, se recorre la lista hasta encontrar una carta que encaje y está es la carta seleccionada a tirar. Luego, con el método de jugar la carta misma, si se le pasa por parámetro None, este método lo toma como que el jugador decidió (o se vio forzado) sacar una carta del mazo (y luego, recursivamente, le da la opción de tirarla), si recibe por parámetro una verdadera carta, esta carta la apila en el pozo y la saca de la mano del jugador. Finalmente, aplica la especialidad con el método de la carta misma

Ciclo e Iterador Ciclo

El ciclo fue pensado como una lista doblemente enlazada de jugadores (nodos) y a su vez circular. Doblemente enlazada, para poder moverse para ambos lados (que es necesario para cuando un jugador usa la carta de invertir el sentido de la ronda) y circular para recorrerla infinitamente, sin principio ni fin, y solo rompiendo la partida cuando un jugador gana el juego.

El iterador, una clase privada del ciclo, tiene como atributo para que sentido va (derecha True, izquierda False) y como método el invertir sentido, que simplemente da vuelta los valores de este atributo. El método next de la clase depende del sentido de la ronda, así viendo para que lugar avanzar

El ciclo tiene como atributos un primer nodo (que recibe por parámetro y es siempre el jugador usuario), un último nodo, un largo.

Métodos principales de Ciclo:

- Invariación: Llamado cuando es construido y cuando se le agrega un jugador, este método fuerza que el jugador a la derecha del ultimo sea el primero y que el jugador a la izquierda del primero sea el último, forzando así a que sea una lista circular
- Agregar jugador: Agrega un jugador a la derecha del ultimo y luego llama a la invariación constante de la clase