Relatório do Trabalho Prático
Pokémon Netventure
Redes de Computadores (2023-2)

Thiago Ferronatto
Yuri Moraes Gavilan

23 de outubro de 2023

1 Introdução

O Pokémon Netventure (PokéNet) é um jogo ao espírito dos clássicos do Game Boy. Dois jogadores têm a chance de se enfrentar em batalha com a ajuda de seus monstrinhos de bolso, tudo isso à distância, já que a ação foi criada com o aspecto *multiplayer online* em mente.

Para iniciar sua jornada rumo ao título de mestre Pokémon, você vai precisar de uma conta; para isso, basta escolher um nome de usuário e uma senha, além de nos contar seu nome completo. Depois disso, o caminho estará aberto à sua frente!

Ao entrar no jogo, você poderá formar sua equipe de Pokémons e, em seguida, procurar um adversário à sua altura utilizando o mais novo sistema de criação de partidas do gênero, permitindo que você não tenha que se preocupar com encontrar um oponente; nós fazemos isso por você!

2 Visão técnica

A arquitetura lógica utilizada divide o domínio da aplicação em três partes: um servidor de autenticação, um servidor de *matchmaking*, e clientes.

2.1 Servidor de Autenticação e Informação (SAI)

O SAI tem como propósito autenticar usuários antes que eles possam prosseguir para o servidor de *matchmaking*. Isso engloba a realização de cadastros, *logins*, *logouts* e algumas outras operações extras. O uso do SAI, no entanto, não é limitado aos clientes: o servidor de *matchmaking* também faz uso de suas funcionalidades para autenticar jogadores em várias ocasiões.

O SAI responde a pedidos via TCP e, por padrão, escuta por requisições na porta 1234 do servidor onde opera[1]. É recomendado que uma conexão segura seja utilizada em qualquer comunicação com o SAI, tendo em vista a natureza das informações transeuntes. Requisições ao SAI seguem o seguinte formato: `opcode|||args`. Um `opcode` representa uma operação que o SAI pode realizar; cada operação leva diferentes argumentos, descritos no campo `args` como uma lista separada por espaços. Segue as descrições das operações disponíveis.

2.1.1 `AUTHENTICATE`

Representa uma operação de *login*. Os argumentos requeridos são um nome de usuário e uma senha. As possíveis respostas do SAI para uma requisição `AUTHENTICATE` bem-formada são

- `LOGIN_SUCCESSFUL`, que sinaliza um *login* bem-sucedido, e
- `LOGIN_FAILED`, que sinaliza alguma falha no processo.

2.1.2 `REGISTER`

Representa uma operação de cadastro. Os argumentos requeridos são um nome de usuário, uma senha e um nome pessoal. As possíveis respostas do SAI para uma requisição `REGISTER` bem-formada são

- `REGISTRATION_SUCCESSFUL`, que sinaliza um cadastro bem-sucedido, e
- `USER_ALREADY_EXISTS`, que sinaliza que o usuário em questão já está cadastrado no banco de dados.

2.1.3 `VERIFY`

Representa uma operação de verificação de *login*. Os argumentos requeridos são um *token* de autenticação e um nome de usuário. As possíveis respostas do SAI para uma requisição `VERIFY` bem-formada são

- `VERIFICATION_SUCCESSFUL`, que sinaliza que o usuário está devidamente logado e autenticado, e
- `VERIFICATION_FAILED`, que sinaliza alguma falha no processo.

[1] A título da simplicidade, durante todo o desenvolvimento tal servidor foi a máquina local, acessada pelo endereço de *loopback*.

2.1.4 LOGOUT

Representa uma operação de *logout*. Os argumentos requeridos são um *token* de autenticação e um nome de usuário. As possíveis respostas do SAI para uma requisição LOGOUT bem-formada são

- LOGOUT_SUCCESSFUL, que sinaliza que o usuário não está mais logado, e
- LOGOUT_FAILED, que sinaliza alguma falha no processo.

2.1.5 LIST_USERS

Representa uma operação de listagem dos usuários logados em determinado instante. Essa requisição não leva argumentos. A única possível resposta do SAI para uma requisição LIST_USERS bem-formada é USER_LIST, que acompanha a lista solicitada e sinaliza um fornecimento bem-sucedido.

2.2 Servidor de *Matchmaking* (SMM)

O objetivo do SMM é criar partidas entre duplas de jogadores autenticados pelo SAI. Para tal, ele mantém uma fila de usuários que desejam jogar; quando essa fila tem dois ou mais usuários, os dois localizados na frente da fila são contatados pelo SMM para confirmar a criação da partida. Caso algum deles recuse, este será removido da fila, enquanto o outro continuará ali. Caso ambos aceitem, eles são removidos da fila, e as informações de um são enviadas ao outro pelo SMM para que a partida aconteça via P2P entre ambos.

Requisições ao SMM, que está disponível pela porta 1235 na máquina em que opera, são feitas via UDP, mas seguem o mesmo formato das feitas ao SAI, porém, aqui, os argumentos são sempre um token de autenticação do SAI e um nome de usuário; ao realizar qualquer operação, o SMM se comunica com o SAI para determinar a validade da sessão do usuário que a solicitou. Se a sessão for inválida, o SMM sempre irá responder ao usuário com USER_NOT_AUTHENTICATED. A seguir se encontram as operações realizadas pelo SMM.

2.2.1 MAKE_AVAILABLE

Representa uma operação de inserção na fila de *matchmaking*; um usuário está "se fazendo disponível" para partidas. A única resposta do SMM para uma requisição MAKE_AVAILABLE bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é IN_QUEUE, que sinaliza que o usuário está na fila esperando por uma partida.

2.2.2 MAKE_UNAVAILABLE

Representa uma operação de remoção da fila de *matchmaking*; um usuário deseja encerrar sua busca por partidas. A única resposta do SMM para uma requisição MAKE_UNAVAILABLE bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é OUT_OF_QUEUE, que sinaliza que o usuário foi removido da fila com sucesso.

2.2.3 ACCEPT_MATCH

Representa o aceite de uma partida previamente oferecida pelo SMM. A única resposta do SMM para uma requisição ACCEPT_MATCH bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é MATCH_READY, que acompanha os dados do segundo jogador e sinaliza que a partida está pronta para começar; os dados da relevantes são inseridos na lista de partidas ativas.

2.2.4 DECLINE_MATCH

Representa a recusa de uma partida previamente oferecida pelo SMM. A única resposta do SMM para uma requisição DECLINE_MATCH bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é FAILED_TO_ACCEPT, que sinaliza que o jogador não aceitou a partida que lhe foi oferecida e, portanto, será removido da fila.

2.2.5 LIST_PLAYERS

Operação deprecada; não utilizar.

2.2.6 LIST_MATCHES

Representa uma listagem das partidas ativas no momento do pedido. A única resposta do SMM para uma requisição LIST_MATCHES bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é MATCH_LIST, que acompanha a lista solicitada e sinaliza um fornecimento bem-sucedido.

2.2.7 MATCH_ENDED

Representa um aviso de um jogador, informando que a partida em que ele estava envolvido acabou; o SMM pode, agora, remover tal partida da lista de partidas ativas. A única resposta do SMM para uma requisição MATCH_ENDED bem-formada, supondo que o usuário seja devidamente validado pelo SAI, é MATCH_ENDED, que sinaliza uma operação bem-sucedida.

2.3 Clientes

Os clientes, aqui, tratam de toda a lógica do jogo em si. Menus são mostrados na tela, fornecendo opções de autenticação e de requisições a ambos os servidores. Quando uma partida é aceitada por dois jogadores pelo SMM, uma interface gráfica é mostrada na aplicação cliente para que o usuário tenha um guia visual para escolher sua equipe. Após a escolha das equipes, a simulação da batalha é iniciada. A conexão entre clientes é direta (P2P) e estabelecida sobre o TCP.

2.4 Sequência de eventos

Em um mundo ideal, a seguinte sequência é um bom exemplo de como um cenário alvo de uso ocorreria.

1. SAI é iniciado;
2. SMM é iniciado;
3. c_1 é iniciado;
4. c_1 envia `REGISTER` ao SAI;
5. SAI cadastra c_1;
6. Caso o resultado seja `REGISTRATION_SUCCESSFUL`, o SAI automaticamente realiza a operação `AUTHENTICATE` com os dados de c_1;
7. SAI envia `LOGIN_SUCCESSFUL` a c_1, juntamente a um *token* t_1;
8. c_1 utiliza t_1 para enviar `MAKE_AVAILABLE` ao SMM;
9. SMM envia `IN_QUEUE` a c_1;
10. c_2 é iniciado;
11. c_2 envia `REGISTER` ao SAI;
12. SAI cadastra c_2;
13. Caso o resultado seja `REGISTRATION_SUCCESSFUL`, o SAI automaticamente realiza a operação `AUTHENTICATE` com os dados de c_2;
14. SAI envia `LOGIN_SUCCESSFUL` a c_2, juntamente a um *token* t_2;
15. c_2 utiliza t_2 para enviar `MAKE_AVAILABLE` ao SMM;
16. SMM envia `IN_QUEUE` a c_2;
17. SMM envia `MATCH_AVAILABLE` a c_1;
18. SMM envia `MATCH_AVAILABLE` a c_2;
19. c_1 envia `ACCEPT_MATCH` ao SMM;
20. c_2 envia `ACCEPT_MATCH` ao SMM;
21. SMM envia `MATCH_READY` a c_1;
22. SMM envia `MATCH_READY` a c_2;
23. c_1 abre uma conexão P2P via TCP com c_2;
24. A partida progride entre c_1 e c_2 até seu fim;
25. A conexão entre c_1 e c_2 é encerrada;
26. c_1 envia `MATCH_ENDED` ao SMM;
27. c_1 envia `LOGOUT` ao SAI;
28. c_2 envia `LOGOUT` ao SAI;
29. SAI envia `LOGOUT_SUCCESSFUL` a c_1;
30. SAI envia `LOGOUT_SUCCESSFUL` a c_2;
31. c_1 é encerrado;
32. c_2 é encerrado;
33. SMM é encerrado;
34. SAI é encerrado.

[2]No momento da escrita, essa seria a versão 3.12.

3 *Features* implementadas

Todos os itens da descrição do trabalho foram implementados – embora haja diferenças no modo pelo qual tais objetivos foram atingidos – exceto pelas observações F e G.

3.1 *Features* extras

As diferenças mencionadas se dão na forma do SMM. Como há, aqui, um servidor que lida com todos os jogadores interessados, não há necessidade de contato cliente-cliente antes de a partida ser iniciada. Tal abordagem é uma evolução natural da clássica "lista *free for all*" utilizada em jogos mais antigos; o *matchmaking* retira dos usuários o trabalho de procurar manualmente por uma partida, delegando tal função a um servidor especializado. Isso é amplamente utilizado no contexto hodierno; alguns exemplos de jogos que empregam tal estratégia são: Counter-Strike™ 2, Valorant™, Apex Legends™ etc.

Um servidor de *matchmaking* também permite que outros fatores sejam levados em conta no momento da criação de uma partida, como a habilidade dos jogadores – sistemas reais implementam modificações do sistema de Elo do xadrez, como o Glicko-2, a fim de medir a habilidade de seus jogadores e criar partidas mais coerentes –, para tornar o ambiente competitivo o mais justo possível. Isso simplesmente não é possível quando são os clientes que escolhem seus oponentes.

A natureza dessa tática torna alguns aspectos, que sem ela seriam absolutamente necessários, obsoletos. Um exemplo é a listagem de jogadores; isso não é mais requerido, já que quem decide os participantes de uma partida é o servidor. Ainda assim, implementamos essa funcionalidade por meio do SAI; ele fornece uma lista de jogadores atualmente *online*, porém, com seus IPs e portas omitidos, já que não são relevantes para o jogo.

4 Instruções

Aqui é detalhado o processo de preparação e de execução dos programas.

4.1 Preparação

É necessário que o Python 3 esteja instalado nas máquinas que irão executar os *scripts*; mais especificamente, recomenda-se que a versão mais recente[2] esteja presente.

Uma biblioteca também é necessária para a execução do SAI: Argon2; ela é responsável por criar um *hash* seguro – estado-da-arte, para ser mais preciso – para o armazenamento das senhas dos usuários.

4.2 Execução

O projeto é composto por três arquivos principais: `ias.py`, `mms.py` e `player.py`, representando, respectivamente, o SAI, o SMM, e o cliente. O SAI e o SMM podem ser iniciados em qualquer ordem, desde que os clientes só sejam iniciados depois de ambos os servidores estarem disponíveis.

Daí em diante, basta escolher as opções desejadas no menu do cliente. Os servidores mantêm um *log*, tanto no terminal em que executam quanto em arquivos (`ias.log`, `mms.log`), dos eventos encadeados pelos clientes.

5 Código

Todo o código, que foi desenvolvido em conjunto pelos autores, está disponível publicamente no repositório `thiagoferronatto/trabalho-redes-2023-2`, no GitHub. Por ali, também pode ser observada a distribuição de tarefas feita entre os desenvolvedores por meio dos *commits* efetuados.