Documentação do Protocolo de Jogo Online de Tiro

Este documento descreve o protocolo de comunicação utilizado entre o cliente e o servidor nos arquivos client.py e server.py. O protocolo é baseado em sockets TCP/IP e utiliza a biblioteca pickle para serialização e desserialização dos dados trocados entre as partes.

Configurações

Arquivo server.py

No arquivo server.py, as configurações do servidor são definidas nas seguintes variáveis:

- HOST: Define o endereço IP ou nome do host do servidor. No exemplo fornecido, está configurado como 'localhost', indicando que o servidor está sendo executado localmente.
- PORT: Define o número da porta em que o servidor está escutando por conexões.

Além disso, são definidas as seguintes configurações relacionadas ao jogo:

- SCREEN_WIDTH: Define a largura da tela do jogo.
- SCREEN_HEIGHT: Define a altura da tela do jogo.
- PLAYER_RADIUS: Define o raio do jogador.
- PLAYER_SPEED: Define a velocidade de movimentação do jogador.
- SHOOT_COOLDOWN: Define o tempo de recarga da arma, em frames.
- SHOT_SPEED: Define a velocidade dos tiros disparados.
- SHOT_RADIUS: Define o raio dos tiros.

Arquivo client.py

No arquivo client.py, as configurações do cliente são definidas nas seguintes variáveis:

- HOST: Define o endereço IP ou nome do host do servidor. No exemplo fornecido, está configurado como 'localhost', indicando que o cliente está se conectando a um servidor em execução localmente.
- PORT: Define o número da porta em que o servidor está escutando por conexões.

Além disso, são definidas as seguintes configurações relacionadas ao jogo:

- SCREEN_WIDTH: Define a largura da tela do jogo.
- SCREEN_HEIGHT: Define a altura da tela do jogo.
- PLAYER_RADIUS: Define o raio do jogador.
- SHOT_RADIUS: Define o raio dos tiros.
- SHOT_SPEED: Define a velocidade dos tiros disparados.

Funcionamento

Arquivo server.py

O arquivo server.py implementa o servidor do jogo. O servidor aceita conexões de até 2 clientes e gerencia as informações dos jogadores e tiros disparados.

Função handle_client_connection(client_socket, player_id)

Esta função é responsável por lidar com a conexão de um cliente específico. Ela recebe o socket de comunicação com o cliente e o ID do jogador associado a essa conexão.
Dentro da função, ocorre um loop principal que recebe os dados do jogador, atualiza as informações do jogo e envia os dados atualizados de volta para o cliente.
Os dados recebidos do cliente são desserializados usando o módulo pickle e contêm as seguintes informações:

- movement: Uma string indicando o movimento do jogador ('UP', 'DOWN', 'LEFT', 'RIGHT').
- shooting: Um booleano indicando se o jogador está disparando.
- mouse_x: A posição x do cursor do mouse.
- mouse_y: A posição y do cursor do mouse.

Com base nas informações recebidas, as seguintes ações são executadas:

- Atualização da posição do jogador com base no movimento.
- Verificação dos limites da tela para o jogador.
- Verificação de colisões entre o jogador e os tiros disparados.
- Atualização do tempo de recarga da arma.
- Disparo de um novo tiro, se o jogador estiver atirando e o tempo de recarga permitir.

Após as atualizações, os dados são serializados novamente usando o pickle e enviados de volta para o cliente.

Função update_shots()

Esta função é executada em uma thread separada e é responsável por atualizar a posição dos tiros disparados.
Dentro da função, ocorre um loop principal que itera sobre a lista de tiros e atualiza suas posições com base na velocidade configurada. Os tiros que ainda estão dentro dos limites da tela são mantidos, e os demais são descartados.
A função utiliza um pequeno delay (threading.Event().wait(0.01)) para controlar a velocidade de atualização dos tiros.

Loop principal

O loop principal do servidor aguarda a conexão de clientes em um socket TCP/IP. Quando um cliente se conecta, é criada uma nova thread para lidar com esse cliente usando a função handle_client_connection.
O servidor mantém uma lista de informações dos jogadores conectados e uma lista de tiros disparados. Essas listas são atualizadas durante o jogo.

Arquivo client.py

O arquivo client.py implementa o cliente do jogo. O cliente se conecta ao servidor e envia comandos de movimentação e disparo para o servidor, além de receber dados atualizados do jogo.

Loop principal

O loop principal do cliente executa continuamente e lida com os eventos do Pygame, capturando a entrada do teclado e do mouse do jogador.
Com base nos eventos capturados, o cliente envia os dados para o servidor usando o socket de comunicação. Os dados são serializados com o pickle e contêm as seguintes informações:

- movement: Uma string indicando o movimento do jogador ('UP', 'DOWN', 'LEFT', 'RIGHT').
- shooting: Um booleano indicando se o jogador está disparando.
- mouse_x: A posição x do cursor do mouse.
- mouse_y: A posição y do cursor do mouse.

Após o envio dos dados, o cliente recebe os dados atualizados do servidor, também serializados com o pickle. Os dados recebidos incluem informações sobre os jogadores, tiros disparados, estado do jogo (se terminou) e o vencedor do jogo.
O cliente desserializa os dados recebidos e verifica se o jogador foi eliminado. Em caso de eliminação, uma mensagem de "Game Over" é exibida na tela e o cliente espera alguns segundos antes de fechar a janela.
Após o processamento dos dados recebidos, o cliente renderiza a tela do jogo utilizando a biblioteca Pygame. Ele desenha os jogadores e os tiros com base nas informações recebidas e atualizadas.

Encerramento

O servidor e o cliente possuem trechos de código que encerram as conexões e encerram a execução dos programas.
No arquivo server.py, o servidor é encerrado chamando o método close() do socket do servidor. Além disso, a mensagem "Servidor encerrado" é exibida na saída padrão.
No arquivo client.py, o cliente encerra a conexão com o servidor chamando o método close() do socket do cliente. Além disso, a mensagem "Conexão encerrada" é exibida na saída padrão do cliente.
Ambos os programas incluem blocos de código para capturar exceções e lidar com erros de conexão, como a perda de conexão com o servidor. Esses blocos de código geralmente exibem mensagens de erro apropriadas e encerram a execução do programa de forma adequada.
É importante mencionar que o código apresentado é apenas um exemplo básico de implementação de um jogo multiplayer online de tiro em Python usando sockets TCP/IP. Existem muitas outras considerações a serem feitas para tornar o jogo mais robusto e seguro, como validação dos dados recebidos, implementação de sistemas de autenticação, entre outros.