TP2 - Batalha Pokémon

Jackson Nunes Silva - 2018054532

Janeiro 2022

1 Introdução

O objetivo deste trabalho prático foi assimilar e exercitar os conceitos sobre programação com sockets e protocolo **UDP** de comunicação entre cliente e servidor vistos na disciplina Redes de Computadores. Para tal, foi proposta a implementação de um jogo baseado em turnos em que existem 4 arenas de batalha e N linhas de defesa, onde os pokémons defensores são posicionados. A cada turno, os pokémons atacantes andam uma posição em linha reta e o objetivo é derrotá-los antes que cheguem até o final da arena.

2 Implementação

O programa foi implementado em linguagem C e divido em 6 arquivos:

- *server.c*, que possui toda a implementação do servidor.
- *client.c*, que possui a implementação do cliente.
- *common.h* e *common.c*, que implementam duas funções parecidas para inicializar os endereços IPv4 e IPv6, tanto para o cliente quanto para o servidor.
- *game.h* e *game.c*, que implementam as estruturas e funções necessárias para o funcionamento do jogo.

2.1 Estrutura de Dados

Para simplificar a implementação, foram utilizados os dois structs descritos a seguir, que estão declarados no arquivo *game.h*:

> **Pokemon**, contendo dados sobre um Pokémon atacante (nome, identificador, vida, hits tomados e posição).
>
> **Game**, contendo todos os dados sobre o jogo (número de pokémons invocados, número de linhas de defesa, número de pokémons defensores, número de pokémons derrotados, número de pokémons vitoriosos, identificador do turno, um array com 4 elementos do tipo **Pokemon** e uma matriz que corresponde às linhas de defesa).

2.2 Funções

A primeira função a ser chamada é a de inicialização do jogo ***init_game()*** que inicia todas as variáveis do struct "Game" e conta com funções auxiliares que geram aleatoriamente o **número de linhas de defesa**, o **número de pokémons defensores** e os 4 **pokémons atacantes**, que podem ser do tipo "zubat", "lugia" ou "mewtwo". Válido ressaltar que o número de linhas de defesa está definido no intervalo $[4, 6]$ para facilitar os testes, porém o limite superior pode ser facilmente alterado, como descrito no **item 5** da **Seção 3**.

Para o controle das ações requisitadas pelo cliente, são utilizadas 2 funções: ***get_action_type()***, que retorna um número identificando o tipo de resposta que aquela ação gera (se apenas um servidor responde ou se são 4 respostas); e ***run_action()*** que processa a ação, que são basicamente **start**, **getdefenders**, **getturn** ou **shot**, e chama a função correspondente para executá-la. Essas funções são descritas a seguir:

Após iniciar o struct **Game**, a função ***set_defenders()*** é chamada, cuja tarefa é inicializar a matriz de linha de defesa com os pokémons defensores em posições aleatórias. Essa matriz possui dimensões $(5, N)$, onde N é o número de linhas de defesa fixas. As posições onde os defensores são gerados recebem o valor 1 e o número máximo de defensores é igual a $N + N/2$.

A função ***get_defenders()*** é responsável por retornar uma string contendo os pares de coordenadas (X,Y) dos defensores, onde X é o número da linha de defesa e Y a orientação horizontal. Exemplificando, um defensor localizado na posição $(2, 0)$ da matriz é representado no jogo pela coordenada $(1, 2)$ e não $(0, 2)$, ou seja, a coordenada (i, j) na matriz é a coordenada $(j + 1, i)$ no jogo, pois a identificação da linha de defesa começa pelo número 1.

Ao receber um comando *getturn*, a primeira função chamada é ***set_turn()***, que basicamente atualiza o turno, alterando todas as variáveis necessárias. A posição de todos os pokémons atacantes é incrementada em 1. Aqueles pokémons cuja vida se igualou à quantidade de hits tomados incrementam o número de mortos e são substituídos por um novo pokémon aleatório na posição 0. Já os pokémons cuja posição ultrapassou o número de linhas de defesa são os que sobreviveram, portanto, incrementam o número de pokémons vivos. Logo em seguida, a função ***get_turn()*** é chamada, que retorna uma string com os dados do turno requisitado.

Por fim, tem a função ***make_shot()***, que é chamada após um comando *shot* e é responsável por verificar se existe um pokémon de defesa na posição indicada e se ele pode defender o pokémon de ataque indicado. Caso positivo, o número de hits do pokémon atacante é incrementado e o pokémon defensor é inativado até o próximo turno (cada um só pode defender uma vez por turno).

3 Observações Importantes

A seguir, são listadas algumas decisões de implementação que podem afetar a execução do programa:

1. A ideia inicial era criar quatro servidores agindo em 4 *threads* diferentes, no entanto, não foi possível fazer a sincronização entre eles funcionar a tempo. Por isso, o jogo ficou sendo controlado inteiramente por **um único servidor**. Assim, o cliente se conecta com apenas um servidor e as mensagens chegam através de 4 *recvfrom()*, simulando os 4 servidores.

2. A retransmissão é feita quando o cliente fica há 2 segundos sem receber uma resposta após ter enviado uma mensagem. Isso ocorre no máximo 5 vezes e encerra a execução do cliente caso não tenha sucesso. A implementação foi feita como descrito na seção 6.3 do livro *TCP IP Sockets in C - Second Edition*.

3. Não foram feitos tratamentos de erros em relação aos argumentos da linha de comando e para ações inválidas requisitada pelo cliente, principalmente pelo uso da função *atoi()*. Por exemplo, se o cliente requisita a ação "shot -1 123 b", gera um comportamento não definido, podendo causar falha de segmentação. Por isso, recomenda-se o uso adequado das ações.

4. Caso o cliente requisite uma ação desconhecida, a mensagem de *game over* é envida e o servidor inicia um novo jogo. Portanto, nesse caso, o cliente pode continuar jogando como se fosse o início (requisitando os defensores, turno 0, ...) ou enviar o comando *quit*, que encerra tanto a própria execução como a do servidor.

5. Como mencionado na Seção 2.2, o número máximo de linhas de defesa está fixado em 6, porém isso pode ser alterado pelo usuário, basta alterar o valor da constante **MAXDEFLINE**, na **linha 6** do arquivo ***game.h***. Assim, caso o usuário mude para um valor N, o número de linhas de defesa será gerado aleatoriamente no intervalo $[4, N]$.

6. Para auxiliar na visualização do jogo, a cada comando o servidor imprime a matriz de linhas de defesa (0 nas posições vazias, 1 nas posições onde tem um defensor) e os pokémons atacantes, no formato (id nome vida hits). Em cada turno, o valor dos defensores que executaram a ação *shot* é atualizado para 2 (para diferenciar daqueles que não executaram ainda) e é resetado para 1 assim que o próximo turno é chamado.

4 Desafios e Conclusão

Sem dúvidas, o maior desafio deste trabalho prático foi em relação à conexão com os quatro servidores. Infelizmente, não foi possível concluir essa etapa a tempo apesar dos esforços empenhados, então todo o jogo ficou entre um só cliente e um só servidor. Apesar dessas dificuldades, foi uma ótima oportunidade para exercitar os conhecimentos adquiridos. Além disso, tem o fator da satisfação e diversão de ver o jogo em funcionamento, mesmo sendo simples e tendo detalhes a serem melhorados e código a ser refatorado.