03.05C - TicTacToe - Server UDP en Python

Lenguajes Estructurados

Detalles de implementación

Importación de módulos y configuración inicial

```
import socket
import pygame
import threading
```

Estas líneas importan los módulos necesarios para el programa. `socket` se usa para la comunicación de red, `pygame` para la interfaz gráfica y `threading` para ejecutar el servidor en un hilo separado.

```
# Configuración de Pygame
WINDOW_SIZE = (300, 300)
LINE_WIDTH = 5
FONT_SIZE = 80

# Configuración del servidor
SERVER_ADDRESS = ('localhost', 12345)

# Tablero del juego
BOARD_SIZE = 3
```

Aquí se definen algunas constantes que se usarán en el programa. `WINDOW_SIZE` es el tamaño de la ventana de Pygame, `LINE_WIDTH` es el ancho de las líneas del tablero, `FONT_SIZE` es el tamaño de la fuente usada para dibujar las fichas, `SERVER_ADDRESS` es la dirección del servidor y `BOARD_SIZE` es el tamaño del tablero de juego.

Clase GameBoard

```
class GameBoard:
    def __init__(self):
        self.board = [[' ']*BOARD_SIZE for _ in range(BOARD_SIZE)]
        self.window = pygame.display.set_mode(WINDOW_SIZE)
        pygame.display.set_caption("Tateti")
        self.font = pygame.font.Font(None, FONT_SIZE)
```

La clase `GameBoard` representa el tablero de juego. En el método `__init__`, se inicializa el tablero como una matriz de 3x3, se crea la ventana de Pygame y se configura el título y la fuente.

```
    def draw(self):
        self.window.fill((255, 255, 255))
        self._draw_lines()
        self._draw_pieces()
        pygame.display.flip()
```

El método `draw` se encarga de dibujar el tablero en la ventana. Primero, llena la ventana con blanco. Luego, dibuja las líneas y las fichas del tablero. Finalmente, actualiza la ventana con `pygame.display.flip()`.

```
    def _draw_lines(self):
        for i in range(1, BOARD_SIZE):
            pygame.draw.line(self.window, (0, 0, 0), (i*100, 0), (i*100, 300), LINE_WIDTH)
            pygame.draw.line(self.window, (0, 0, 0), (0, i*100), (300, i*100), LINE_WIDTH)
```

El método `_draw_lines` es un método privado que dibuja las líneas del tablero. Recorre cada fila y columna del tablero y dibuja una línea en la posición correspondiente.

```
    def _draw_pieces(self):

        for i in range(BOARD_SIZE):
            for j in range(BOARD_SIZE):

                if self.board[i][j] == '2':
                    text = self.font.render('X', True, (0, 0, 0))
                    self.window.blit(text, (j*100+30, i*100))

                elif self.board[i][j] == '1':
                    text = self.font.render('O', True, (0, 0, 0))
                    self.window.blit(text, (j*100+30, i*100))
```

El método `_draw_pieces` es otro método privado que dibuja las fichas del tablero. Recorre cada celda del tablero y, si la celda contiene una ficha, dibuja la ficha correspondiente en la posición correcta.

```
    def update(self, message):
        self.board = [list(message[i:i+BOARD_SIZE]) for i in range(0, len(message), BOARD_SIZE)]
```

El método `update` actualiza el estado del tablero con un nuevo mensaje recibido del cliente.
El mensaje es una cadena de texto que representa el estado actual del tablero, por lo que este método convierte el mensaje en una matriz 2D que se puede usar para dibujar el tablero.

Clase GameServer

```
class GameServer:
    def __init__(self, game_board):
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.bind(SERVER_ADDRESS)
        self.game_board = game_board
```

La clase `GameServer` representa el servidor del juego. En el método `__init__`, se crea un nuevo socket de servidor, se vincula a la dirección del servidor y se guarda una referencia al tablero de juego.

```
    def start(self):
        print("Servidor iniciado. Esperando mensajes...")

        while True:
            message, client_address = self.server_socket.recvfrom(1024)
            print(f"Mensaje recibido desde {client_address}: {message.decode('utf-8')}")
            self.game_board.update(message.decode('utf-8'))
```

El método `start` inicia el servidor. Entra en un bucle infinito en el que espera mensajes de los clientes. Cuando recibe un mensaje, lo imprime y luego actualiza el tablero de juego con el mensaje.

Código principal

```
if __name__ == "__main__":
    pygame.init()

    game_board = GameBoard()
    game_server = GameServer(game_board)

    server_thread = threading.Thread(target=game_server.start)
    server_thread.start()

    while True:
        game_board.draw()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                game_server.server_socket.close()
                exit(0)
```

Este es el código principal que se ejecuta cuando se inicia el programa.

- Primero, inicializa Pygame. Luego, crea una nueva instancia de `GameBoard` y `GameServer`.
- Después, inicia el servidor en un nuevo hilo.
- Finalmente, entra en un bucle infinito en el que dibuja el tablero y maneja los eventos de Pygame.
- Si se recibe el evento `QUIT`, cierra Pygame y el socket del servidor y termina el programa.