Nome: Fábio Silva Brum
Matrícula: 2018123674

Trabalho Prático 2 - Redes de Computadores

1. Estratégia de implementação

O projeto foi desenvolvido no sistema operacional Ubuntu 18.04.5 LTS, com um processador core i5 e 8GB de ram.

O sistema simula o jogo da seguinte forma: ao iniciar a conexão do servidor enviando o comando "start", o servidor inicia o campo de batalha, os pokemons de ataque e de defesa. Quando é feito o comando "getturn", o servidor avança os pokemons de ataque em uma posição (caso um deles atinja a pokedex, é eliminado do campo e entra para a estatística de vencedores) no campo de batalha, e com 25% de chance ele gera novos pokemons na primeira posição de cada "linha" do campo. O servidor então espera os comandos de ataque do cliente para alterar a vida dos pokemons de ataque e , quando é recebido outro comando de "getturn", o ciclo se repete, até que o turno 50 seja atingido ou uma mensagem inválida seja enviada. Nesses casos, é gerado um "gameover", e o usuário pode escolher entre iniciar outra partida enviando outro comando "start" ou então encerrar o programa, digitando "quit".

Foram criados 4 servidores, sendo um servidor responsável por sua respectiva linha de combate (o servidor 1 manuseia a linha 1, o servidor 2 manuseia a linha 2, e assim por diante). O cliente envia mensagens apenas para o servidor 1, que em seguida verifica qual servidor é responsável por executar a ação que ele recebeu a partir da regra dita anteriormente. A tarefa é então executada e, por fim, o servidor responsável envia uma mensagem para o cliente com o resultado da mesma.

2. Estruturas de dados

- **struct pokemon_attack:**

Este struct foi criado para representar os pokemons de ataque. Ele possui 3 atributos:

1. Int id: armazena o id do pokemon de ataque;
2. Char *nome: o nome do pokemon de ataque (Mewtwo, Zubat ou Lugia);
3. Int life: armazena a vida do pokemon (1, 2 ou 3, dependendo do pokemon);

- **struct pokemon_attack attack_pokemons[MAX_NUM_OF_ATTACK_POKEMONS]:**

Array que armazena todos os pokemons vivos no turno (tamanho máximo de 16, visto que o tabuleiro é 4x4).

- **struct pokemon_defense:**

Este struct foi criado para representar os pokemons de defesa. Ele possui 3 atributos:

1. Int x_pos: armazena a coordenada X do pokemon de defesa (qual "linha" ele está);
2. Int y_pos: armazena a coordenada Y do pokemon de defesa (qual "coluna" ele está);
3. Int has_attacked: armazena se o pokemon já atacou na rodada ou não.

- **struct pokemon_defense defense_pokemons[NUM_OF_DEFENSE_POKEMONS]:**

Array que armazena todos os pokemons de defesa da partida (tamanho do vetor: 6, definido por mim).

- **int battle_fields[NUM_OF_SERVERS][NUM_OF_STEPS]:**

Matriz que representa o estado da partida. Armazena o ID dos pokemons de ataque em sua respectiva posição. Por exemplo, caso o pokemon de ID 9 esteja na última posição da segunda "pista" na rodada, estará armazenado o valor 9 na posição [1][3] da matriz.

3. Desafios e Soluções

Inicialmente, o principal desafio que encontrei foi a interpretação da documentação do trabalho. Dúvidas como o funcionamento do jogo, os exemplos fornecidos, regras de envio e retorno, dentre outros aspectos, dificultaram para que eu conseguisse iniciar minhas tarefas. Com as lives de tira-dúvidas e as observações feitas na documentação eu consegui eliminar a maior parte das ambiguidades que tinha em minha interpretação (e outras não vi a resposta, como o número de pokemons de defesa que defini como 6 pokemons e a chance de aparecerem novos pokemons que defini como 25%), e comecei a desenvolver o sistema.

Comecei a construir os servidores pela parte das regras de jogo, ou seja, condições de ataque, movimentação, armazenamento dos dados de tempo de execução, pokemons mortos e vencedores, etc. Já que essa parte não exige conhecimento da matéria, os únicos desafios que encontrei foram relacionados à linguagem de programação C, entretanto, devido ao trabalho prático 1, eu estava mais preparado para enfrentá-los e tive pouca dificuldade para encontrar soluções para os erros de execução ocorridos.

Depois de construir o jogo, parti para o tratamento dos IPs (IPV4 e IPV6). Novamente, o último TP foi de grande ajuda no desenvolvimento dessa parte, então não tive maiores problemas.

Em seguida, iniciei o desenvolvimento da retransmissão de pacotes, parte da qual tive uma maior dificuldade. Como não tinha feito essa retransmissão antes, fui buscar auxílio no material oferecido pelo professor para ter ideia de como fazê-la. Encontrei uma possível solução no "Livro sobre programação com sockets" (página 112 do livro, seção 6.3.3) e tentei aplicá-la com algumas alterações.

Primeiramente alterei a forma de usar o timeout. Na minha versão, adicionei a opção *SO_RCVTIMEO* ao socket do cliente através da função *setsockopt*. Essa opção é configurada junto a uma variável do tipo *struct timeval* que também é enviada junto aos parâmetros da função, na qual é armazenada o tempo do timeout (no meu caso o timeout é de 0.01s, visto que a comunicação é feita no localhost).

Na questão de checagem do recebimento da mensagem, mantive a ideia do livro, na qual ele criou um loop *while*, cuja condição é o retorno da função *recvfrom*. Caso seja retornado um valor <0, significa que o tempo do timeout foi atingido e a mensagem não foi recebida. Nesse caso, é conferido se a string que receberia a mensagem realmente está vazia, e caso esteja, o código entra no tratamento de retransmissão de pacotes. Assim, é feita uma limpeza da variável do tipo *struct sockaddr_storage* que armazena as informações do servidor cuja mensagem está sendo esperada, e depois a preenchemos novamente com as informações do servidor. Em seguida, é feita um novo envio da mesma mensagem para o servidor, e voltamos para a condição inicial do loop, onde será esperado novamente pelo retorno desse servidor.

4. Exemplos do código

- **Comando "start":**

```
fabio@fabio:~/ufmg/redes_tp2$ ./client 127.0.0.1 9000 start
game started: path 1
game started: path 2
game started: path 3
game started: path 4
```

- **Comando "getdefenders":**

```
getdefenders
defender [[4, 4], [3, 4], [4, 2], [2, 4], [2, 1], [1, 2]]
```

- **Comando "shot X Y ID":**

```
shot 2 1 2
shotresp 2 1 2 0
```

```
shot 0 3 1
shotresp 0 3 1 0
shot 0 3 1
shotresp 0 3 1 1
```

- **Comando “quit”:**

```
getturn 50
gameover 0 4 4 245
quit
fabio@fabio:~/ufmg/redes_tp2$
```

```
some invalid message
gameover 1 0 0 397
quit
fabio@fabio:~/ufmg/redes_tp2$
```

```
getturn 50
gameover 0 0 0 13
start
game started: path 1
game started: path 2
game started: path 3
game started: path 4
```

- **Comando “getturn X”:**

```
getturn 0
Base 1
turn 0
fixedLocation 1
1 Mewtwo 3

turn 0
fixedLocation 2


turn 0
fixedLocation 3


turn 0
fixedLocation 4



Base 2
turn 0
fixedLocation 1
2 Mewtwo 3

turn 0
fixedLocation 2


turn 0
fixedLocation 3


turn 0
fixedLocation 4
```

```
Base 3
turn 0
fixedLocation 1
3 Lugia 2

turn 0
fixedLocation 2


turn 0
fixedLocation 3


turn 0
fixedLocation 4



Base 4
turn 0
fixedLocation 1
4 Lugia 2

turn 0
fixedLocation 2


turn 0
fixedLocation 3


turn 0
fixedLocation 4
```