Universidade Federal de Minas Gerais
DCC023: Redes de Computadores
Trabalho Prático

Introdução

Esta prática tem como meta principal introduzir os conceitos de comunicação UDP com transmissão e perda de pacotes em uma comunicação cliente - servidores. Os alunos devem ser responsáveis por implementar tanto o cliente quanto os servidores descritos neste trabalho. Dentre os objetivos gerais deste trabalho, destacam-se:

- Introduzir o conceito de multiplexação de comunicação/soquetes.
- Reforçar conceitos de codificação de dados.
- Introduzir o conceito de detecção de erros e retransmissão de pacotes.

Neste trabalho continuaremos a jornada no mundo dos Pokémons e iremos desenvolver o novo jogo Pokémon Go New. A Pokédex está em perigo e para protegê-la será necessário distribuir alguns dos Pokémons mais fortes para defender. Os Pokémons de defesa vão ficar em locais fixos para defender a Pokédex enquanto os Pokémons adversários vão se movimentar em linha reta a cada rodada que não forem destruídos. Além disso, podem surgir novos Pokémons adversários para chegar até a Pokédex. Não se preocupe, o Pokémon de defesa será o mais forte. O jogo possui um conjunto simples de regras e um protocolo de comunicação que os alunos deverão implementar. Vale ressaltar que **não** será necessário reutilizar o código do trabalho anterior nesta implementação, assim tenha em mente a implementação do trabalho independente do anterior.

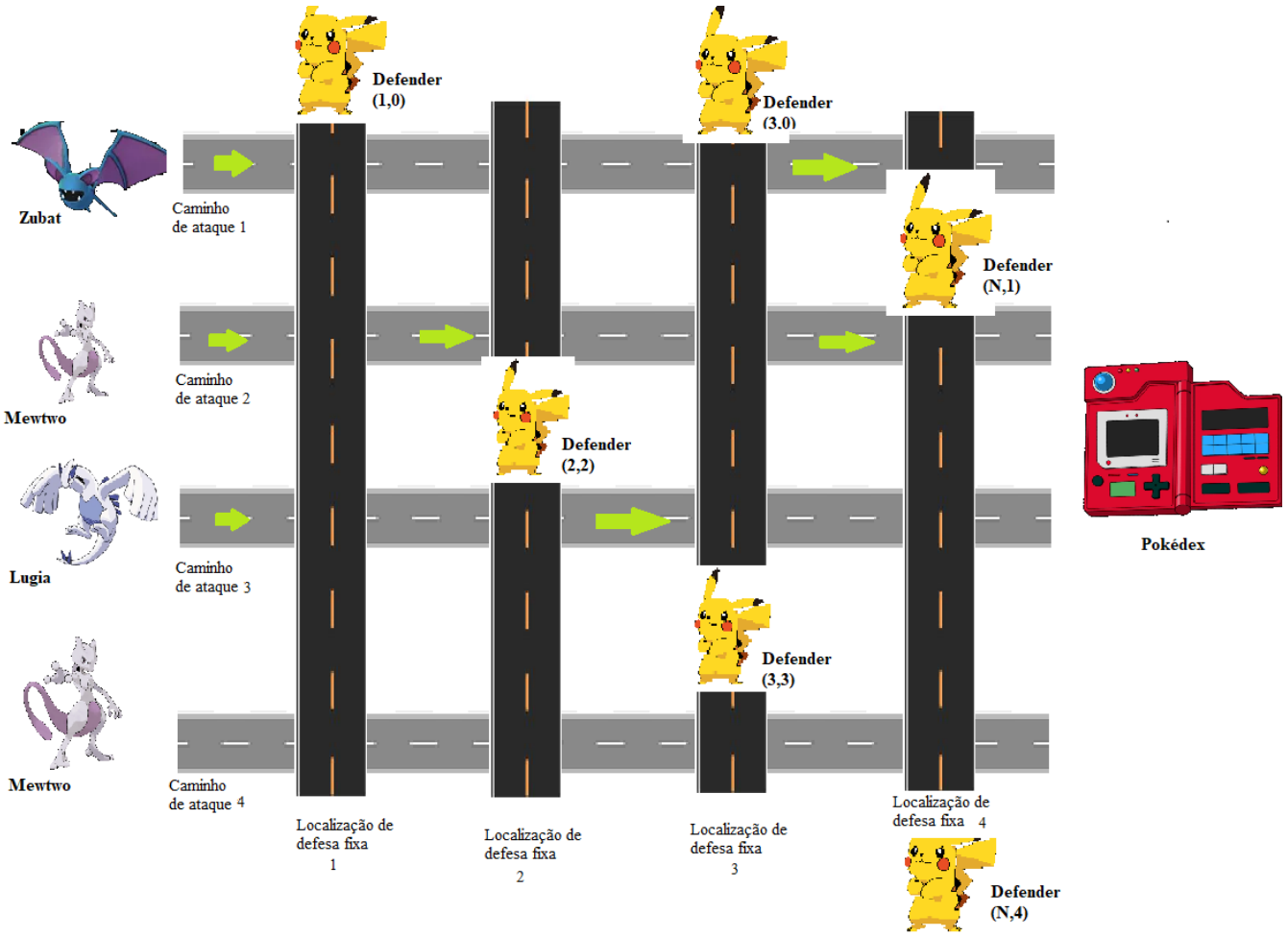


Figura 1 - Mapa de exemplo em alto nível do jogo que será desenvolvido.

Regras

O **Pókemon Go New** é um jogo onde o jogador (representado pelo seu programa) controla **os ataques Pokémons** contra os Pokémons adversários que desejam chegar na **Pokédex**. O objetivo do Pokémon adversário é entrar na Pokedex e soltar os Pokémons lá existentes. O Pokémon adversário se movimenta para frente em uma **base de batalha numérica de 1 a 4**. Os Pokémons de defesa estão em locais estratégicos para efetuar ataques de proteção na entrada da Pokédex. **Os locais estratégicos de defesa são numerados de 1 a 4.** A localização dos Pokémons de defesa são informadas através de coordenadas. A Figura 1 ilustra em alto nível o ambiente que corresponde ao jogo, e portanto a interação entre as partes do sistema a ser desenvolvido..

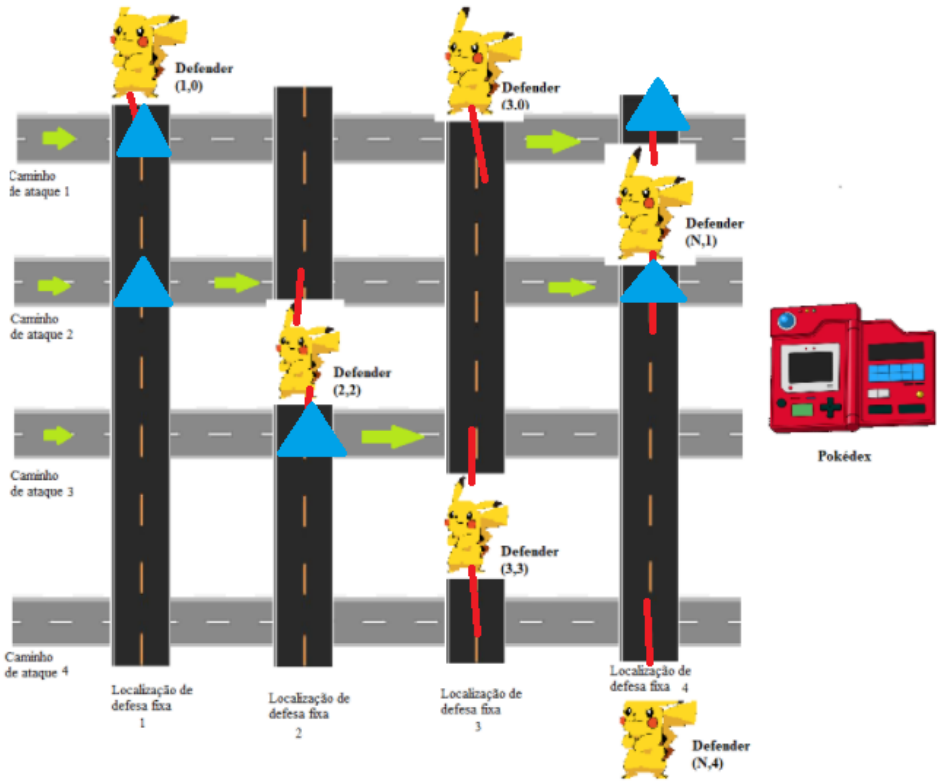
O jogo procede em rodadas. O início de uma rodada é disparado pelo jogador (seu programa). Em cada rodada os seguintes eventos ocorrem:

1. Os primeiros Pokémon adversários aparecem e realizam a primeira movimentação estratégica. **Cada Pokémon adversário não atingido pelo ataque de proteção pode se movimentar em direção a Pokédex.**
2. Cada **Pokémon de defesa pode efetuar um ataque por rodada**. Um Pokémon de defesa pode atacar o Pokémon adversário (Lembrete: O adversário se movimenta a cada jogada). A tabela abaixo mostra um exemplo do número de Pokémons adversário que cada Pokémon de defesa pode alvejar:

Pokemon de Defesa	**Pokemon Adversário**
(1, 0)	1
(3, 0)	0
(N, 1)	2
(2, 2)	1
(3, 3)	0
(N, 4)	0

Formato da coordenada: (número do caminho vertical, número do caminho horizontal)
*N é o número do caminho que está na vertical

A imagem abaixo representa o posicionamento da tabela de exemplo. Os triângulos em azul representam os pokémons de ataque.



O jogo possui três tipos de Pokémons adversários listados a seguir.

- **Mewtwo:** o grande antagonista da primeira geração de Pokémon se destaca pelo seu extraordinário poder psíquico. Além de voar através do uso de telecinesia, ele também pode falar telepaticamente e assumir o controle de outro ser vivo com os poderes da mente. Para destruí-lo é necessário 3 ataques na mesma rodada;
- **Lugia:** inteligente e ágil, esse monstrinho é considerado o líder dos pássaros lendários. Para destruí-lo é necessário 2 ataques na mesma rodada;
- **Zubat:** conhecido por sua facilidade de evolução para Golbat. Para destruí-lo é necessário 1 ataque na mesma rodada.

Para pontuar, os servidores devem contabilizar:
1º) O número de Pokémons adversários que foram atingidos e destruídos pelos ataques de proteção
2º) O número de Pokémons adversários que chegaram na Pokédex
3º) O tempo total em segundos utilizados pelo programa cliente para completar o jogo

Protocolo

O protocolo utilizado no trabalho é o UDP. Todas as mensagens são um array. O programa cliente é responsável por implementar um temporizador para detectar problemas de transmissão e retransmitir pacotes perdidos.

O protocolo considera que cada base de batalha é controlada por um servidor diferente. O programa cliente deve conectar-se e executar o protocolo com quatro servidores simultaneamente, recebendo informações sobre Pokémon adversários e informando as defesas realizadas a cada servidor separadamente.

Requisição de início de jogo

Um cliente inicia uma conexão com o servidor enviando uma mensagem de jogo passando os seguintes parâmetros:

```
./client <host> <first port> start
```

O parâmetro `<first port>` refere-se a porta do primeiro servidor. **O cliente deve conectar-se automaticamente nas demais portas.**
O cliente deve enviar uma mensagem de início de jogo para cada um dos servidores e confirmar a resposta de cada um dos quatro servidores do jogo (conforme a próxima orientação).

Resposta de início de jogo

Após a conexão estabelecida (cliente com os 4 servidores): **cada** um dos servidores confirma o recebimento da mensagem de **início de jogo** enviada pelo cliente. A cada conexão bem sucedida, cada servidor envia uma mensagem ao cliente:

```
game started: path 1
game started: path 2
game started: path 3
game started: path 4
```

Requisição de posicionamento de Pokémons defensores [getdefender]

O programa cliente deve iniciar o jogo enviando uma mensagem de requisição de posicionamento de Pokémons defensores. Isto deve ser feito apenas após o cliente conectar-se com todos os servidores, o que pode ser verificado **pelo recebimento de pelo menos uma**

resposta de início de jogo (“game started: path x”, de qualquer servidor) **de cada um dos quatro servidores**. Uma mensagem `getdefender` tem apenas a string:

```
getdefenders
```

`O cliente precisa enviar apenas uma mensagem de posicionamento de Pokémons defensores a qualquer um dos servidores.` O cliente precisa enviar apenas uma mensagem `getdefender` para um servidor no começo do jogo. Os Pokémons defensores permanecem fixos durante a execução do jogo.

Resposta de posicionamento dos Pokémons defensores

Qualquer servidor responde a uma requisição de posicionamento de Pokémons defensores com uma resposta de posicionamento do Pokémon defensor. A resposta contém um campo `defender` com uma lista de coordenadas indicando a localização fixada e o caminho adversário onde o Pokémon defensor está localizado. A posição de um Pokémon defensor na coordenada referente aos caminhos do Pokémon adversário vai de 0 a 4 (como na figura 1). **Um Pokémon defensor pode atirar Pokémons adversários em posições adjacentes**; por exemplo, o Pokemon defensor com coordenada (X, 1) pode atirar em um Pokémon adversário passando pela localização de defesa fixa X nos caminhos 0 e 1. Os Pokémons defensores com índices 0 e 4 podem atirar apenas em Pokémon adversário nos caminhos 1 e 4, respectivamente. Por exemplo, para os Pokémons defensores mostrados na figura acima, um servidor enviará a seguinte resposta de posicionamento de Pokémons defensores:

```
defender [[1, 0], [3, 0], [N, 1], [2, 2], [3, 3], [N, 4]]
```

Requisição de estado de rodada [getturn]

O programa cliente deve iniciar o jogo enviando **uma mensagem** de requisição de estado de rodada (`getturn`) **para um servidor apenas** após o recebimento da mensagem de início de jogo de todos os servidores `game started: path x`. **Para isso, o cliente deve ter recebido pelo menos uma resposta de confirmação (de qualquer servidor)**.

As rodadas são numeradas a partir de zero e devem ser **requisitadas em ordem**. O número da rodada é indicado após a palavra `getturn`. Após estabelecer conexão com os quatro servidores, o cliente deve requisitar a rodada zero, por exemplo:

```
getturn 0
```

Quando estiver pronto para avançar para a próxima rodada, o cliente deve enviar nova mensagem `getturn` para o próximo rodada. Por exemplo, após terminar a rodada zero, o cliente pode requisitar a próxima rodada com a mensagem a seguir:

```
getturn 1
```

Uma mensagem `getturn` enviada a um servidor avança de rodada em todos os servidores de forma síncrona.

Mensagem de estado [state]

Um servidor responde a uma requisição de estado da rodada com várias mensagens de estado. Cada mensagem de estado é referente a rodada requisitada (indicado no campo `turn`) e uma localização estratégica do Pokémon defensor (indicada no campo `fixedLocation`). Uma mensagem de estado contém, após o `fixedLocation`, uma lista de Pokémons adversários passando por uma localização de defesa fixa em uma rodada. Lembre-se que cada servidor controla uma base de batalha específica, o que determina em qual base de batalha o Pokémon adversário está. Quando nenhum Pokémon adversário chega a uma localização de defesa fixa em uma rodada, a lista de Pokémons deve ser vazia. Por exemplo, o servidor que controla a base de batalha 2 (*caminho de ataque 2*) na Figura 1 de exemplo irá responder a uma mensagem `getturn` com as seguintes mensagens de estado (supondo que o número do rodada é 1337):

```
Base 2
turn 1337
fixedLocation 1
45 Mewtwo 0
…

turn 1337
fixedLocation 2
<id> <name> <hits>
…

turn 1337
fixedLocation 3
<id> <name> <hits>
…

turn 1337
fixedLocation 4
<id> <name> <hits>
 …
```

Onde os objetos referentes aos Pokémon adversários são especificados como na próxima seção e estão representados acima com "…" . Note que segundo as regras do jogo, vários Pokémon adversários podem estar passando por uma localização de defesa fixa em um caminho. Para obter o estado completo de um rodada, o cliente deve enviar uma mensagem `getturn` para cada servidor.

Codificação de um Pokémon adversário

Um Pokémon adversário é um array que contém três campos. O campo `id` é um número inteiro identificador único para aquele Pokémon adversário (o identificador é mantido constante quando um Pokémon adversário avança de uma localização de defesa fixa para a próxima localização de defesa fixa entre rodadas). O campo `name` identifica a classe de Pokémon adversário e pode ter três valores: "zubat", "lugia" e "mewtwo". O campo `hits` indica quantos ataques o Pokémon adversário já recebeu. (Lembre-se que cada Pokémon tem um valor conforme especificado em "Regras") Por exemplo:

```
id name hits
```

Cada resposta de estado pode ter 0 ou n Pokémons no caminho. Cada mensagem de Pokémon estará em uma linha.

Mensagem de defesa [shot]

Após receber o estado de um rodada, o cliente deve decidir em quais Pokémon adversários cada Pokémon defensor deve atirar, e enviar ao servidor que está controlando o Pokémon adversário uma mensagem de defesa. Uma mensagem de defesa tem tipo `shot` e identifica o Pokémon defensor atirando através de um campo `defender` que deve conter uma lista de dois elementos indicando as coordenadas do Pokémon defensor. Por fim, a mensagem contém um campo `id` com o identificador do Pokémon adversário alvejado. Por exemplo:

```
shot <defender position N> <defender position 0-4> <id>
shot 2 2 35
```

Resultado de defesa [shotresp]

Uma mensagem de defesa é respondida pelo servidor com uma mensagem indicando se a defesa é válida ou se ocorreu algum erro. Uma mensagem de resultado de defesa contém o tipo `shotresp`; o identificador do Pokémon defensor (`defender`); o identificador do Pokémon adversário (`id`); e um campo `status` com valor 0(zero) em caso de sucesso e com valor 1(um) em caso de falha. Por exemplo:

```
shotresp <defender position x> <defender position y> <id> <status>
shotresp 2 2 35 0
```

Mensagem de fim de jogo [gameover]

Servidores que receberem mensagens inválidas de um cliente ou sem que um jogo esteja em andamento receberão uma mensagem de fim de jogo. **Após receber uma mensagem de fim de jogo, o cliente deve iniciar a execução do protocolo novamente a partir da fase de início de jogo.**

O servidor também envia uma mensagem de fim de jogo em resposta a uma mensagem de requisição de estado de rodada (`getturn`) após o fim do jogo.

Uma mensagem de fim de jogo contém um campo com valor `gameover`; um campo `status` com valor zero caso o jogo tenha terminado após execução de todas as rodadas ou com valor 1 em caso de terminação precoce; e um campo `score` com várias métricas de desempenho do cliente durante o jogo. Por exemplo:

```
gameover <status> <score>
gameover 0 200 100 23000
```

O score contém os parâmetros das regras, onde:

- `200` o número de Pokémons adversários atingidos e destruídos pelos defensores
- `100` o número de Pokémons adversários que chegaram na Pokédex
- `23000` o tempo total em segundos utilizados pelo programa cliente para completar o jogo

Fechamento de jogo [quit]

Um cliente pode sair de um jogo enviando uma mensagem de fechamento de jogo a qualquer um dos servidores. Em particular, o cliente deve fechar o jogo após receber uma mensagem de `gameover` com `status` zero (i.e., quando o jogo acaba).

Uma mensagem de fechamento de jogo enviada a um servidor irá fechar o jogo em todos os servidores. Uma mensagem de fechamento de jogo possui a palavra `quit`, por exemplo:

```
quit
```

Controle de execução

O controle de execução é de responsabilidade do cliente, que deve primeiro enviar uma mensagem de início de jogo, conectar-se com o servidor, requisitar o posicionamento de

Pokémons defensores, e avançar as rodadas. Apesar de não ser necessário, recomendamos que o cliente envie mensagens de defesa e confira as mensagens de resultado de defesas para todos os Pokémons defensores antes de avançar para a próxima rodada.

Note que a comunicação com os servidores está sujeita a erros, de forma que mensagens podem ser perdidas. Caso alguma mensagem seja perdida, é de responsabilidade do cliente transmitir novamente. Note que o algoritmo de retransmissão é parte importante da avaliação do seu programa e deve estar detalhado na documentação. Em particular, o servidor contabiliza o número de mensagens enviadas e o **tempo total** para terminar o jogo. Enviar muitas mensagens de retransmissão desnecessariamente tem impacto negativo na avaliação, e esperar demasiadamente antes de enviar uma retransmissão para um pacote perdido irá aumentar o tempo de execução e também afetar negativamente a avaliação. Você deve projetar e desenvolver um algoritmo de retransmissão adequado para o trabalho.

Desenvolvimento

Servidores

Os servidores devem ser inicializados:

```
./server <host> <port1>
```

`port1` é a porta do primeiro servidor. Com este comando o servidor deve ser capaz de iniciar todos os quatro servidores (um para cada caminho).

Cliente

A interface de usuário inicia com a requisição de início de jogo explicada anteriormente.

```
./client <host> <first port> start
```

Detalhes de implementação

- Clientes devem implementar um algoritmo de retransmissão que tente reduzir o número de retransmissões e o tempo de execução total do programa.
- Programas clientes devem funcionar com protocolo IPv4 ou IPv6 (utilize apenas funções que tenham suporte aos dois protocolos).

Sugestão de implementação e depuração

Implemente uma requisição por vez. Comece pelas requisições de início de jogo, seguida pela requisição de fechamento do jogo, e depois vá implementando as requisições na ordem

necessária para fazer progresso no jogo. Assim você pode ir testando seu programa enquanto vai desenvolvendo. Durante o processo de depuração, considere comparar suas mensagens com as mensagens enviadas por clientes de colegas para verificar se os valores batem.

Entrega e avaliação

Este trabalho deve ser realizado individualmente. Seu programa deve rodar no sistema operacional Linux e, em particular, não deve utilizar bibliotecas do Windows, como o winsock. Jogue o jogo por **50 rodadas**. Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor.

Entrega

Deve ser entregue o código fonte na **linguagem C** e **um arquivo README com as instruções para compilação (se houver) e execução**. Além disso, deve ser entregue também um arquivo PDF de documentação, descrito a seguir. **Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.**

Cada aluno deve entregar, além da documentação, o **código fonte em C** e um **Makefile** para compilação do programa. Instruções para a submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado `client` e os servidores em um binário chamado `server`.
- Seu programa deve ser compilado ao se executar apenas o comando `make`, ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP2_MATRICULA.zip
- O nome dos arquivos deve ser padronizado:
    - `server.c`
    - `client.c`
    - `common.c, common.h (se houver)`

Documentação

Cada aluno deverá entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir **dificuldades e imprevistos de projeto, bem como as soluções adotadas para os problemas**. A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente

Testes

Pelo menos os testes abaixo serão realizados durante a avaliação do seu emulador:

- Conexão com os quatro servidores
- Execução de cada um dos comandos descritos do jogo, bem como avaliação das mensagens retornadas
- Retransmissão de pacotes pelo cliente
- Conexão via IPv4 ou IPv6
- Terminação do jogo com `quit` e com número máximo de rodadas (50 rodadas)

Desconto de nota por atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2^{d-1} \div 0.32\ \%$$

onde *d* é o atraso em dias úteis. Note que após 5 dias, o trabalho não pode ser mais entregue.

Exemplo

A seguir, alguns exemplos de como se dará a execução do jogo com o cliente e o servidor operantes em relação ao cenário descrito na Figura 1.

Server

```
./server v4 9000
```

Inicializa os demais servidores nas portas 9001, 9002, 9003

Client

```
./client 127.0.0.1 9000 start

< game started: path 1
< game started: path 2
< game started: path 3
< game started: path 4
> getdefenders
< defender [[1, 0], [3, 0], [4, 1], [2, 2], [3, 3], [4, 4]]
```

RODADA 0

```
> getturn 0
< base 1
  turn 0
  fixedLocation 1
  1 Zubat 0

  turn 0
  fixedLocation 2

  turn 0
  fixedLocation 3

  turn 0
  fixedLocation 4
< base 2
  turn 0
  fixedLocation 1
  2 Mewtwo 0

  turn 0
  fixedLocation 2
```

```
  turn 0
  fixedLocation 3

  turn 0
  fixedLocation 4
< base 3
  turn 0
  fixedLocation 1

  turn 0
  fixedLocation 2
  3 Lugia 0

  turn 0
  fixedLocation 3

  turn 0
  fixedLocation 4
< base 4
  turn 0
  fixedLocation 1

  turn 0
  fixedLocation 2
  4 Mewtwo 0

  turn 0
  fixedLocation 3

  turn 0
  fixedLocation 4
> shot 1 0 1
< shotresp 1 0 1 0
```

RODADA 1

```
> getturn 1
< base 1
  turn 1
  fixedLocation 1
  5 Zubat 0            // Zubat morre com 1 hit, apareceu um novo Zubat

  turn 1
```

```
  fixedLocation 2

  turn 1
  fixedLocation 3

  turn 1
  fixedLocation 4
< base 2
  turn 1
  fixedLocation 1

  turn 1
  fixedLocation 2
  2 Mewtwo 0

  turn 1
  fixedLocation 3

  turn 1
  fixedLocation 4
< base 3
  turn 1
  fixedLocation 1

  turn 1
  fixedLocation 2
  3 Lugia 0

  turn 1
  fixedLocation 3

  turn 1
  fixedLocation 4
< base 4
  turn 1
  fixedLocation 1

  turn 1
  fixedLocation 2
  4 Mewtwo 0

  turn 1
  fixedLocation 3
```

```
  turn 1
  fixedLocation 4
> shot 2 2 3
< shotresp 2 2 3 0
```

RODADA 2

```
> getturn 2
< base 1
  turn 2
  fixedLocation 2
  5 Zubat 0

  turn 2
  fixedLocation 2

  turn 2
  fixedLocation 3

  turn 2
  fixedLocation 4
< base 2
  turn 2
  fixedLocation 1

  turn 2
  fixedLocation 2

  turn 2
  fixedLocation 3
  2 Mewtwo 0

  turn 2
  fixedLocation 4
< base 3
  turn 2
  fixedLocation 1

  turn 2
  fixedLocation 2

  turn 2
  fixedLocation 3
```

```
  3 Lugia 1          // O Pokemon anda uma casa a cada turno, mesmo quando atacado

  turn 2
  fixedLocation 4
< base 4
  turn 2
  fixedLocation 1

  turn 2
  fixedLocation 2

  turn 2
  fixedLocation 3
  4 Mewtwo 0

  turn 2
  fixedLocation 4
> shot 2 2 3
< shotresp 2 2 3 0
> shot 3 3 4
< shotresp 3 3 4 0
> shot 4 4 5
< shotresp 4 4 5 1   // Erro - Pokémon defensor não consegue atacar esse adversário
```

RODADA 3

```
> getturn 3
< base 1
  turn 3
  fixedLocation 2

  turn 3
  fixedLocation 2
  5 Zubat 0

  turn 3
  fixedLocation 3

  turn 3
  fixedLocation 4
< base 2
  turn 3
  fixedLocation 1
```

```
  turn 3
  fixedLocation 2

  turn 3
  fixedLocation 3

  turn 3
  fixedLocation 4
  2 Mewtwo 0         // Conseguirá chegar na Pokédex pois não houve defesa
< base 3
  turn 3
  fixedLocation 1
  6 Zubat 0

  turn 3
  fixedLocation 2

  turn 3
  fixedLocation 3

  turn 3
  fixedLocation 4
< base 4
  turn 3
  fixedLocation 1

  turn 3
  fixedLocation 2

  turn 3
  fixedLocation 3

  turn 3
  fixedLocation 4
  4 Mewtwo 1
> shot 3 3 4
< shotresp 3 3 4 0
> shot 4 4 4
< shotresp 4 4 4 0
> get                  // Mensagem inválida finaliza o jogo
< gameover 3 1 10
> quit                 // Encerra a conexão
```