¿Cómo hacer un ta-te-ti en Python?

Hacer un ta-te-ti básico en Python no requiere mucho conocimiento sobre el lenguaje en sí, el desafío consiste en cómo encarar el proyecto y cuál será la lógica con la que el juego va a funcionar.

La lógica general es sencilla, juegan el usuario y la computadora en un bucle infinito que finaliza si se cumple alguna de las siguientes condiciones

1. El usuario completa una línea
2. La computadora completa una línea
3. El tablero se completa sin que haya un ganador

Hacer un diagrama de flujo es primordial para entender de forma gráfica cada etapa de una aplicación y los caminos que puede tomar en base al resultado de cada algoritmo que se programe.

Diagrama

Descripción generada automáticamente

La solución aplicada a este caso en particular fue dividir toda la aplicación en algoritmos independientes para simplificar la lectura del código, separando cada algoritmo en un módulo distinto. Un algoritmo es una sucesión de líneas de código, recibe datos de entrada, los procesa de alguna manera y genera datos de salida. Cada algoritmo se convierte en una función separada que hace una tarea específica como por ejemplo la jugada del usuario, la validación de ganador, etc. Tener un sistema modular permite separar el código en fragmentos más chicos que residen en archivos independientes, cada archivo en este caso contiene una función que hace una tarea específica.

Al separar el código en algoritmos independientes se simplifica también el diagrama de flujo para disponer de uno general y el resto de los procesos en diagramas separados, la idea es tratar a toda la aplicación como un conjunto de sub-aplicaciones independientes que van a recibir datos como entrada y van a retornar datos de salida procesados y listos para pasar a la siguiente sub-aplicación en un ciclo que se va a repetir hasta que el juego se termine.

Para representar el tablero elegí usar una lista porque me pareció lo más simple debido a la sencillez del juego, la idea fue inicializarla cuando el juego empieza con valores enteros del 0 al 8 y por cada jugada reemplazar el valor de la celda jugada por una “X” para el caso del usuario y una “O” para el caso de la computadora. No puedo usar una tupla porque es de solo lectura asique mis opciones eran un diccionario o una lista.

Al iniciar el juego se muestra en la consola una grilla de 9 posiciones numeradas, cada posición representa una celda válida para que el usuario y la computadora hagan su jugada. Como la aplicación carece de interfaz gráfica se usa teclado para ingresar el número de celda que se quiere jugar, entonces el usuario debe hacer su jugada indicando un número del 1 al 9.

Texto

Descripción generada automáticamente

El algoritmo responsable de este proceso se llama jugada\_usuario y consiste en un bucle infinito que recibe el valor ingresado por el usuario y hace tres validaciones:

1. Que el valor ingresado sea un número
2. Que el valor ingresado esté dentro del rango 1-9 (los números de celdas disponibles para jugar)
3. Que la celda indicada no haya sido jugada previamente

El juego mostrará el error correspondiente y volverá a pedir la jugada al usuario cada vez que se cumpla alguna de las tres condiciones anteriores, si la jugada es válida entonces el proceso reemplaza el número correspondiente de la lista de jugadas por una “X”. Si bien el tablero se muestra con números del 1 al 9, la lista contiene valores del 0 al 8, de esta forma puedo cambiar el valor de la lista llamándolo por el número de índice para asignar la jugada.

El diagrama de flujo se ve de la siguiente manera:

Diagrama

Descripción generada automáticamente

Una vez que el usuario hizo su jugada es momento de chequear si hay un ganador, empate, o ninguna de las anteriores y el juego sigue su curso.

Para resolver este problema armo una función valida\_ganador que recibe la lista de jugadas y evalúa una serie de condiciones que pueden ocurrir para cada uno de los casos posibles:

El usuario gana cuando:

|  |  |  |
| --- | --- | --- |
| 1) Completa una fila |  |  |
|  |  |  |
|  |  |  |
| 2) Completa una columna |  |  |
|  |  |  |
|  |  |  |
| 3) Completa una diagonal |  |  |
|  |  |  |

La computadora gana cuando:

|  |  |  |
| --- | --- | --- |
| 1) Completa una fila |  |  |
|  |  |  |
|  |  |  |
| 2) Completa una columna |  |  |
|  |  |  |
|  |  |  |
| 3) Completa una diagonal |  |  |
|  |  |  |

El empate se declara cuando todos los valores de la lista son alfanuméricos y no se cumple ninguna de las condiciones anteriores.

La utilidad de poner estas evaluaciones dentro de una función es escribir un proceso general que pueda ser aplicado no sólo a las jugadas del usuario sino también a las que hace la computadora, de esta forma disminuye la cantidad de líneas de código, la aplicación es más liviana y mejora el rendimiento.

Suponiendo que no haya un ganador luego de la jugada del usuario el sistema pasa a generar la jugada de la computadora que se realiza a través de la función jugada\_compu, y para hacer la jugada va a chequear primero como fue jugando el usuario. para el caso puntual del juego recién iniciado donde el usuario solo hizo una sola jugada, el algoritmo no hace validaciones de ningún tipo, genera un valor entero aleatorio entre 0 y 8 y chequea si el valor obtenido es una celda libre para jugar, si la celda está libre se marca con una “O”, si por el contrario ya está ocupada se vuelve a generar un nuevo valor aleatorio y se chequea nuevamente si la celda está ocupada, este ciclo se repite indefinidamente hasta que el número aleatorio es válido para jugar.

A partir de la segunda jugada, el usuario ya muestra una estrategia para conseguir la línea y entonces la función hace un chequeo de cuales fueron las jugadas del usuario para determinar cuál será la próxima jugada probable y completa la celda con una “O” para impedir que el usuario complete la línea.

Para hacer esta tarea se recorre cada elemento de la lista y se chequean si las siguientes combinaciones de celdas están ocupadas por una “X”, si alguna de estas condiciones se cumple, la computadora juega con una “O” la celda faltante para completar la línea de “X”.

Para las filas:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Para las columnas:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Para las diagonales:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Dependiendo de como haya ido jugando el usuario puede darse el caso donde la computadora tenga dos opciones diferentes para impedir una línea del usuario, en tales casos la jugada es la primera validación que se encuentre como cierta. Si no se cumple ninguna de estas condiciones, la computadora genera un nuevo entero aleatorio válido y libre para hacer su jugada.

El diagrama de flujo para la jugada de la computadora sería el siguiente:

Diagrama

Descripción generada automáticamente

Al terminar el turno de la computadora se vuelve a ejecutar la función valida\_ganador para determinar el estado del juego.

El ciclo completo de jugada\_usuario, valida\_ganador, jugada\_compu y valida\_ganador nuevamente se repite en un bucle infinito que solo se termina si hay un ganador o el tablero se completa y hay un empate.

Una vez que el juego se termina, se muestra el resultado del juego y se pregunta si se quiere volver a jugar. En caso afirmativo se reinicia la lista de jugadas, se muestra el tablero vació y todo el ciclo vuelve a comenzar, si la respuesta es “N” el sistema muestra un mensaje de despedida y termina su ejecución. La aplicación válida la entrada del usuario y controla si la respuesta es “S” o “N”, si se ingresa una opción inválida, se muestra un mensaje de error y se vuelve a pedir al usuario la respuesta en otro bucle infinito que se repite hasta que la entrada sea alguna de las dos opciones posibles. Para evitar problemas con las mayúsculas, la entrada del usuario se convierte directamente a mayúsculas antes de validar la respuesta.

Se incluye un parámetro de nivel dentro del código que incluye dos opciones, nivel 1 y 2. El nivel 1 es un nivel “fácil” donde la computadora no valida las jugadas del usuario y cada vez que le toca jugar solo genera un valor aleatorio válido, el nivel 2 sería el nivel “intermedio” donde la computadora va controlando las jugadas del usuario y trata de anular las líneas que están por completarse. Para cambiar el nivel de juego se establece el valor correspondiente para la variable nivel antes de ejecutar la aplicación.

Esta es mi primera versión de ta-te-ti, un juego donde busqué aplicar los aspectos básicos de Python como bucles, condicionales, tipos de datos, funciones y módulos.

De a poco se irán sumando mejoras, el proyecto se está haciendo mediante el método PACER, la versión inicial del juego incluía todo el código en un solo archivo principal, esta nueva versión es el primer remanufacturado del original donde busqué hacer la aplicación modular para mejorar la legibilidad.

Repositorio github:

<https://github.com/martintokman/ta-te-ti>