Aplicação Bomberman Online e seus Protocolos

Igor B. Valerão[1], **Lucas B. Araujo**[1], **Pablo A. Ramalho**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPel)
Caixa Postal 15.064 – 91.501-970 – Pelotas – RS – Brazil

{ibvalerao, lbdaraujo, pablo.aramalho}@inf.ufpel.edu.br

1. Objetivo

Desenvolver um protocolo de camada de aplicação para um jogo Bomberman multiplayer, utilizando a arquitetura cliente-servidor. O sistema permite que até 4 clientes se conectem a um servidor central, que é responsável por gerenciar a criação de partidas, sincronizar o estado do jogo e coordenar as ações dos jogadores. O protocolo define as interações entre clientes e servidor, incluindo a entrada em partidas, movimentação dos personagens no mapa, posicionamento e explosão de bombas, detecção de colisões e definição das condições de vitória.

2. Características

- **Arquitetura:** Cliente-Servidor, com o servidor hospedando a partida e os clientes fazendo o acesso;
- **Modelo de Conexão:** sem estado, em que as informações, como posição do personagem, colisão e disposição das caixas de madeira, estão no cliente;
- **Persistência:** conexão persistente, que é mantida durante toda a partida;
- **Modo de Comunicação:** pull, em que clientes enviam ações, como posicionar e explodir bombas, e servidor responde com atualizações;
- **Controle:** na banda, em que mensagens de controle e dados trafegam no mesmo canal;
- **Segurança:** Não foram implementadas medidas de segurança.

3. Tipos de Mensagens

Todas as mensagems utilizadas são precedidas pelo seu tamanho em bytes, no seguinte formato:

<TAMANHO DA MENSAGEM> < ... >

Isso permite que o receptor saiba exatamente quantos bytes precisa ler para montar a mensagem inteira. Por sua vez, esse tamanho da mensagem é um short de 2 bytes sem sinal. A codificação dos dados utiliza o formato big endian, em que o byte mais significativo é guardado no endereço menor. é um short de 2 bytes sem sinal.
É importante notar que os campos das mensagens são passados através de bytes, ao invés de json.

3.1. Cliente para o Servidor

Listagem dos tipos de mensagens possíveis de um cliente para o servidor e seus campos de cabeçalho.

Bomb Placed: posicionamento de bomba por parte de um jogador.

<BOMB PLACED><POSIÇÂOX><POSIÇÂOY><TEMPO>
Inteiro sem sinal 4 bytes, 11 Bytes, 2 x Inteiro 4 bytes, Inteiro sem sinal 4 bytes
Total = 27 Bytes

Player Joined: acesso à partida por um jogador.

<PLAYER JOINED><R><G><B><HEIGHT><WIDTH><POSIÇÂOX><POSIÇÂOY><ID>
13 bytes, 3 bytes, 2 x Inteiro sem sinal 4 bytes, 3 x Int 4 Bytes
Total = 36 Bytes

Lobby Data: envio de mensagem requisitando os dados do lobby.

<LOBBY DATA>
10 Bytes

Change Pos: movimentação dos jogadores no cenário do jogo.

<CHANGEPOS><ID><POSIÇÂOX><POSIÇÂOY>
9 bytes, Short sem sinal 2 bytes, inteiro 4 bytes x 2
Total = 19 bytes

3.2. Servidor para o Cliente

Listagem dos tipos de mensagens possíveis do servidor para os clientes e seus campos de cabeçalho.

Bomb Placed:reconhecimento do posicionamento de uma bomba por parte de algum jogador e repasse dessa informação para os demais clientes. Ela usa o parâmetros de coordedas nos eixos x e y e tempo. Também cria um objeto bomba.

<BOMB PLACED>< ... >

Dados do jogador: mensagem enviada quando um jogador entra no jogo. O servidor envia a lista de todos os jogadores já conectados, bem como sua posição inicial no cenário, para os clientes existentes e também informa o novo jogador que acabou de se conectar.

<R><G><B><HEIGHT><WIDTH><POSIÇÂOX><POSIÇÂOY><ID>, 23 bytes

Player Joined: mensagem enviada quando um novo jogador entra na partida. Ela contém o ID do jogador que entrou e o número de jogadores conectados. Também contém os dados de cada jogador.

<PLAYER JOINED><ID DO JOGADOR QUE ENTROU><NÚMERO DE JOGADORES>< DADOS DO JOGADOR 1 > < DADOS DO JOGADOR 2 > < ... > <

DADOS DO JOGADOR n >
13 bytes, Inteiro sem sinal 4 bytes, Short 2 bytes, 23 bytes para cada jogador, ou seja N * 23 bytes.

$$Total = 19 + 23N$$

PDATA = <SE O JOGADOR ENTROU NO LOBBY><ID DO JOGADOR><POSIÇÂO NO LOBBY>

Lobby Data: informa a lista de jogadores no lobby.

<LOBBY DATA> < 4 X PDATA >
10 Bytes, 4x (Boolean 1 byte, Inteiro 4 bytes, Short 2 bytes)

Total = 38 Bytes

Player Updated: atualiza o estado dos demais jogadores.

<PLAYER UPDATED> < DADOS DO JOGADOR >
14 Bytes, 23 bytes

Total = 37 Bytes

Player Disconnected: informa se um dos jogadores deixar o jogo, através do campo de ID do mesmo.

<PLAYER DISC><ID>
11 Bytes, Short 2 bytes

Total = 13 bytes

4. Outras Informações

4.1. Funcionamento Geral:

- Inicialmente o servidor é inicializado. Isso é feito através do arquivo *Server.py.*
- Jogador inicia a aplicação, rodando *Menu.py.*.
- O jogador solicita uma partida e conecta-se ao servidor, por meio dos IPs definidos em *Global.py.*, referente ao Cliente, e *Server.py.*, referente ao Servidor.
- As informações do jogo, como disposição inicial das caixas de madeira e condições relativas as bombas são guardadas no Cliente.
- Jogador solicita o começo da partida e ela começa para todos os jogadores.
- Durante a partida, jogadores enviam ações para o servidor.
- O servidor faz a comunicação entre os jogadores através das mensagens.
- O servidor processa cada ação, atualiza os Clientes.

4.2. Bibliotecas Utilizadas

No desenvolvimento do jogo Bomberman multiplayer foram utilizadas diversas bibliotecas Python, cada uma com uma função específica:

- **socket:** utilizada para comunicação em rede, permitindo que clientes e servidor troquem mensagens TCP.
- **pygame** usada para criação da interface gráfica, gerenciamento de sprites, detecção de eventos de teclado e mouse, e renderização do jogo.
- **math:** fornece funções matemáticas básicas, como cálculos de distância ou ângulos, úteis para movimentação e colisões.
- **sys:** utilizada para acessar funções do sistema, como encerrar o programa de forma controlada.
- **time:** usada para controlar temporizações, como o tempo de explosão das bombas e ticks do jogo.
- **struct:** permite empacotar e desempacotar dados binários em formatos específicos, essencial para enviar mensagens padronizadas pelo protocolo de rede.
- **threading** utilizada para criar threads paralelas, garantindo que o jogo e a comunicação em rede rodem simultaneamente. Além disso, Locks e Events ajudam a controlar o acesso a recursos compartilhados entre threads.
- **numpy:** utilizada para cálculos numéricos e manipulação de arrays, facilitando operações como cálculos de colisão ou transformação de coordenadas.

4.3. Dificuldades Encontradas

- Não foi possível aplicar medidas de segurança sobre as conexões.
- Não foi possível fazer com que as bombas causassem dano aos personagens do jogadores.
- Não foi possível estabelecer as condições para o fim da partida.