Primer obligatorio

Programación I

Federico Becoña
Programación I
Prof. Gonzalo Pennino
Prof. Federico Wagner

25/05/2018

Planteo del problema:

La propuesta era programar el juego “4 en línea” en Python. Este consiste en que dos participantes coloquen fichas en un tablero de seis filas por siete columnas, de a una por turno, eligiendo en cada uno de ellos la columna en la que quiere depositar una pieza y esta va a la posición correspondiente, desde arriba, al último casillero vacío de la misma.

El objetivo es conseguir una línea de cuatro piezas juntas ya sea horizontal, vertical o diagonal para así ganar y finalizar el juego. Si se completa el tablero con fichas y ninguno consigue la victoria, la partida termina en empate.

Además, hay que agregar la función de dudas que consiste en que si un jugador la elige escribiendo la palabra consulta debe aparecer en pantalla el nombre del creador y su correo electrónico para que este le pueda enviar sus opiniones o problemas relacionados con el juego.

Análisis de soluciones alternativas:

En todas las soluciones posibles se requiere de una plantilla que funcione de tablero en el cual se vayan registrando las fichas colocadas durante la partida, una forma para poner las piezas en los lugares correspondientes cuando se hacen las jugadas, una manera de cambiar de turno y un modo de evaluar si hay un ganador en cada uno de los estados intermedios de la partida.

Se podría utilizar una clase que creara al objeto plantilla cuya función sería ir almacenando los estados intermedios de juego. Esta debería hacerse a partir de listas dado que así se podría aprovechar la característica que estas tienen de ser modificables. De esta manera, basta con crear un tablero con espacios vacíos, en cada turno encontrar el índice del casillero en el que se debía colocar la nueva ficha y modificar la lista allí por la pieza del jugador correspondiente.

El tablero podría estar formado a partir de seis listas con siete espacios cada una, lo cual tendría como problema que el código de posicionamiento de fichas y el de la evaluación de ganadores o empate sería repetitivo ya que no podría iterar sobre las filas con un for(): o un while(): para aplicarles a todas ellas unas mismas sentencias de código. Otra manera sería utilizar una lista compuesta a su vez por seis listas de siete objetos con un espacio cada uno, esta no tiene el inconveniente de la opción mencionada antes ya que esta vez sí se podría iterar tanto en las filas como en los objetos dentro de las mismas.

Ejemplo del problema mencionado anteriormente:

```
class Cuatro_en_linea():
    def __init__(self):
        self.fila1 = list(" "*7)
        self.fila2 = list(" "*7)
        self.fila3 = list(" "*7)
        self.fila4 = list(" "*7)
        self.fila5 = list(" "*7)
        self.fila6 = list(" "*7)
        self.jugando = 0

    def movimiento_de_fichas(self,columna):
        if self.jugando % 2 == 0:
            self.jugando = self.jugando + 1
            if self.fila6[int(columna)-1] == " ":
                self.fila6[int(columna)-1] = "X"
            elif self.fila5[int(columna)-1] == " ":
                self.fila5[int(columna)-1] = "X"
            elif self.fila4[int(columna)-1] == " ":
                self.fila4[int(columna)-1] = "X"
            elif self.fila3[int(columna)-1] == " ":
                self.fila3[int(columna)-1] = "X"
            elif self.fila2[int(columna)-1] == " ":
                self.fila2[int(columna)-1] = "X"
            elif self.fila1[int(columna)-1] == " ":
                self.fila1[int(columna)-1] = "X"
        elif self.jugando % 2 != 0:
            self.jugando = self.jugando + 1
            if self.fila6[int(columna)-1] == " ":
                self.fila6[int(columna)-1] = "O"
            elif self.fila5[int(columna)-1] == " ":
                self.fila5[int(columna)-1] = "O"
            elif self.fila4[int(columna)-1] == " ":
                self.fila4[int(columna)-1] = "O"
            elif self.fila3[int(columna)-1] == " ":
                self.fila3[int(columna)-1] = "O"
            elif self.fila2[int(columna)-1] == " ":
                self.fila2[int(columna)-1] = "O"
            elif self.fila1[int(columna)-1] == " ":
                self.fila1[int(columna)-1] = "O"
```

```
class Cuatro_en_linea():
    def __init__(self):
        self.tablero = []
    def plantilla(self):
        for filas in range(6):
            self.tablero.append([])
            for espacios_en_filas in range(7):
                self.tablero[filas].append(" ")
    def posicion_de_ficha(self,columna):
        for filas in range(6):
            if self.tablero[5-filas][int(columna)-1] == " ":
                return 5-filas
    def movimiento_de_fichasX(self,columna):
        self.tablero[self.posicion_de_ficha(columna)][int(columna)-1] = "X"
    def movimiento_de_fichasO(self,columna):
        self.tablero[self.posicion_de_ficha(columna)][int(columna)-1] = "O"
```

En las imágenes también se observan dos maneras para que el juego cambie la ficha al cambiar de turno. Para esto se podría poner una variable que inicialmente fuera cero y que se fuera incrementando de a uno por jugada para que cuando esta fuera un número par se diera el turno al jugador 1 y al ser impar al jugador 2 o al revés. También se podría utilizar una función que diera la posición, que según las reglas del juego debía tomar cada nueva ficha, en relación con el número que se ingrese, la cual se podría reutilizar para cuando juegue el primero y el segundo con otras dos funciones que coloquen un cierto tipo de pieza dependiendo del jugador.

La evaluación de los estados intermedios se podría hacer mediante una única función o a través de varias distinguiendo la vertical, la horizontal y la diagonal. La segunda sería mejor dado que facilita la corrección de errores al poder colocar contadores en el constructor y ver con "setters" y "getters" porque estos ocurren de forma más sencilla.

Para comprobar si en cada estado intermedio un jugador consigue poner cuatro fichas juntas se podrían utilizar los índices correspondientes a las posiciones del tablero para evaluar si los elementos que se encuentran en ellas siguen algún patrón que de un ganador o sino también en ciertos casos se podría ver si la cadena "XXXX" o la "OOOO" están en una lista que representara una fila, una columna o una diagonal, dependiendo de cómo se haga la plantilla del tablero.

Por último, se necesitarían una o más funciones para ejecutar la partida de manera tal de que los participantes jueguen de forma intercalada, si estos no colocan un número correspondiente a una columna disponible les salga un mensaje de error salvo que estos hayan ejecutado la funcionalidad "consulta" del programa, con la cual debería aparecer en pantalla el correo y el nombre del creador para que los jugadores les envíen sus dudas.

Justificación de la solución elegida:

Utilicé una clase para hacer un objeto plantilla que funcionara de tablero dado que de esta manera iba a poder almacenar la información de los estados intermedios del tablero entre jugada y jugada de una forma sencilla para utilizar en el código y también muy organizada.

Agregue métodos relacionados con el movimiento de las fichas en cada turno, la evaluación de si un jugador ganó o si hubo empate en cada estado intermedio de la partida, si la columna en la que quiere ingresar una ficha un jugador está llena, la funcionalidad "consulta" del juego y dos más para que junto con otra que se encuentra fuera de la clase se pueda reproducir la partida. De esta manera logré una estructura en la que era fácil de verificar por qué sucedían errores agregando funciones get(): y set(): y poniendo por ejemplo contadores en el constructor.

La plantilla del tablero la realice a partir de una lista que tenía dentro seis listas conformadas a su vez por siete objetos que contenían un espacio cada uno. Así se podían hacer iteraciones tanto en las filas como en los elementos de las mismas y se lograba hacer el código muy conciso utilizando funciones como for(): y while():.

En todo lo anterior intenté hacer la mayor cantidad de funciones posibles para que no se tuvieran que repetir conjuntos de sentencias y de este modo conseguir que el código del programa quedara concreto.

Desarrollo de la solución:

Comencé designando una clase sin atributos para crear el objeto "tablero" en el cual acumular las jugadas de manera organizada. La plantilla la hice a partir de una lista con seis listas, que representaban las filas del tablero, conformadas por espacios en cada uno de sus objetos dado que así iba a poder hacer iteraciones tanto en las filas como en sus elementos. De esta manera se consigue que no se tengan que reiterar líneas de código haciéndolo más conciso y sencillo.

```
class Cuatro_en_linea():
    def __init__(self):
        self.tablero = []
        for filas in range(6):
            self.tablero.append([])
            for espacios_en_filas in range(7):
                self.tablero[filas].append(" ")
```

Utilicé un módulo para poder encontrar el índice dentro de la plantilla que corresponda a la posición en donde hay que colocar la ficha cuando un concursante hace una nueva jugada. Esta está compuesta por la columna mencionada por el jugador (si esta es válida) y el primer casillero vacío de la misma comenzando desde abajo.

Luego utilicé otras dos funciones para que cuando se ejecutaran cambiaran el espacio de la lista en el lugar dado por el módulo anterior por una ficha que depende de si se trataba del jugador 1 o del 2. Con el primero se utiliza una "X" y con el segundo una "O".

Como los índices de las listas comienzan desde el cero y las opciones de columnas válidas para los participantes arrancan desde el uno, podemos observar que estás se encuentran zafadas lo cual se soluciona poniendo que al valor donde se quiere poner la ficha se le restara uno para luego colocarla en el lugar más bajo de esa columna de la plantilla viéndolo desde arriba.

```
def movimento_de_fichasX(self,columna):
    self.tablero[self.posicion_de_ficha(columna)][int(columna)-1] = "X"
def movimento_de_fichasO(self,columna):
    self.tablero[self.posicion_de_ficha(columna)][int(columna)-1] = "O"
```

Posteriormente coloqué tres métodos vinculados con las evaluaciones de los estados intermedios del tablero para comprobar si había un ganador.

La primera evaluaba si había cuatro fichas iguales juntas de forma horizontal lo cual hice viendo si la cadena "XXXX" o la "OOOO" se encontraban en alguna fila. Esto lo pude hacer por cómo había armado la plantilla, una lista con seis listas dentro que correspondían a las filas.

```
def evaluacion_horizontal(self):
    for filas in self.tablero:
        if "XXXX" in "".join(filas):
            return "finalizador_1"
        if "OOOO" in "".join(filas):
            return "finalizador_2"
```

Las otras dos evaluaban si había un ganador de forma vertical o diagonal y esto lo realice a partir de observar si en alguna de las líneas del tablero que tenían una dirección de esos estilos había cuatro fichas juntas de un mismo jugador en cada estado intermedio de la partida utilizando varios for(): y contadores para ambos tipos de piezas.

En la "evaluación_vertical():" el primer for(): recorre todas las columnas de la plantilla, el segundo solamente las filas iniciales en las que puede encontrarse la primera ficha comenzando desde arriba que forme un cuatro en línea vertical y el tercero recorre desde ese casillero las tres filas siguientes de la misma columna. De esta forma, se evalúa si a partir de encontrar una primera ficha después se pueden encontrar las otras tres y comprobar la existencia de un cuatro en línea vertical.

En la función mencionada anteriormente, como el ultimo for(): tiene siempre cuatro iteraciones, si hay tres fichas de un jugador pero una en medio del otro no se va a dar un ganador dado que en una de sus iteraciones se sumará uno al contador de las fichas del adversario imposibilitando que el de dicho jugador llegue a cuatro, quedandose en tres y no activando el if(): que se encuentra debajo. Además, cabe de destacar que los contadores se reinician al cambiar de posición inicial a partir de la cual se estudia si hay más casilleros junto a él con fichas iguales para evitar errores.

```
def evaluacion_vertical(self):
    contador_x = 0
    contador_o = 0
    for columna in range(7):
        for fila_inicial in range(3):
            for fila_final in range(4):
                if self.tablero[fila_inicial+fila_final][columna]=="X":
                    contador_x += 1
                if self.tablero[fila_inicial+fila_final][columna]=="O":
                    contador_o += 1
            if contador_x == 4:
                return "finalizador_1"
            if contador_o == 4:
                return "finalizador_2"
            contador_x = 0
            contador_o = 0
```

La "evaluación_diagonal():" consiste prácticamente en lo mismo que la anterior solo que como las casillas que se evalúan a partir de la primera siguen una dirección diagonal se le sumo el valor dado por el último for(): también a las columnas para que funcionara correctamente con las diagonales de izquierda a derecha y luego se hizo lo mismo con aquellas de derecha a izquierda restando el valor de que sale de dicho for(): al de la columna.

El range(): de la columna inicial varía dado que en el caso de las diagonales de izquierda a derecha, las fichas iniciales comenzando desde la izquierda solo pueden encontrarse en las primeras tres columnas y en el de derecha a izquierda estas se pueden ubicar, con el mismo criterio, desde la tercera columna hasta la última pero en ambos casos sus filas tendrán que estar entre las primeras tres.

```
def evaluacion_diagonal(self):
    contador_x = 0
    contador_o = 0
    for fila_inicial in range(3):
        for columna_inicial in range(4):
            for diagonal in range(4):
                if self.tablero[fila_inicial+diagonal][columna_inicial+diagonal]=="X":
                    contador_x += 1
                if self.tablero[fila_inicial+diagonal][columna_inicial+diagonal]=="O":
                    contador_o += 1
            if contador_x == 4:
                return "finalizador_1"
            if contador_o == 4:
                return "finalizador_2"
            contador_x = 0
            contador_o = 0
        for columna_inicial in range(3,7):
            for diagonal in range(4):
                if self.tablero[fila_inicial+diagonal][columna_inicial-diagonal]=="X":
                    contador_x += 1
                if self.tablero[fila_inicial+diagonal][columna_inicial-diagonal]=="O":
                    contador_o += 1
            if contador_x == 4:
                return "finalizador_1"
            if contador_o == 4:
                return "finalizador_2"
            contador_x = 0
            contador_o = 0
```

Además, coloque una función dentro de la clase llamada empate que evalúa si en todos los casilleros hay fichas o si alguno sigue con el estado que obtiene al comenzar la partida conteniendo un espacio.

```
def empate(self):
    contador = 0
    for columnas in range(7):
        for filas in range(6):
            if self.tablero[filas][columnas]!= " ":
                contador += 1
    if contador == 42:
        return "finalizador_E"
```

Hice que estas funciones devolvieran cadenas de caracteres y no “True” o “False” dado que al ser bastantes me pareció más conveniente ponerles que devolvieran distintas cosas para saber más fácilmente de dónde venían cada instrucción.

Utilice un módulo de evaluación que está relacionado con los anteriores devolviendo si algún jugador ganó y quien es o si hay empate.

```
def evaluacion(self):
    if self.evaluacion_horizontal()=="finalizador_1" or self.evaluacion_vertical()=="finalizador_1" or self.evaluacion_diagonal()=="finalizador_1":
        return "gano_1"
    elif self.evaluacion_horizontal()=="finalizador_2" or self.evaluacion_vertical()=="finalizador_2" or self.evaluacion_diagonal()=="finalizador_2":
        return "gano_2"
    elif self.empate()=="finalizador_E":
        return "empate"
```

Luego realicé dos funciones más dentro de la clase. Una que evalúa si la columna en la que se quiere poner una ficha está llena, para luego con otra función darle la oportunidad al jugador de cambiar su elección, a partir de hacer un append(): a una lista vacía con los elementos de cada columna, reiniciándose al cambiar la misma, y evaluar si en alguna de estas hay espacios vacíos. La otra tiene como funcionalidad imprimir el tablero con un formato que sea fácil de comprender para los jugadores para lo cual utilice la función join(): y se separé cada objeto de las listas con una barra "|".

```
    def llena(self,entrada):
        columna = int(entrada)-1
        lista = []
        for fila in range(6):
            lista.append(self.tablero[fila][columna])
        if " " not in lista:
            return "completa"
    def imprimir_tablero(self):
        for fila in self.tablero:
            print(" "*32 + "|"+"|".join(fila)+"|")
```

Por último, coloque dos métodos más y una función fuera de la clase para reproducir la partida en la cual se imprima el tablero actualizado y se evalúe si hay un ganador o empate entre cada turno. Además, en estas se utilizan sentencias para, en base a los métodos hechos anteriormente lograr, que si se escribe la palabra “consulta” ya sea en mayúsculas o minúsculas se indique el nombre del creador del juego y su correo electrónico para que el jugador envíe sus dudas y si no se hace esto ni y el participante no menciona una columna válida o escribe una que esté llena se le pregunte nuevamente para que cambie su elección.

```
def una_jugada(self,jugador):
    print("-"*79)
    print("Para la opción de problemas o consultas adicionales ingrese la palabra consulta")
    entrada = input("-Ingrese el número " + jugador + " : ")
    return entrada
```

```
def proceso(self,jugador):
    entrada = self.una_jugada(jugador)
    while entrada not in ["1","2","3","4","5","6","7"] or self.llena(entrada)=="completa":
        if entrada.lower()!="consulta":
            print(" "*13 + "Esa columna no está disponible, intente otra vez " + jugador)
            self.imprimir_tablero()
            entrada = self.una_jugada(jugador)
        else:
            print()
            print("Creador: Federico Becoña, Mail: fmbecona@gmail.com. "+"\n"+ jugador +
                  ", usted puede enviarnos sus dudas a esa casilla de correo electronico.")
            print()
            self.imprimir_tablero()
            entrada = self.una_jugada(jugador)
    if jugador == "Jugador 1":
        self.movimento_de_fichasX(entrada)
    elif jugador == "Jugador 2":
        self.movimento_de_fichasO(entrada)
    evaluador = self.evaluacion()
    self.imprimir_tablero()
    if evaluador == "gano_1":
        print("-"*83 + "\n"*2 + " "*27 + "¡Ha ganado el jugador 1!" + "\n")
        return "ganador"
    elif evaluador == "gano_2":
        print("-"*83 + "\n"*2 + " "*27 + "¡Ha ganado el jugador 2!" + "\n")
        return "ganador"
    elif evaluador == "empate":
        print("-"*83 + "\n"*2 + " "*33 + "¡Es un empate!" + "\n")
        return "ganador"
```

```
def reproducir():
    partida = Cuatro_en_linea()
    partida.plantilla()
    print(" "*27 + "¡Bienvenido a 4 en línea!" + "\n" + " "*29 + "Que comience el juego")
    partida.imprimir_tablero()
    para_iterar_jugadas = 0
    while partida.proceso("Jugador 1")!= "ganador" and partida.proceso("Jugador 2") != "ganador":
        para_iterar_jugadas += 1
    return "\n" + " "*28 + "El juego ha finalizado"
print(reproducir())
```

Hay que destacar que la “entrada” es una cadena de caracteres y en el while(): se la compara con una lista compuesta con cadenas de caracteres con los números del 1 hasta el 7 dado que si la pasaba a int(): y luego ponía que siguiera el bucle si la “entrada” era menor a 1 o mayor a siete me salía un error cuando en el juego cuando apretaba la tecla “enter” porque no podía convertir ese input a un int():.

```
Traceback (most recent call last):
  File "C:\Users\estudiante.fit\Desktop\4 EN LÍNEA FINAL!!!.py", line 135, in <module>
    print(reproducir())
  File "C:\Users\estudiante.fit\Desktop\4 EN LÍNEA FINAL!!!.py", line 132, in reproducir
    while contador_jugadas <= 21 and partida.proceso("Jugador 1")!= "ganador" and partida.proceso("Jugador 2") != "ganador":
  File "C:\Users\estudiante.fit\Desktop\4 EN LÍNEA FINAL!!!.py", line 99, in proceso
    while int(entrada)>7 or int(entrada)<1 or self.llena(entrada)=="completa":
ValueError: invalid literal for int() with base 10: ''
```

Por último en la función “reproducir():” es en la que se crea el objeto para la partida utilizando la clase anteriormente mencionada y se decreta una buena parte de lo que los jugadores ven, exceptuando los input colocados para que los jugadores ingresen sus jugadas en la función “una_jugada():”, con un mensaje de bienvenida y otro de finalización del juego cuando alguno de los participantes ganó o si se produjo un empate entre ellos. En esta también se dispone la iteración de los turnos utilizando un while(): que hasta que se decreté un ganador o haya empate sumará uno a una variable cuya función es hacer funcionar la estructura sin que se produzcan errores, no tiene una funcionalidad en sí misma.

Conclusiones:

En un primer momento no me resultó muy difícil la tarea, pero me di cuenta de que la manera en la que había hecho el programa no era la mejor entonces busqué cambiarla y complejizarla con lo que también se hizo más difícil de programar.

Estuvo motivante, estuve un par de días concentrado con esto porque me parecía un trabajo interesante y siempre estaba intentando mejorarlo. Además, estuvo bueno que se lo pude mostrar a gente de otros ámbitos y pensaban que detrás del programa había una estructura mucho más compleja de la que realidad había, con esto que pude ver lo que he aprendido en este curso dado que hace unos meses hubiera pensado igual que ellos.

El planteo estuvo bueno dado que se trata de un juego bastante conocido y para hacerlo se ponen a prueba casi todos los conocimientos que hemos dado hasta el momento.

Una lección aprendida es que a veces cuando te trancas es mejor dejar el programa y verlo luego con la cabeza despejada para que se te ocurran cosas distintas y así solucionar los problemas. También, durante el trabajo me di cuenta de la diferencia que hay en cuanto a la eficiencia del código al utilizar funciones que sean reutilizables y no hacerlo.