**MANUAL TÉCNICO**

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

Este fragmento de código importa dos módulos en Python:

- `tkinter as tk`: Este importa el módulo `tkinter`, que es una biblioteca estándar de Python para crear interfaces gráficas de usuario (GUI). Al importarlo con el alias `tk`, podemos acceder a sus clases y funciones utilizando el prefijo `tk.`.

- `random`: Este importa el módulo `random`, que proporciona funciones para generar números pseudoaleatorios. Se usa para realizar elecciones aleatorias en el juego, como determinar el próximo movimiento de la máquina en el juego de Tres en Raya.

Texto

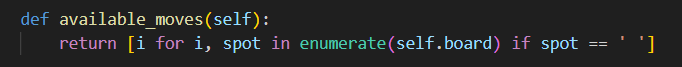
Descripción generada automáticamente

El método `\_\_init\_\_()` de la clase `TicTacToe` se ejecuta cuando se crea una nueva instancia de la clase. Aquí está el propósito de cada línea dentro de este método:

1. `self.board = [' '] \* 9`: Esto inicializa el tablero del juego como una lista de 9 elementos, donde cada elemento representa una celda del tablero. Inicialmente, todas las celdas están vacías, representadas por espacios en blanco `' '`.

2. `self.current\_winner = None`: Esto inicializa la variable `current\_winner` como `None`. Esta variable se utiliza para almacenar el ganador del juego. Si no hay un ganador, su valor permanece como `None`.

3. `self.states = {}`: Esto inicializa el diccionario `states`, que se utiliza para almacenar los estados del tablero del juego y las ponderaciones de los movimientos de la máquina. Cada estado del tablero se representa como una cadena de longitud 9 (resultado de concatenar los elementos del tablero). Los valores asociados con cada estado son otros diccionarios que contienen las ponderaciones de los movimientos posibles en ese estado.



Este método `available\_moves()` de la clase `TicTacToe` devuelve una lista de índices de las celdas del tablero que están vacías, es decir, donde aún no se ha realizado ningún movimiento. Aquí está la explicación de cómo funciona:

- `enumerate(self.board)`: Esto itera sobre los elementos de la lista `self.board`, devolviendo tanto el índice como el valor de cada elemento.

- `for i, spot in enumerate(self.board)`: Esto desempaqueta los valores devueltos por `enumerate` en las variables `i` (índice) y `spot` (valor de la celda).

- `if spot == ' '`: Esto comprueba si el valor de la celda es un espacio en blanco, lo que indica que la celda está vacía.

- `[i for i, spot in enumerate(self.board) if spot == ' ']`: Esto crea una lista de índices `i` para los cuales la celda correspondiente en el tablero está vacía. En otras palabras, crea una lista de movimientos disponibles.

Texto

Descripción generada automáticamente

El método `make\_move(self, square, letter)` de la clase `TicTacToe` permite a un jugador realizar un movimiento en el tablero. Aquí está la explicación de cómo funciona:

- `square`: Este parámetro representa la celda del tablero en la que el jugador desea realizar su movimiento. Es un número entero entre 0 y 8, que corresponde a los índices de las celdas del tablero.

- `letter`: Este parámetro representa la letra que el jugador está usando para marcar su movimiento en el tablero, que puede ser 'X' o 'O'.

- `if self.board[square] == ' ':`: Esta línea verifica si la celda en la posición indicada por `square` está vacía. Si es así, significa que el movimiento es válido y puede continuar.

- `self.board[square] = letter`: Aquí se marca la celda del tablero con la letra del jugador que realizó el movimiento.

- `if self.winner(square, letter):`: Después de realizar el movimiento, se verifica si el jugador que realizó el movimiento es el ganador. Esto se hace llamando al método `winner()` con los parámetros `square` y `letter`.

- `self.current\_winner = letter`: Si el jugador que realizó el movimiento gana, se actualiza la variable `current\_winner` con la letra del jugador que ganó.

- `return True`: Se devuelve `True` para indicar que el movimiento se realizó con éxito.

- `return False`: Si la celda en la posición indicada por `square` no está vacía, se devuelve `False` para indicar que el movimiento no se pudo realizar.

Texto

Descripción generada automáticamente

El método `winner(self, square, letter)` de la clase `TicTacToe` determina si un jugador ha ganado el juego después de realizar un movimiento en una celda específica. Aquí tienes una explicación de cómo funciona:

- `square`: Este parámetro representa la celda del tablero donde se realizó el último movimiento.

- `letter`: Este parámetro representa la letra ('X' o 'O') del jugador cuyo movimiento se está verificando.

- `row\_ind = square // 3`: Calcula el índice de fila de la celda `square`.

- `row = self.board[row\_ind\*3:(row\_ind+1)\*3]`: Obtiene los elementos de la fila correspondiente del tablero.

- `if all([spot == letter for spot in row]):`: Verifica si todos los elementos en la fila son iguales a la letra del jugador (`letter`). Si es así, significa que el jugador ha ganado en esa fila.

- `col\_ind = square % 3`: Calcula el índice de columna de la celda `square`.

- `column = [self.board[col\_ind+i\*3] for i in range(3)]`: Obtiene los elementos de la columna correspondiente del tablero.

- `if all([spot == letter for spot in column]):`: Verifica si todos los elementos en la columna son iguales a la letra del jugador (`letter`). Si es así, significa que el jugador ha ganado en esa columna.

- `if square % 2 == 0:`: Comprueba si el movimiento fue realizado en una celda que está en una de las diagonales.

- `diagonal1 = [self.board[i] for i in [0, 4, 8]]`: Obtiene los elementos de la diagonal principal del tablero.

- `if all([spot == letter for spot in diagonal1]):`: Verifica si todos los elementos en la diagonal principal son iguales a la letra del jugador (`letter`). Si es así, significa que el jugador ha ganado en esa diagonal.

- `diagonal2 = [self.board[i] for i in [2, 4, 6]]`: Obtiene los elementos de la otra diagonal del tablero.

- `if all([spot == letter for spot in diagonal2]):`: Verifica si todos los elementos en la otra diagonal son iguales a la letra del jugador (`letter`). Si es así, significa que el jugador ha ganado en esa diagonal.

- `return False`: Si ninguna de las condiciones anteriores se cumple, significa que el jugador no ha ganado en ninguna fila, columna o diagonal, por lo que devuelve `False`.

Texto

Descripción generada automáticamente

El método `get\_board\_state(self)` de la clase `TicTacToe` devuelve una representación del estado actual del tablero como una cadena. Aquí está cómo funciona:

- `return ''.join(self.board)`: Este método une todos los elementos de la lista `self.board` en una sola cadena. Cada elemento de la lista representa el contenido de una celda del tablero ('X', 'O' o espacio en blanco). Al unirlos todos en una cadena, obtenemos una representación del estado actual del tablero.

Por ejemplo, si `self.board` es `['X', ' ', 'O', ' ', 'X', 'O', ' ', ' ', 'X']`, entonces la cadena resultante será `'X O XOX X'`, que representa el estado del tablero donde 'X' y 'O' son los marcadores de los jugadores, y los espacios en blanco representan las celdas vacías del tablero.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

El método `add\_state(self, state)` de la clase `TicTacToe` agrega un nuevo estado del tablero al diccionario `self.states`, junto con un diccionario que inicializa las ponderaciones de los movimientos posibles en ese estado. Aquí está cómo funciona:

- `state`: Este parámetro representa el estado actual del tablero, que es una cadena obtenida mediante el método `get\_board\_state()`.

- `if state not in self.states:`: Esta línea verifica si el estado del tablero ya está presente en el diccionario `self.states`. Si el estado no está presente, se procede a agregarlo.

- `self.states[state] = {move: 0 for move in range(9)}`: Si el estado del tablero no está presente en el diccionario `self.states`, se agrega al diccionario como una nueva clave. El valor asociado a esta clave es otro diccionario que inicializa las ponderaciones de los movimientos posibles en ese estado. El diccionario interior se crea utilizando un diccionario de comprensión, donde las claves son los movimientos posibles (números del 0 al 8, que representan las celdas del tablero) y los valores iniciales de las ponderaciones se establecen en 0.

En resumen, este método asegura que cada estado del tablero se agregue al diccionario de estados (`self.states`) si aún no está presente, junto con un diccionario que inicializa las ponderaciones de los movimientos posibles en ese estado.

Texto

Descripción generada automáticamente

El método `update\_weights(self, move)` de la clase `TicTacToe` actualiza las ponderaciones de los movimientos en el diccionario `self.states` después de que se realiza un movimiento por parte de la máquina. Aquí tienes cómo funciona:

- `move`: Este parámetro representa el índice de la celda del tablero donde la máquina realizó su movimiento.

- `for state in self.states:`: Esto itera sobre cada estado del tablero presente en el diccionario `self.states`.

- `if move in self.states[state]:`: Esto verifica si el movimiento realizado (`move`) está presente como una clave en el diccionario de ponderaciones asociado al estado actual del tablero (`state`). Esto indica que el movimiento es uno de los posibles movimientos válidos en el estado actual.

- `self.states[state][move] += 10`: Si el movimiento es válido en el estado actual, se aumenta su ponderación en 10 unidades. Esto se hace accediendo al diccionario de ponderaciones asociado al estado actual y aumentando el valor correspondiente a la clave `move` en 10.

En resumen, este método aumenta la ponderación del movimiento realizado por la máquina en el estado actual del tablero para indicar que fue una elección beneficiosa. Esto ayuda a la máquina a aprender y tomar decisiones más efectivas en futuras jugadas.

Pantalla de computadora

Descripción generada automáticamente

Esta es la clase `TicTacToeGUI`, que maneja la interfaz gráfica de usuario del juego de Tres en Raya. Aquí hay una explicación de cómo funciona su método `\_\_init\_\_`:

- `self.game = TicTacToe()`: Esto crea una instancia de la clase `TicTacToe`, que representa la lógica del juego.

- `self.root = root`: Esta línea guarda una referencia a la ventana principal de la interfaz gráfica.

- `self.root.title("Proyecto Programación 3")`: Establece el título de la ventana principal como "Proyecto Programación 3".

- `self.buttons = [tk.Button(root, text=' ', font='normal 20 bold', height=3, width=6, command=lambda i=i: self.player\_move(i)) for i in range(9)]`: Esta línea crea una lista de 9 botones, cada uno representando una celda del tablero de Tres en Raya. Cada botón se configura con un texto en blanco, un tamaño de fuente grande, y un comando asociado (`command`) que llama a la función `self.player\_move(i)` cuando se hace clic. El parámetro `i` se captura en una función lambda para asegurarse de que cada botón se asocie con el índice correcto.

- Dentro del bucle `for`, se colocan los botones en la cuadrícula de la ventana principal utilizando el método `grid()`, asegurándose de que estén distribuidos en una cuadrícula de 3x3.

- `self.reset\_button = tk.Button(root, text='Reiniciar', command=self.reset\_game, font='normal 15 bold', bg='yellow')`: Este botón permite reiniciar el juego. Cuando se hace clic, llama al método `self.reset\_game()`. Se le da un fondo amarillo para destacarlo visualmente.

- `self.reset\_button.grid(row=3, column=0, columnspan=3, sticky='nsew')`: Coloca el botón de reinicio en la cuadrícula de la ventana principal, abarcando las tres columnas en la fila 3.

- Similarmente, se crea y se coloca un botón para mostrar las ponderaciones de la máquina y una etiqueta para mostrar el estado actual del juego (`self.status\_label`). La etiqueta inicialmente muestra "Tu turno".

- `self.game\_history = []`: Esta lista se utiliza para mantener un registro del historial de movimientos del juego.

Texto

Descripción generada automáticamente

Este método, `player\_move(self, index)`, maneja el movimiento realizado por el jugador. Aquí está cómo funciona:

- `index`: Este parámetro representa el índice de la celda del tablero en la que el jugador ha hecho clic.

- `if self.game.board[index] == ' ' and not self.game.current\_winner:`: Esta condición verifica si la celda seleccionada está vacía y si el juego todavía no ha terminado (es decir, no hay un ganador). Si ambas condiciones son verdaderas, el movimiento del jugador es válido y el juego continúa.

- `self.game.make\_move(index, 'X')`: Llama al método `make\_move()` de la instancia del juego para realizar el movimiento del jugador en la celda seleccionada, marcándola con una 'X'.

- `self.buttons[index].config(text='X', state='disabled')`: Actualiza el texto del botón correspondiente en la interfaz gráfica para mostrar la 'X' del jugador, y deshabilita el botón para evitar que se hagan más clics en él.

- `self.game\_history.append((self.game.get\_board\_state(), index))`: Agrega una tupla al historial de juego que contiene el estado actual del tablero y el índice de la celda donde el jugador realizó su movimiento.

- `self.game.update\_weights(index)`: Actualiza las ponderaciones de los movimientos de la máquina basándose en el movimiento realizado por el jugador.

- Si el jugador ha ganado después de su movimiento, se actualiza la etiqueta de estado (`self.status\_label`) para indicar que el jugador ha ganado y se deshabilitan todos los botones. Si el juego termina en empate, se actualiza la etiqueta de estado para indicar que es un empate. Si ninguno de estos casos ocurre, se llama al método `ai\_move()` para que la máquina realice su siguiente movimiento.

Texto

Descripción generada automáticamente

Este método `ai\_move(self)` maneja el movimiento de la máquina en el juego. Aquí tienes una explicación de cómo funciona:

- `state = self.game.get\_board\_state()`: Obtiene el estado actual del tablero.

- `self.game.add\_state(state)`: Agrega el estado actual del tablero al diccionario de estados (`self.states`) en la instancia del juego.

- `available\_moves = self.game.available\_moves()`: Obtiene una lista de movimientos disponibles para la máquina en el estado actual del tablero.

- `if available\_moves:`: Verifica si hay movimientos disponibles para la máquina. Si hay al menos uno disponible, la máquina seleccionará uno de forma aleatoria para realizar su movimiento.

- `move = random.choice(available\_moves)`: Elige aleatoriamente un movimiento disponible de la lista de movimientos disponibles.

- `self.game.make\_move(move, 'O')`: Llama al método `make\_move()` de la instancia del juego para realizar el movimiento de la máquina en la celda seleccionada, marcándola con una 'O'.

- `self.buttons[move].config(text='O', state='disabled')`: Actualiza el texto del botón correspondiente en la interfaz gráfica para mostrar la 'O' de la máquina, y deshabilita el botón para evitar que se hagan más clics en él.

- `self.game\_history.append((self.game.get\_board\_state(), move))`: Agrega una tupla al historial de juego que contiene el estado actual del tablero y el índice de la celda donde la máquina realizó su movimiento.

- `self.game.update\_weights(move)`: Actualiza las ponderaciones de los movimientos de la máquina basándose en el movimiento realizado por la máquina.

- Si la máquina ha ganado después de su movimiento, se actualiza la etiqueta de estado (`self.status\_label`) para indicar que la máquina ha ganado y se deshabilitan todos los botones. Si el juego termina en empate, se actualiza la etiqueta de estado para indicar que es un empate.

Texto

Descripción generada automáticamente

Este método `disable\_all\_buttons(self)` deshabilita todos los botones del tablero una vez que el juego ha terminado. Aquí está cómo funciona:

- Itera sobre todos los botones del tablero, que están almacenados en la lista `self.buttons`.

- Para cada botón, configura su estado (`state`) como "disabled". Esto significa que los botones ya no pueden ser clicados, lo que evita que los jugadores realicen más movimientos una vez que el juego ha terminado.

Texto

Descripción generada automáticamente

Este método `reset\_game(self)` se encarga de reiniciar el juego a su estado inicial. Aquí tienes una explicación de cómo funciona:

- `self.game = TicTacToe()`: Crea una nueva instancia de la clase `TicTacToe`, lo que reinicia el estado del juego.

- `for button in self.buttons:

button.config(text=' ', state='normal')`: Itera sobre todos los botones del tablero y los restaura a su estado inicial. Esto significa que el texto de cada botón se establece en un espacio en blanco, indicando que la celda está vacía, y el estado del botón se restablece a "normal", lo que permite que los jugadores hagan clic en ellos.

- `self.status\_label.config(text="Tu turno")`: Restablece el texto de la etiqueta de estado (`self.status\_label`) para indicar que es el turno del jugador humano.

Texto

Descripción generada automáticamente

Este método `show\_weights(self)` abre una nueva ventana (`Toplevel`) que muestra las ponderaciones de los movimientos de la máquina para cada estado del tablero. Aquí está cómo funciona:

- `weights\_window = tk.Toplevel(self.root)`: Crea una nueva ventana emergente (toplevel) que se muestra encima de la ventana principal (`self.root`). Esta ventana se utilizará para mostrar las ponderaciones de los movimientos.

- `weights\_window.title("Ponderaciones de Movimientos")`: Establece el título de la ventana emergente como "Ponderaciones de Movimientos".

- `row = 0`: Inicializa la variable `row` que se utilizará para posicionar los elementos en la ventana emergente.

- Se itera sobre el diccionario `self.game.states.items()`, que contiene cada estado del tablero como clave y un diccionario de movimientos con sus ponderaciones como valor.

- Para cada estado del tablero, se crea una etiqueta (`Label`) en la ventana emergente (`weights\_window`) que muestra el estado del tablero.

- Luego, se itera sobre el diccionario de movimientos de ese estado, mostrando cada movimiento y su ponderación en una etiqueta adicional.

- Las etiquetas se colocan en la ventana emergente utilizando el método `grid()`, con la variable `row` que se actualiza después de agregar cada movimiento para posicionar correctamente las etiquetas en filas consecutivas.

Esto crea una ventana que muestra todas las ponderaciones de los movimientos de la máquina para cada estado del tablero, proporcionando una visión detallada de cómo la máquina toma decisiones en el juego.

Texto

Descripción generada automáticamente

Este bloque de código es típico en aplicaciones de GUI en Python con tkinter. Aquí está lo que hace:

- `if \_\_name\_\_ == '\_\_main\_\_':`: Esto asegura que el bloque de código dentro de este condicional solo se ejecute si el script se está ejecutando directamente, no si se está importando como un módulo en otro script.

- `root = tk.Tk()`: Crea la ventana principal de la aplicación utilizando `tk.Tk()`. Esta ventana servirá como el contenedor principal para todos los elementos de la interfaz gráfica.

- `gui = TicTacToeGUI(root)`: Crea una instancia de la clase `TicTacToeGUI`, pasando la ventana principal `root` como argumento. Esto inicializa y muestra la interfaz gráfica del juego de Tres en Raya.

- `root.mainloop()`: Inicia el bucle principal de eventos de tkinter, que espera y maneja eventos de entrada del usuario (como clics de ratón y pulsaciones de teclas) y actualiza la interfaz gráfica en consecuencia. Este bucle continuará ejecutándose hasta que se cierre la ventana principal.