Dificuldades e Desafios

O maior desafio desse trabalho foi a falta de materiais de consulta para guiar a implementação do jogo. Servidores utilizando o protocolo UDP na linguagem C têm vários exemplos na internet, porém, são muito simples, e não consegui achar nenhum que tivesse conectando mais de um servidor ao mesmo tempo. Dessa forma, tive de entender tudo do zero.

Além disso, as regras do jogo também estavam extremamente confusas e com padrões diferentes dos habituais. Por exemplo o tabuleiro, o Pokémon de defesa que fica na posição (1,0), 1 está identificando a coluna e 0 a linha. O mais comum seria ver (0,1), 0 para linha e 1 para coluna. Dessa forma, tomei a liberdade de inverter esses valores, logo, quando o comando getdefenders for mandado ao servidor, a resposta estará codificada em [linha, coluna].

Ainda falando das regras, não ficou estabelecido em nenhum momento qual seria a dimensão do tabuleiro, eu tentei entender pelos exemplos da documentação e pela imagem ilustrativa, porém, não consegui concluir como o tabuleiro poderia ter N em algum dos lados. Dessa forma, tive de entrar num consenso comigo mesmo e definir o tamanho do tabuleiro baseado nos dados que a documentação dava. Optei por produzir um tabuleiro 4x4, tanto para os atacantes quanto para os defensores.

Falando dos defensores, partindo da ideia que eles têm uma posição física (o que corrobora com a minha conclusão de que o tabuleiro seria 4x4), eu também decidi reduzir a coordenada da coluna deles. Dessa forma, o Pokémon defensor que antes estaria na posição (1,0), coluna e linha, agora está na posição (0,0), linha e coluna. Dessa forma, o alcance do ataque dos pokémon de defesa também precisou ser alterado, porém, ainda funciona como a especificação. Os pokémon de defesa que estão nas linhas 0 ou 3, isto é, nas bordas superior e inferior do tabuleiro, so conseguem acertar pokémon atacantes que estejam nas mesmas posições deles. E Pokémon de defesa que estão nas linhas do meio (linhas 1 e 2), conseguem atacar na posição que eles estão e a linha acima deles. Por exemplo, um Pokémon de defesa que esteja na posição (3,2), antigo (3,3), consegue atacar um Pokémon que esteja na posição (3,2) e na posição (2,3). Exatamente como está na especificação, porém, um pouco mais facilitado para que o jogo funcione com um tabuleiro só, ao invés de dois com dimensões diferentes.

Ainda sobre os defensores, eles tem apenas um shot por turno, isto é, caso um defensor tente acertar um pokémon na posição (3,2), mesmo que ele não consiga acertar essa posição, o shot será lançado, logo não desperdice tiros!

Os pokémon de ataque, por outro lado, não foram modificados, eles ainda andam apenas para frente e têm a quantidade de vida da especificação. Como não estava escrito na documentação como eles deveriam aparecer, nem onde, eu decidi que eles apareceriam a cada turno em cada linha da coluna 0, [(0,0), (1,0), (2,0), (3,0)], independente de como fosse a jogada de defesa. Os Pokémon de ataque têm três variantes, e essas variantes têm a mesma chance de nascer, isto é, tanto um Zubat quanto um Mewtwo tem a mesma chance de aparecer no início do tabuleiro. Além disso, eles nascem exclusivamente nas posições acima citadas.

Por fim, novamente, um dos desafios (menos desafiante do que no TP1, devo dizer) desse trabalho foi a manipulação de string em C. Já vacinado por causa do TP1, tive menos problemas com strings, porém, em alguns momentos, tive dores de cabeça com ponteiros apontando para nulo. Ainda sobre as strings, uma parte inesperada foi descobrir que a função itoa (integer to string) não está implementada na biblioteca padrão do Linux. Dessa forma, foi necessário pegar uma função itoa funcional na internet. A referência está aqui e no código.

Server

Para o arquivo server.c, há diversas varáveis criadas para fazer o controle dos soquetes, do tabuleiro, dos Pokémon. Primeiramente, transforma o protocolo e a porta enviada pelo cliente para inteiro, para que possa ser feita a soma das portas seguintes. Após, cria os soquetes dos quatro servidores. Um dos servidores recebe o start que o cliente enviou e todos os servidores respondem ao cliente com a mensagem de resposta padrão do servidor.

Para a abstração da imagem disponível na documentação, como eu já disse, fiquei muito tempo tentando entender o que era para o jogo fazer, porém, decidi mudar algumas coisas. Ambos os tabuleiros são matrizes 4x4 iniciadas com placeholders negativos. Assim que os Pokémon de defesa são alocados, os valores de suas respectivas posições mudam para 1, e para os Pokémon de ataque, o valor adicionado na matriz são os seus respectivos ids.

Após isso, o loop do jogo começa. Existem cinco opções de mensagens permitidas: getturn x, getdefenders, shot x y z, quit e todo o resto cai num quinto caso.

- Getdefenders posiciona os pokémon de defesa no tabuleiro nas suas posições estratégicas e retorna uma string com as coordenadas desses Pokémon “defender [[0, 0], [0, 2], [1, 3], [2, 1], [3, 2], [3, 3]]”. **Para deixar claro, as posições de defesa foram invertidas, isto é (y, x) virou (x, y-1)**. Foram invertidas pelo simples fato de o mais habitual de se ver é (linha, coluna) e o -1 foi para a coluna começar em 0, diferente da especificação que começava com 1. Dessa forma, ambos os tabuleiros puderam ser codificados da mesma forma, facilitando o entendimento tanto meu, quanto de quem vai corrigir o código;
- Getturn x retorna a situação atual do tabuleiro 4x4 do jogo, mostrando a posição, vida, identificador e nome dos Pokémon atacantes vivos. X precisa começar com 0 e deve ser acrescido corretamente, isto é, caso um usuário mande um getturn 0 e, na próxima rodada de getturn, algo diferente de 1, o programa encerra a conexão com os servidores e fecha o jogo sem mostrar o placar. Para a implementação, eu sinceramente não lembro mais o porquê, mas num primeiro momento, eu separei a primeira interação, turno igual a 0, e os demais turnos.
    1. Para turno igual a zero, há apenas pokémon na primeira fixed location, logo, apenas essa informação é computada, o restante é texto fixo. Para cada uma das quatro bases, gera um Pokémon para a fixed location 1 e seta no tabuleiro de atacantes o id desse pokémon recém gerado. Lembrando que no meu jogo, os Pokémon defensores nascem aleatoriamente na primeira coluna do tabuleiro em todas as linhas;
    2. Para os demais turnos, todas as fixed locations (colunas) são computadas, logo, não há texto fixo. Os pokémon que foram gerados nos turnos anteriores e não foram derrotados (quantidade máxima de hits - quantidade de hits sofridos) andam uma coluna (fixed location) para frente (ou direita). Como dito anteriormente, a cada rodada gera um novo Pokémon em cada linha (base). Logo, se o primeiro Pokémon da base 1 não foi derrotado, no segundo turno, nessa base, terão dois Pokémon.

  A cada getturn recebido, o jogo avança um turno, e as ações da seção 2 são repetidas, dando continuidade ao jogo. Se o Pokémon de ataque chega na coluna 4 e não é derrotado, ele é removido do jogo e é somado um no número de Pokémon que chegaram na Pokédex, do placar do jogo. O jogo encerra na rodada 50.

- Shot x y z, o Pokémon de defesa na posição (x, y) atira no Pokémon atacante de id z. Cada pokémon de defesa pode fazer apenas um shot por turno, mesmo se o ataque não acerte o Pokémon de ataque. Caso o Pokémon de defesa não consiga atacar o pokémon de id z, o pokémon de defesa na posição (x, y) perde sua chance de atacar nessa rodada. Caso ele consiga, é somado um na quantidade de hits daquele Pokémon e na quantidade de hits totais do tabuleiro, valor que é mostrado na mensagem de game over do jogo. Caso o Pokémon de ataque tiver apenas um ponto de vida restante e seja atacado, ele é excluído do tabuleiro. Caso contrário, ele avança uma casa pra direita, mas perde um ponto de vida.

  Relembrando as modificações nas regras, os Pokémon que estejam nas bases 1 e 4 (linhas 0 e 3) só conseguem atacar na posição que eles estão, isto é Pokémon de defesa na posição (0,0) só ataca na posição (0,0). Já para as duas linhas do meio do tabuleiro (linhas 1 e 2 ou bases 2 e 3), esse Pokémon podem atacar tanto as linhas que estão quanto uma linha imediantamente acima, isto é, pokémon no (1,3) pode atacar tanto (1,3) quanto (0,3), assim como está na documentação, porém com as modificações de notação que eu decidi adotar no meu TP;
- Quit fecha a conexão com os servidores de forma abrupta
- Qualquer outro comando encerra o jogo e mostra o placar "game over a x y z", onde a é o status, o status é 0 apenas se o jogador for até a 50a rodada, caso contrário vale 1; x é a quantidade de acertos dos Pokémon de defesa nos Pokémon de ataque, y é a quantidade de Pokémon de ataque que conseguiram alcançar a coluna 5 do tabuleiro, ou seja, chegaram na Pokédex; por fim, z é o tempo em segundos que o jogo rodou.

Client

Para o arquivo client.c, ele precisa ser obrigatoriamente executado após o servidor, para que o start funcione corretamente. Após conectar com os servidores e receber suas mensagens de sucesso de conexão, o cliente entra num loop que é o jogo em si. Nesse loop, ele aceita os comandos que rodam o jogo: getdefenders, getturn, shot, quit e qualquer outro comando encerra o jogo com a mensagem de game over. Os comandos não são case sensitive.

- Getdefenders retorna a posição dos Pokémon de defesa no tabuleiro, [linha, coluna];
- Shot x y z, o Pokémon de defesa na posição (x, y) atira no Pokémon atacante de id z. Cada pokémon de defesa pode fazer apenas um shot por turno, mesmo se o ataque não acerte o Pokémon de ataque. Caso o Pokémon de defesa não consiga atacar o pokémon de id z, o pokémon de defesa na posição (x, y) perde sua chance de atacar nessa rodada. Caso ele consiga, é somado um na quantidade de hits daquele Pokémon. Caso o Pokémon de ataque tiver apenas um ponto de vida restante, ele é excluído do tabuleiro. Caso contrário, ele avança uma casa pra direita, mas perde um ponto de vida;
- Getturn x retorna a situação atual do tabuleiro 4x4 do jogo, mostrando a posição, vida, identificador e nome dos Pokémon atacantes vivos. X precisa começar com 0 e deve ser acrescido corretamente, isto é, caso um usuário mande um getturn 0 e, na próxima rodada de getturn, algo diferente de 1, o programa encerra a conexão com os servidores e fecha o jogo sem mostrar o placar;
- Quit fecha a conexão com os servidores;

- Qualquer outro comando encerra o jogo e mostra o placar “game over a x y z”, onde a é o status, o status é 0 apenas se o jogador for até a 50a rodada, caso contrário vale 1; x é a quantidade de acertos dos Pokémon de defesa nos Pokémon de ataque, y é a quantidade de Pokémon de ataque que conseguiram alcançar a coluna 5 do tabuleiro, ou seja, chegaram na Pokédex; por fim, z é o tempo em segundos que o jogo rodou.

Todos os comandos acima são enviados para o servidor e têm suas respectivas respostas instantanemente dos servidores.

Qualquer mensagem enviada do cliente após a quinquagésima rodada será convertida em game over e encerrará o jogo com status 0.

Common

Para o common.c, temos funções úteis para o trabalho, importações de bibliotecas, structs criados para o jogo e constantes. Para as funções úteis temos as funções que fazem a conversão dos protocolos, para isso, foram usadas as funções que o Professor Ítalo nos passou na playlist de programação com soquetes. Além dessas, temos o itoa e sua função auxiliar my_reverse que fazem a conversão de um valor inteiro para string, pega do site techcrashcourse. Além dessa, também há a função de inicilizar um Pokémon, sendo Pokémon um struct que armazena o nome (string), quantidade máxima de hits, quantidade de hits sofridos e o identificador, todos esses três últimos do tipo int. Para inicilizar um Pokémon, é gerado um número aleatório, entre 0 e 2 incluso, cada um sendo um tipo de Pokémon diferente. Por fim, há a função de que gera os soquetes dos servidores, decidi passá-la para uma função separada para modularizar o código, já que seria repetida 4 vezes.

Makefile

Para o makefile, há duas possibilidades: *make* e *make clean*. O comando *make* compila o programa e o comando *make clean* apaga os arquivos executáveis e os arquivos .o.