Departamento de Ciência da Computação – IME-USP

MAC2166 Introdução à Computação
Escola Politécnica – Primeiro Semestre de 2018

Segundo Exercício-Programa (EP2)
Data máxima para entrega: 19/5/2018
versão 1.1

19 de abril de 2018

"Evacuate the city! Engage all defenses!"
T'Challa em Avengers: Infinity War

"Meeting an advanced civilisation could be like Native Americans encountering Columbus. That didn't turn out so well."
Stephen Hawking

1 Introdução

Em 1978, a *Taito Corporation* lançou um jogo eletrônico de arcade que revolucionou a indústria de jogos eletrônicos e que se tornaria um ícone da cultura pop alguns anos depois: *Space Invaders*. No jogo em duas dimensões, criado por Tomohiro Nishikado, o jogador controla um canhão que emite lasers contra naves alienígenas, que por sua vez atiram lasers contra o canhão. O canhão e as naves alienígenas são destruídos quando atingidos por um laser. Uma partida do jogo termina quando o jogador não tiver mais nenhum canhão, quando alguma nave alienígena alcançar a parte inferior da tela, ou quando o jogador conseguir destruir todas as naves. Nos dois primeiros casos o jogador perde a partida. No último caso, ele vence. O canhão do jogador fica localizado na parte inferior da tela e a movimentação dele pelo jogador é sempre horizontal, enquanto as principais naves alienígenas começam na parte superior da tela, se movimentam na horizontal e vão descendo quando atingem um canto lateral da tela. O jogo tem algumas outras características que não serão detalhadas aqui porque fugiria do escopo do EP mas se você tiver curiosidade para ver uma partida do *Space Invaders* original, veja aqui.

A tarefa neste EP é implementar um jogo de *Space Invaders* mais simples do que o original. O principal objetivo com essa implementação é exercitar a programação com

listas e funções em python. A cada iteração do jogo, o usuário poderá mover o canhão que emite lasers ou emitir lasers, enquanto as naves alienígenas se movem de acordo com o padrão de movimentação das principais naves do jogo original e atiram de forma pseudo-aleatória.

Dois objetivos secundários são: exercitar a implementação passo a passo de um problema (no caso, função a função) e aprender a seguir padrões pré-definidos em um código-fonte. A implementação passo a passo remete à técnica de divisão e conquista, relacionada por exemplo a táticas militares em que pequenas porções de um exército inimigo são atacadas em sequência ao invés de todo o exército ser atacado de uma só vez. A implementação seguindo padrões pré-definidos remete a situações em que alguém entra em uma equipe já existente para finalizar algum projeto. Nesse caso é necessário concluir as partes faltantes do projeto aproveitando aquilo que já tiver sido feito e respeitando os padrões pré-definidos por aqueles que já estavam no projeto desde o seu início para evitar que tudo que já foi feito "quebre".

2 Regras do jogo

É importante observar que na implementação do EP você só poderá usar recursos do python apresentados em sala de aula pelo seu professor. Leia as instruções gerais para entrega de EPs em https://www.ime.usp.br/~mac2166/infoepsPy/

A única informação passada como entrada pelo usuário antes da partida do jogo começar é a quantidade de naves alienígenas, que será um número inteiro maior que 1 e menor que 53. O canhão do jogador ocupará a linha inferior da área do jogo e estará inicialmente na coluna do meio. As naves sempre começarão a partida no canto superior esquerdo da área do jogo e alinhadas em pares com 1 espaço em branco entre cada par. Por exemplo, se o valor passado pelo usuário for 2, e considerando que uma nave é representada pelo caracter `V`, a linha 0 e a linha 1 da área do jogo terão este conteúdo no início da partida:

```
V
V
```

Se o valor for 6, a linha 0 e a linha 1 da área do jogo terão este conteúdo no início da partida:

```
V V V
V V V
```

Caso o valor passado como entrada seja um valor ímpar qualquer, $2n + 1$, haverá n pares de naves e 1 única nave na coluna mais à direita ocupada pelas naves. Por exemplo, se o valor for 5, a linha 0 e a linha 1 da área do jogo terão este conteúdo no início da partida:

```
V V V
V V
```

A qualquer momento do jogo, as naves terão uma *direção de movimento* que indica para onde elas se movem. As direções possíveis são *direita*, *esquerda* e *baixo*. No início da partida a direção é *direita* até o momento em que alguma nave alcançar a coluna mais à direita da área do jogo. Nesse momento, em que uma nave alcançou a última coluna à direita, ela e as demais movem-se uma linha para *baixo* e passam a se mover para a *esquerda*. De modo similar, uma nave que alcançou a coluna 0 e as demais, movem-se uma linha para baixo e passam, todas elas, a moverem-se para a *direita* e o processo se repete.

Cada iteração (também chamada no restante deste enunciado de **rodada**) `i` do jogo prossegue da seguinte forma, considerando que no início, a `rodada` vale 1:

1. atualizar lasers anteriores emitidos pelo canhão (movê-los para cima)

   verificar colisões:

   se atingiu nave ou laser de nave → apagar os elementos e atualizar a pontuação

   se não há mais naves → jogador venceu

2. imprimir matriz do jogo

3. usuário escolhe lance sobre canhão (esquerda=tecla '`e`', direita= tecla '`d`', laser=tecla '`l`')

   realizar ação e ver se ocorre alguma colisão, apagando elementos e atualizando pontuação

   esquerda = pode colidir com nave ou laser de nave

   direita = pode colidir com nave ou laser de nave

   atirar = pode atingir nave ou laser de nave. Se não houver mais naves → jogador venceu

4. atualizar lasers anteriores emitidos pelas naves (movê-los para baixo)

   verificar colisões:

   se atingiu laser emitido pelo canhão → apagar os elementos e atualizar a pontuação

   se atingiu o canhão, final de jogo → jogador perdeu

5. para cada nave sem outra imediatamente abaixo decidir (pseudo-aleatoriamente) se emite laser

   verificar colisões:

   se atingiu laser emitido pelo canhão → apagar os elementos e atualizar a pontuação

   se atingiu o canhão, final de jogo → jogador perdeu

6. se rodada par, move-se as naves de acordo com regra de movimentação de naves

   verificar colisões:

   se foi atingida por laser do canhão → apagar os elementos e atualizar a pontuação

   se chegou na última linha, final de jogo → jogador perdeu

   se colidiu com o canhão → jogador perdeu

Note que a cada rodada, cada um dos itens anteriores é realizado **por completo** e, após cada item, deve ser atualizada a pontuação do jogo e verificado se o jogo termina, ou seja, se alguma dessas situações aconteceu:

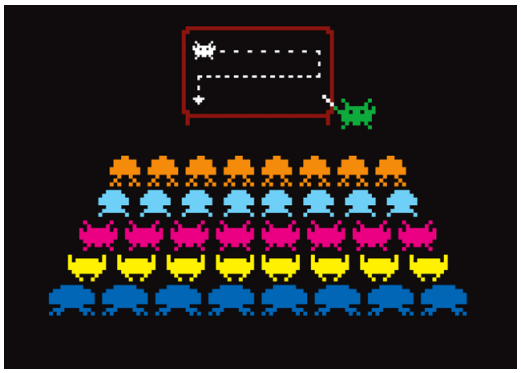
- O canhão foi atingido por alguma nave
- O canhão foi atingido pelo laser de alguma nave
- Alguma nave alcançou a linha inferior da área do jogo
- Todas as naves foram atingidas por algum laser do canhão

Nos três primeiros casos o jogador perde e o caracter * tem que ser impresso na posição onde o canhão do jogador estava. No último caso, o jogador ganha. Ao término da partida essa informação sobre vitória ou derrota deve ser impressa juntamente com a pontuação obtida pelo jogador. Após essas impressões, o programa deve finalizar a sua execução.

A pontuação do jogo vai sendo atualizada à medida que o jogador vai atingindo as naves ou os lasers das naves. É somado 1 ponto a cada laser atingido e 3 pontos a cada nave atingida.

Atenção: Neste EP considera-se que o canhão pode fazer movimentos cíclicos. Ou seja, considerando uma área de jogo de 20 linhas e 57 colunas, se em algum momento o canhão se encontra na linha 19, coluna 0, e move-se para a esquerda, então ele reaparece na linha 19, coluna 56. Já se ele se encontra na linha 19, coluna 56 e move-se para a direita, então ele reaparece na linha 19, coluna 0.

3 Preparando o canhão de laser



Neste EP, é necessário implementar funções que sigam as regras do jogo e os passos explicados na Seção 2. Você deve implementar e utilizar as funções que serão especificadas daqui para frente **sem modificar** os protótipos e os comentários e **sem modificar** qualquer parte do código que não seja solicitado aqui. É altamente recomendável que você implemente uma função de cada vez e teste antes de passar para a próxima. Recomendamos fortemente que os passos descritos abaixo sejam seguidos à risca. Comece pelo Passo 0 e, com exceção deste passo, não comece a realizar a tarefa do Passo n sem ter finalizado a tarefa do Passo $n-1$.

Passo 0: impressão da área do jogo

Acesse a página do EP2 e faça download do arquivo `ep2.py`[1]. Coloque esse arquivo no local onde você mantém os arquivos dos seus EPs, abra no editor de sua preferência e preencha o cabeçalho.

O arquivo `ep2.py` é o arquivo que contém o `main` e as diversas funções para controle do jogo. Você precisará editar o `ep2.py` para escrever as funções e, ao finalizar, submetê-lo na Graúna.

Rode o `ep2.py` da forma como você baixou da página do EP2. Você verá que ele já é capaz de pedir para o usuário a quantidade de naves alienígenas e imprimir na tela que o jogador venceu. OK, já é alguma coisa :-) .

Volte a abrir o `ep2.py` no seu editor de preferência e busque por:

```
Passo 0
```

Você chegará na função `imprimeMatriz`. Esta função imprime a área do jogo na tela. A área do jogo é representada por uma matriz de caracteres com dimensões 20×57. Cada elemento do jogo é representado por um caracter diferente:

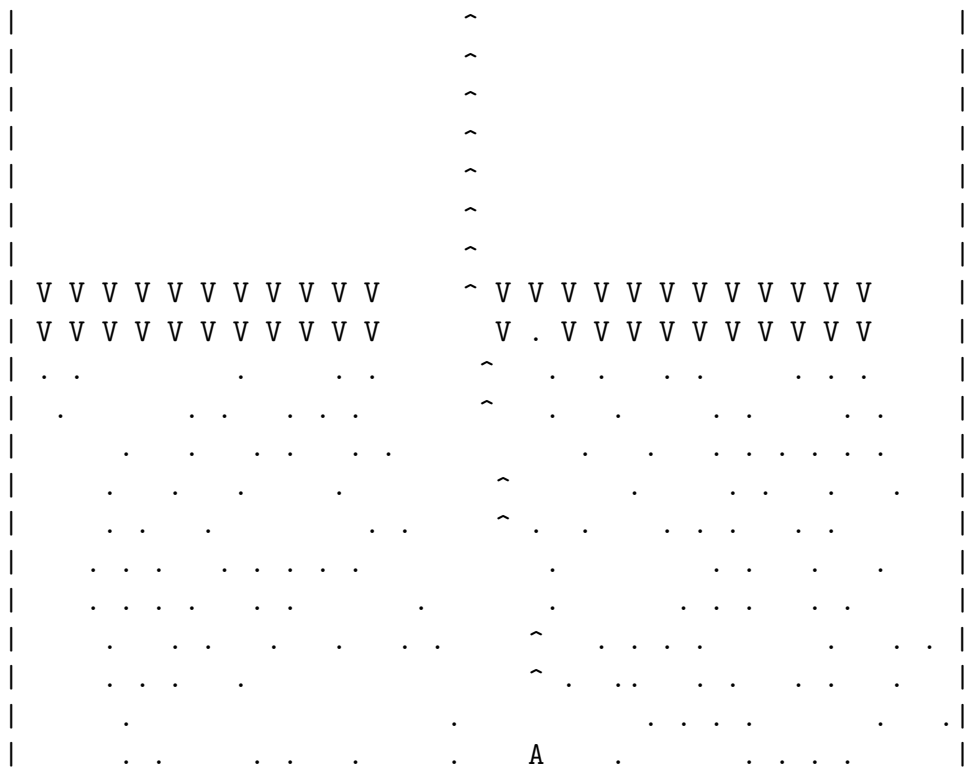
- `A`: canhão do jogador
- `V`: nave alienígena
- `^`: laser emitido pelo canhão do jogador

[1] maiores detalhes sobre os passos, as implementações e funções a serem escritas, podem ser encontradas diretamente no código ou nos comentários do `ep2.py`

- `.`: laser emitido pela nave alienígena
- `*`: explosão do canhão do jogador

Se há um espaço em branco em alguma posição da matriz, significa que nenhum dos elementos acima se encontra naquela posição.

O que a função **imprimeMatriz** deve fazer é imprimir a área do jogo exatamente como ela está naquele momento. A seguir está um exemplo de impressão feita pela função:



Note que, da forma como o arquivo `ep2.py` está, a função `imprimeMatriz` já tem algumas linhas escritas. Porém, a borda da área do jogo não está sendo impressa. A sua tarefa nesta função é modificar o código para que essa borda seja impressa utilizando o caracter `|`. Note que você não deve aumentar o tamanho da matriz internamente no jogo. A única coisa que precisa ser feita é imprimir uma borda à esquerda e à direita da matriz, sem aumentar o espaço que está sendo usado para armazenar o seu conteúdo.

Faça essa mudança e não esqueça de testar a função invocando-a com o interpretador python que você está usando.

Se você conseguiu chegar até aqui, você terminou o Passo 0!

Obs.: Os caracteres que representam os elementos do jogo já estão definidos nas variáveis `CANHAO`, `NAVE`, `LASER_CANHAO`, `LASER_NAVE` e `EXPLOSAO` no início do `ep2.py`. É altamente recomendável usar essas variáveis ao invés dos valores dos caracteres diretamente no seu código para aumentar a legibilidade do mesmo. De forma similar, a matriz do jogo está

com a quantidade de linhas definida pelo valor `LINHA_MAXIMA`+1 e a quantidade de colunas pelo valor `COLUNA_MAXIMA`+1 e ambas as variáveis estão definidas também no início do código.

Passo 1: criação do canhão e das naves

Quando você testou a `imprimeMatriz` no passo anterior, ela imprimiu uma matriz vazia, o que era de se esperar já que os elementos do jogo ainda não foram criados. Para criar o canhão e as naves, busque no `ep2.py` por:

```
Passo 1
```

Você chegará na função `criaElementos`. Esta função modifica a matriz do jogo, recebida como parâmetro, adicionando as naves e o canhão de acordo com as posições iniciais explicadas na Seção 2. A quantidade de naves que devem ser criadas também é passada como parâmetro na função.

Essa função não está escrita, escreva-a, e faça testes invocando-a com o interpretador python que você está usando. Para verificar se deu tudo certo, recomenda-se testar esta função seguida da invocação da função `imprimeMatriz`.

Neste ponto do EP você já tem o conteúdo da área do jogo carregado em uma matriz na memória. Agora é necessário colocar tudo para se movimentar!

Passo 2: movimentação do canhão e das naves

Busque no `ep2.py` por:

```
Passo 2
```

Você encontrará duas funções relacionadas a este passo. A função `moveCanhao` e a função `moveNaves`. Note que apesar da primeira ser chamada apenas quando o usuário digitar o caracter referente à movimentação e a segunda ser chamada apenas nas rodadas pares, o `input` para a primeira e a verificação da rodada atual pela segunda, não devem ser feitas aqui! Estas funções simplesmente movem os elementos sem fazer essas verificações. Adiante você entenderá que essas verificações deverão ser feitas na função do Passo 5.

As duas funções recebem como parâmetros um número inteiro e a matriz do jogo. O número inteiro representa a direção para onde a movimentação deve ser realizada. Conforme explicado na Seção 2, as naves só se movem para a esquerda, para a direita e para baixo e o canhão só se move para a esquerda e para a direita, sendo que o canhão tem movimento cíclico. A relação entre os valores inteiros e a direção está definida nas variáveis `ESQUERDA`, `DIREITA` e `BAIXO` no início do código:

- `ESQUERDA = -1`

- `DIREITA = 1`
- `BAIXO = -2`

Cada função deve verificar possíveis colisões depois que os movimentos forem realizados.

Diferente das funções dos dois passos anteriores, essas funções retornam valores. A função `moveCanhao` retorna um booleano que vale `True` se o canhão foi destruído e `False` caso contrário. Já a função `moveNaves` retorna uma lista com 3 valores:

```
[bool, int, int]
```

O primeiro valor é um booleano que vale `True` se o canhão foi destruído e `False` caso contrário. O segundo valor é um inteiro que informa se algum limite da matriz foi alcançado (canto esquerdo, canto direito, linha inferior ou nenhum dos anteriores). O terceiro valor é um inteiro que informa a quantidade de naves destruídas por lasers após a movimentação de todas as naves.

A relação entre os valores inteiros e os limites alcançados está definida nas variáveis `ATINGIU_DIREITA`, `ATINGIU_ESQUERDA` e `ATINGIU_EMBAIXO` no início do código:

- `ATINGIU_ESQUERDA = -1`
- `ATINGIU_DIREITA = 1`
- `ATINGIU_EMBAIXO = -2`

Esses valores de retorno farão sentido quando você for escrever a função do Passo 5 para implementar corretamente a movimentação das naves.

Teste as duas funções criadas tanto em casos em que deveriam haver colisões quanto em casos em que não deveriam. Nos seus testes, sempre depois de chamar a função, imprima os valores retornados e invoque a função `imprimeMatriz` para confirmar que a movimentação realizada modificou a matriz como deveria.

Nesse ponto do EP, os elementos já conseguem se movimentar. Agora eles precisam emitir os lasers.

Passo 3: emissão de lasers

Busque no `ep2.py` por:

```
Passo 3
```

Você encontrará duas funções relacionadas a este passo. A função `emiteLaserCanhao` e a função `emiteLasersNaves`. Note que apesar da primeira ser chamada apenas quando o usuário digitar o caracter referente à emissão de um laser, o `input` não deve ser feito aqui! Adiante você entenderá que essa verificação deverá ser feita na função do Passo 5.

As duas funções recebem como parâmetro apenas a matriz do jogo. Elas devem varrer a matriz do jogo e a cada vez que encontrarem o elemento referente ao seu objetivo (o canhão, no caso da função `emiteLaserCanhao` e alguma nave, no caso da função `emiteLasersNaves`), devem colocar na matriz o caracter do laser imediatamente acima, no caso do canhão, ou imediatamente abaixo, no caso das naves. Assim como no caso das funções do passo anterior, todas as possíveis colisões explicadas na Seção 2 precisam ser verificadas.

Note que a função que emite os lasers das naves deve sortear valores para definir se uma nave deve ou não emitir lasers. Para isso deverá ser invocada, para cada nave candidata a emitir lasers, a função `random.randint`[2]. Esta função recebe dois números inteiros como parâmetro e retorna um valor inteiro entre esses dois números, inclusive. Utilize a função `random.randint` com parâmetros $(0, 1)$ para sortear 0's e 1's. Se o valor sorteado for 1, faça a nave sendo verificada no momento emitir um laser.

Com a emissão de novos lasers, elementos do jogo podem ter sido atingidos e os retornos dessas funções são listas que retornam essas informações.

A função `emiteLaserCanhao` retorna uma lista com dois valores:

```
[int, int]
```

O primeiro valor é um inteiro que informa a quantidade de naves destruídas pelo laser que acabou de ser emitido. O segundo valor é um inteiro que informa a quantidade de lasers destruídos pelo laser que acabou de ser emitido.

A função `emiteLasersNaves` retorna uma lista com dois valores:

```
[bool, int]
```

O primeiro valor é um booleano que vale `True` se o canhão foi destruído com algum laser que acabou de ser emitido e `False` caso contrário. O segundo valor é um inteiro que informa a quantidade de lasers destruídos pelos lasers que acabaram de ser emitidos.

Assim como no passo anterior, não esqueça de testar essas funções e de verificar, imprimindo os valores retornados e chamando a função `imprimeMatriz`, se o resultado está correto.

Nesse ponto do EP, tanto o canhão quanto as naves já conseguem se movimentar e emitir lasers. Falta fazer com que os lasers se movimentem.

[2] Assim como feito no EP1, o gerador de números aleatórios deve ser inicializado com alguma semente. Isso já está sendo feito no `main` com a função `random.seed`.

Passo 4: movimentação dos lasers

Busque no `ep2.py` por:

```
Passo 4
```

Você encontrará duas funções relacionadas a este passo. A função `moveLasersCanhao` e a função `moveLasersNaves`. Essas funções tem o papel de mover todos os lasers que estão na matriz do jogo. Os lasers do canhão sempre se movem 1 posição para cima e os lasers das naves sempre se movem 1 posição para baixo.

As duas funções recebem como parâmetro apenas a matriz do jogo. Elas devem varrer a matriz do jogo e a cada vez que encontrarem o elemento referente ao seu objetivo (algum laser emitido pelo canhão, no caso da função `moveLasersCanhao` e algum laser emitido por alguma nave, no caso da função `movLasersNaves`), devem mover os lasers corretamente. Assim como no caso das funções dos dois passos anteriores, todas as possíveis colisões explicadas na Seção 2 precisam ser verificadas.

Com a movimentação dos lasers, elementos do jogo podem ter sido atingidos e os retornos dessas funções são listas que retornam essas informações da mesma forma que as funções do passo anterior.

A função `moveLasersCanhao` retorna uma lista com dois valores:

```
[int, int]
```

O primeiro valor é um inteiro que informa a quantidade de naves destruídas pelos lasers que se movimentaram. O segundo valor é um inteiro que informa a quantidade de lasers destruídos pelos lasers que se movimentaram.

A função `moveLasersNaves` retorna uma lista com dois valores:

```
[bool, int]
```

O primeiro valor é um booleano que vale `True` se o canhão foi destruído por algum laser que foi movimentado e `False` caso contrário. O segundo valor é um inteiro que informa a quantidade de lasers destruídos pelos lasers que se movimentaram.

Assim como nos dois passos anteriores, não esqueça de testar essas funções e de verificar, imprimindo os valores retornados e chamando a função `imprimeMatriz`, se o resultado está correto.

Nesse ponto do EP, todas as funções de controle dos elementos do jogo estão prontas. Falta unir tudo no loop principal do jogo.

Passo 5: loop principal do jogo

Conforme explicado na Seção 2, o jogo é um conjunto de rodadas e a cada rodada são realizados 6 conjuntos de ações. Neste passo 5 você deve implementar isso.

Busque no `ep2.py` por:

```
Passo 5
```

Você encontrará uma função chamada `joga` parcialmente escrita. Ela já cria uma matriz vazia, define os valores iniciais de algumas variáveis e executa um laço que é o laço principal do jogo que deve ser modificado por você (provavelmente alguma outra ação também terá que ser feita fora deste laço).

Para completar a função `joga` você deve utilizar as funções que foram criadas nos passos anteriores. A função recebe como parâmetro apenas a quantidade de naves alienígenas e retorna uma lista com dois valores:

```
[bool, int]
```

Onde o primeiro valor vale True se o jogador venceu ou False se o jogador perdeu e o segundo valor armazena a quantidade de pontos que o jogador fez. A pontuação é atualizada no decorrer do jogo da seguinte forma:

- +3 pontos se o jogador consegue acertar 1 laser em alguma nave
- +1 ponto se o jogador consegue acertar 1 laser em algum laser de alguma nave

Para manter uma boa legibilidade no código, ao contabilizar a pontuação do jogo na função `joga`, utilize as variáveis `PONTOS_ACERTOU_LASER` e `PONTOS_ACERTOU_NAVE` definidas no início do código.

É nesta função também que o jogador deverá informar a sua ação. Faça isso solicitando entrada de dados da seguinte forma:

```
input("'e' para esquerda, 'd' para direita e 'l' para emitir laser: ")
```

Não altere a string usada no `input` acima. Utilize exatamente da forma como apresentada. Como a própria mensagem do `input` sugere, as ações do jogador serão feitas com as teclas '`e`', '`d`' e '`l`' seguidas de ENTER.

Por fim, a função `joga` precisa ser modificada para imprimir o conteúdo final da matriz quando a partida terminar.

Neste ponto do EP, testes realizados com a função `joga` já estarão testando o EP como um todo, então não é necessário testar a função separadamente. Teste rodando o `main` do seu código. Entretanto, vá realizando testes após implementar cada um dos 6 conjuntos de ações que devem ser realizados na função `joga`.

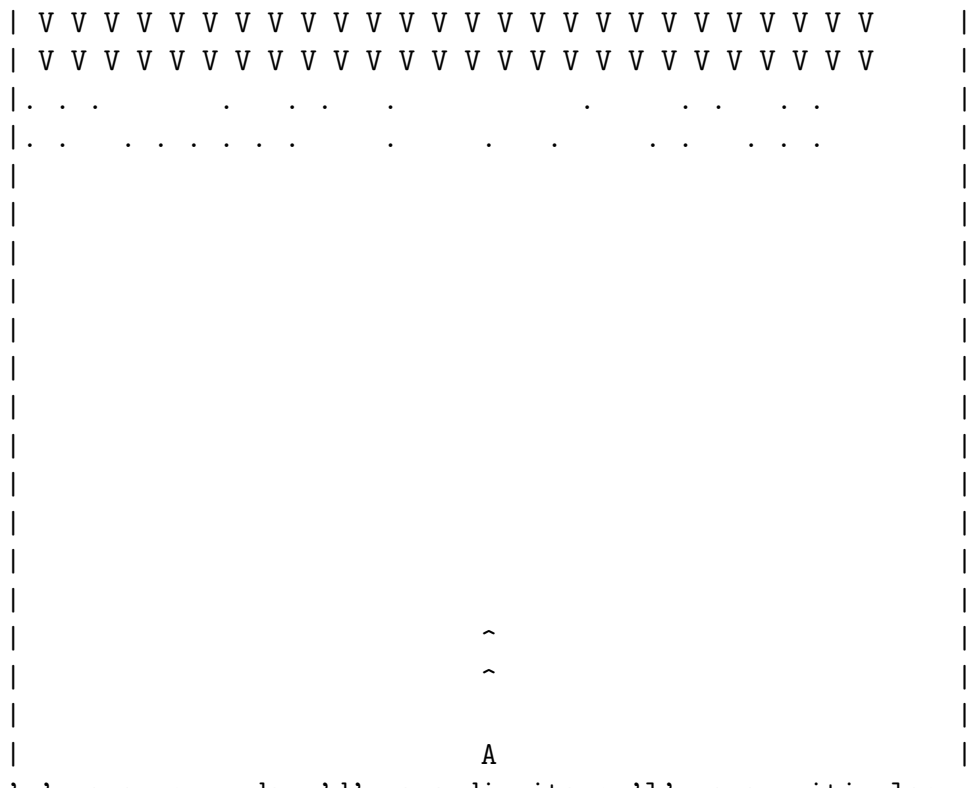
4 Entrega do EP

O arquivo `ep2.py` deverá ser submetido na área especificada na Graúna.

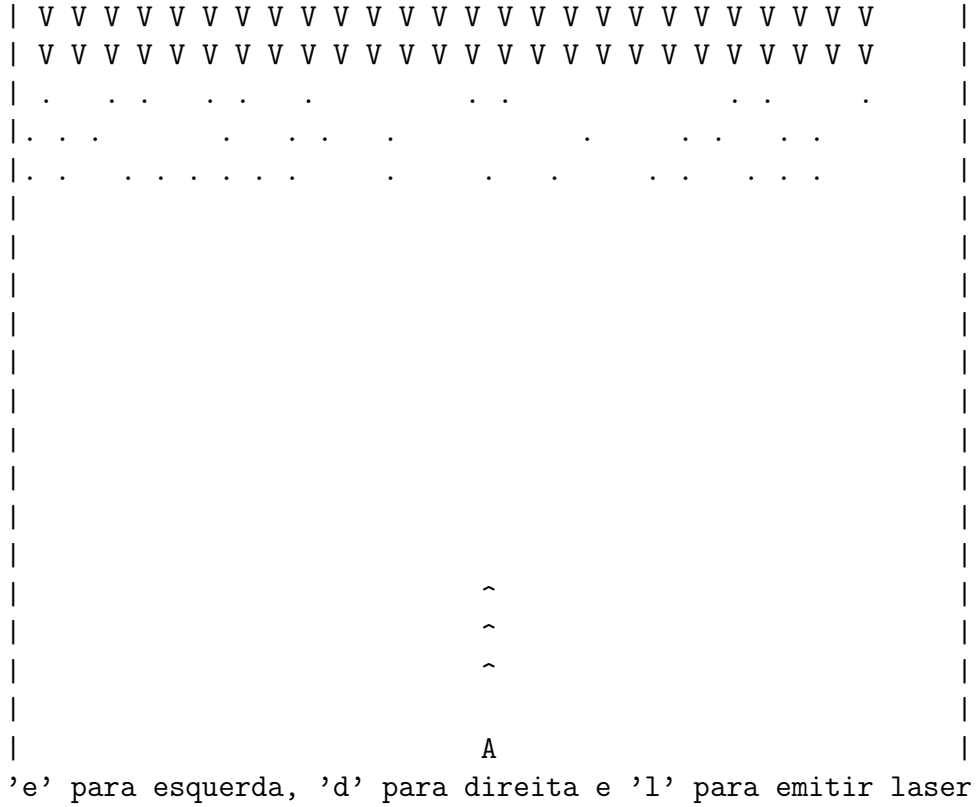
Lembre que **na implementação deste EP você só poderá usar recursos do python apresentados em sala de aula pelo seu professor. Leia as instruções gerais para entrega de EPs em** https://www.ime.usp.br/~mac2166/infoepsPy/

Apêndice – Uma partida com a saída esperada do EP

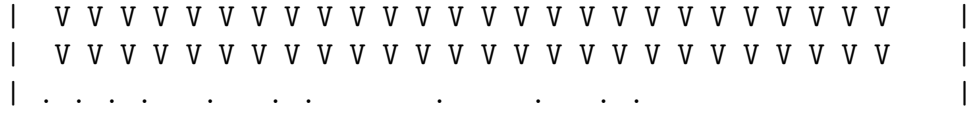
```
Digite o numero de naves (inteiro maior que 1 e menor que 53): 52
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                            A                            |
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|. .   . . . . . .     .     .   .     . .   . . .        |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                            ^                            |
|                                                         |
|                            A                            |
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

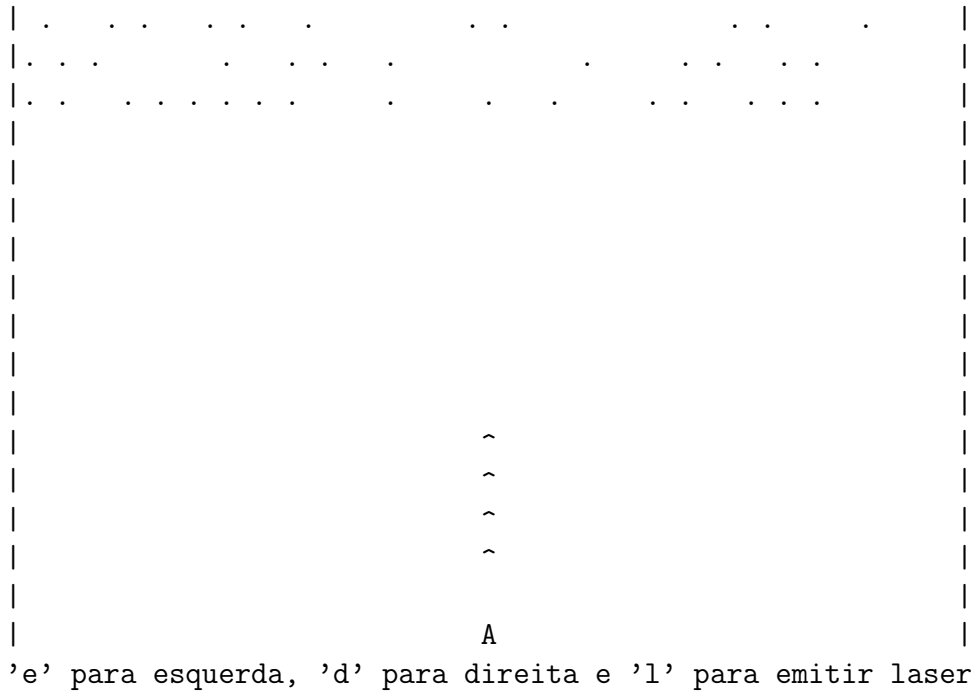


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

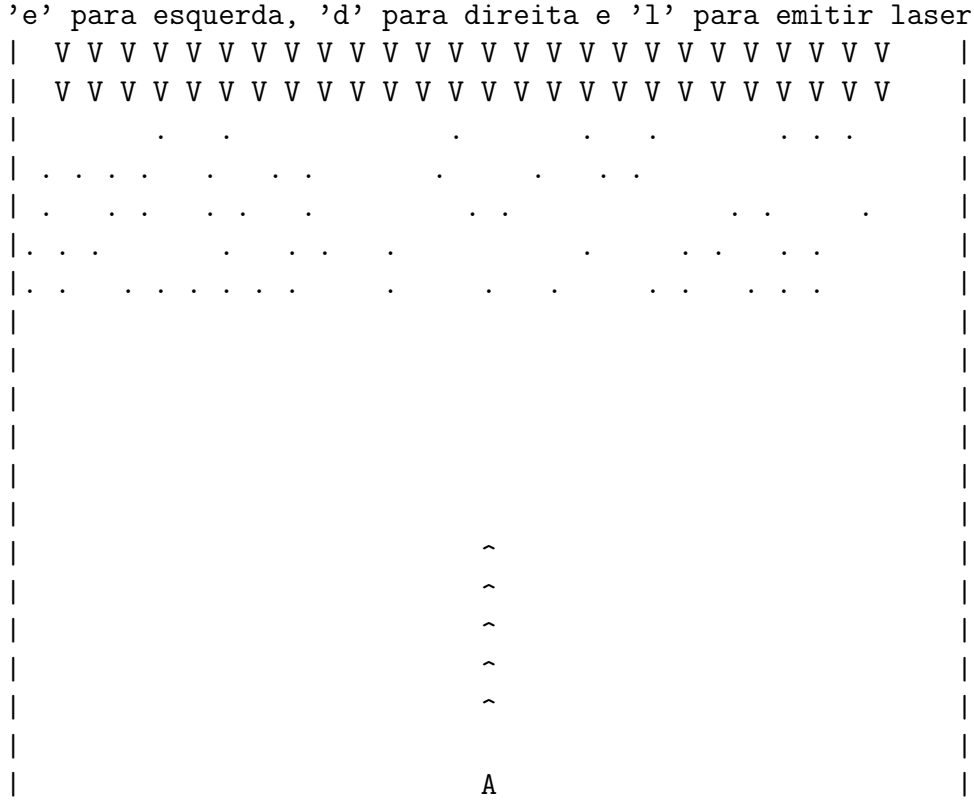


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

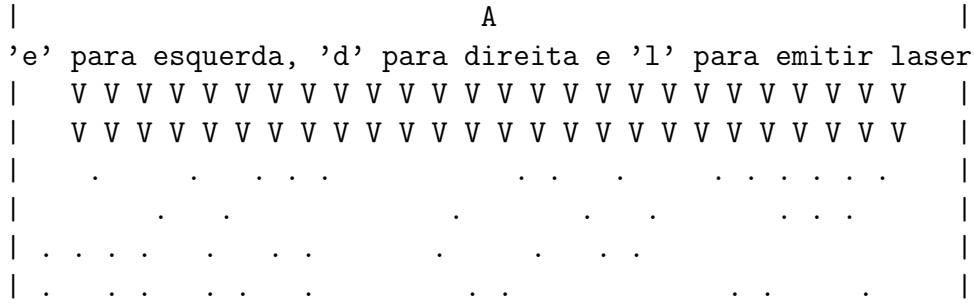


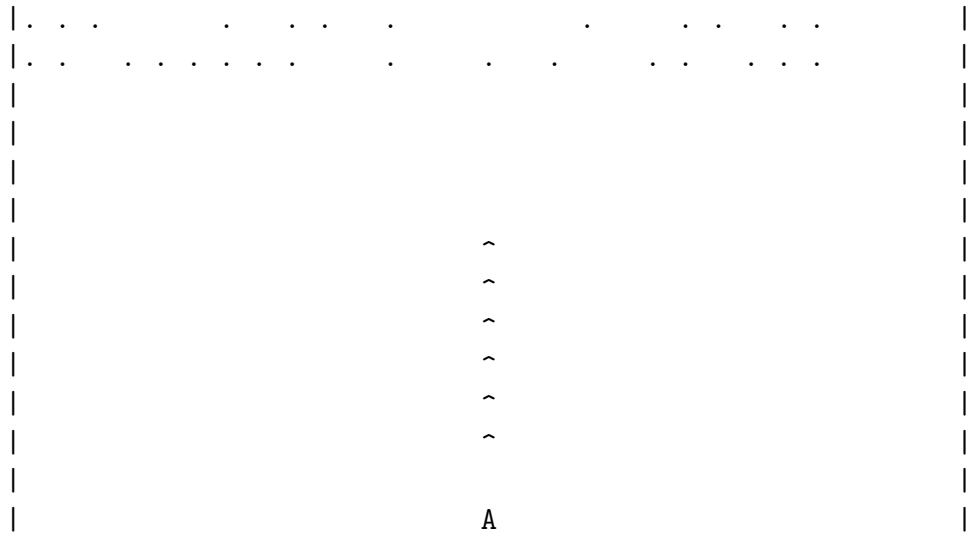


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

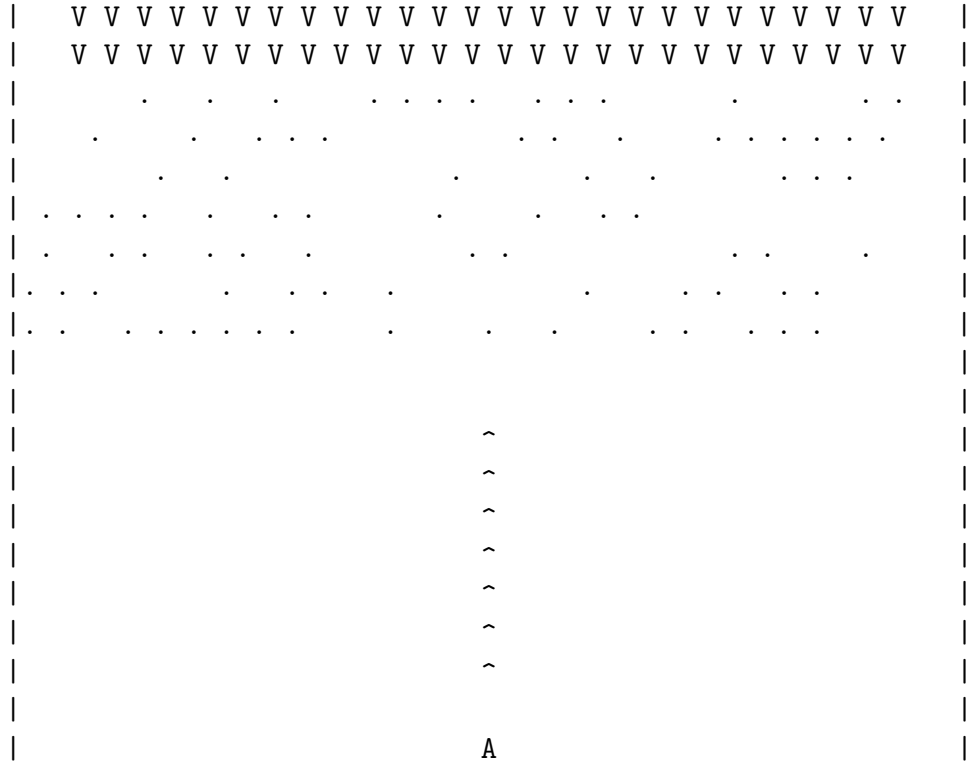


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

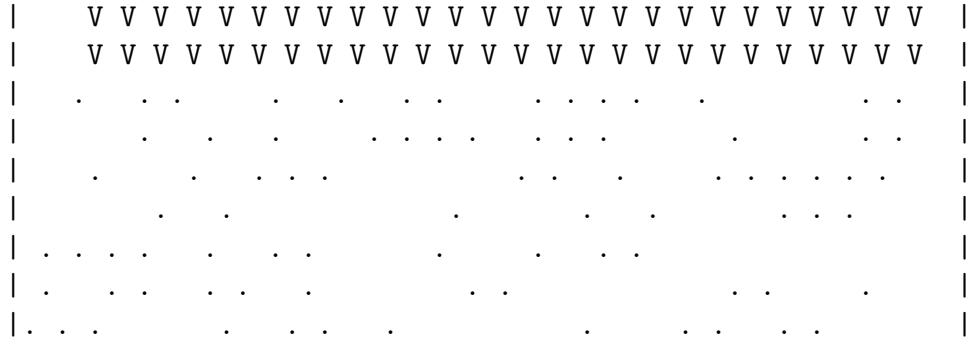


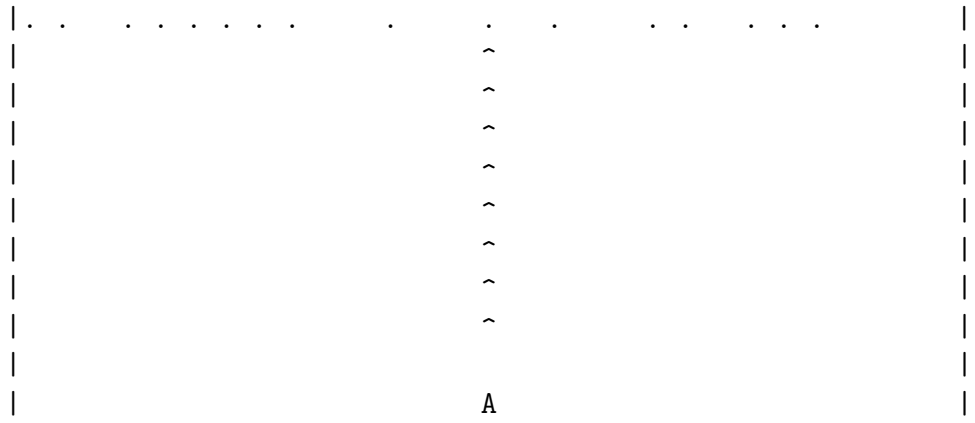


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

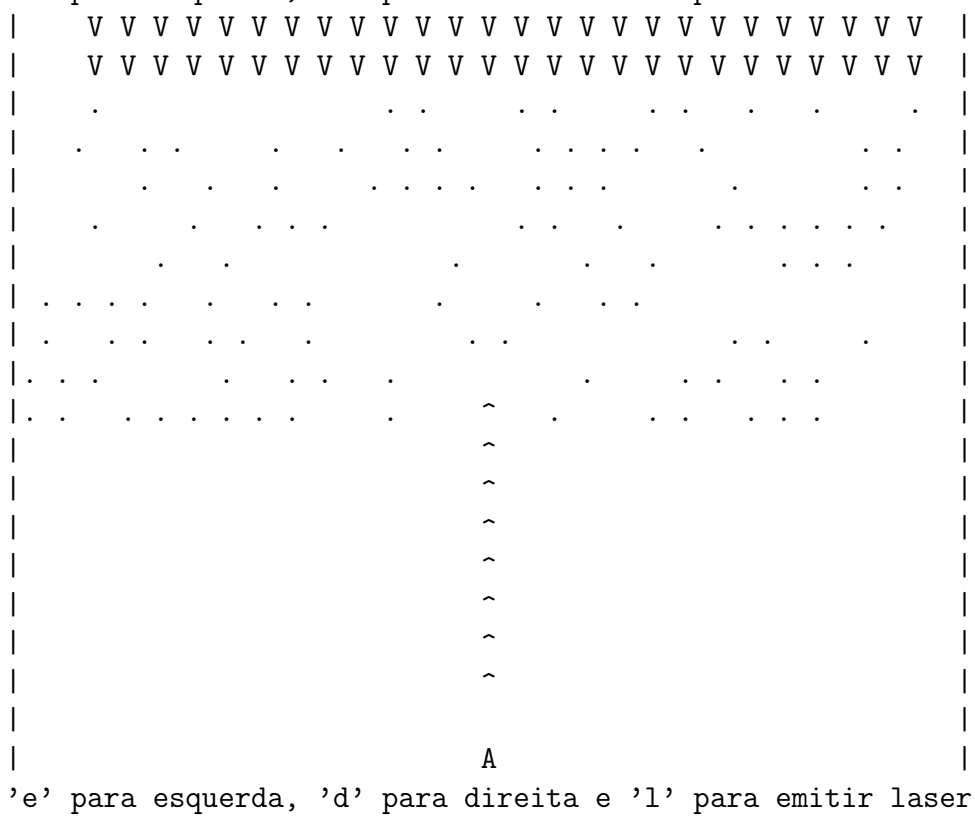


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

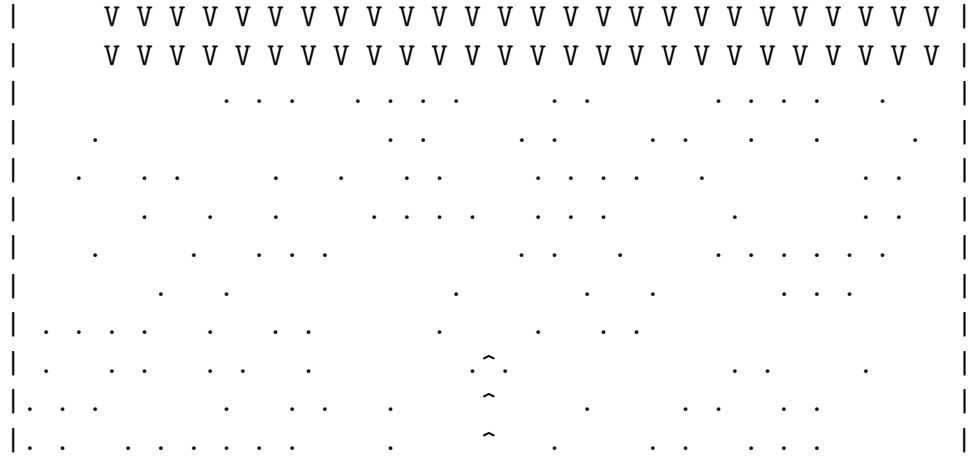


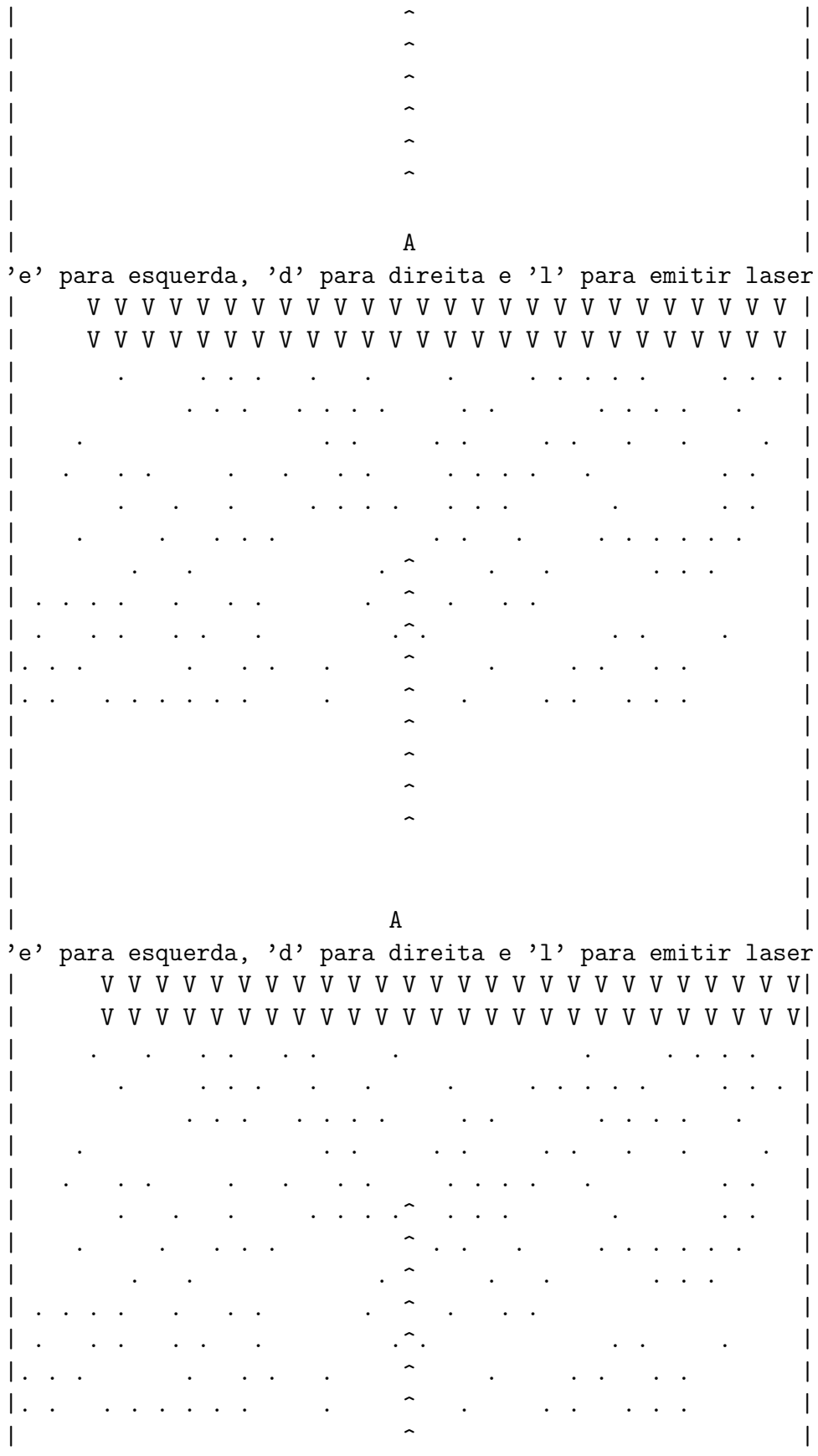


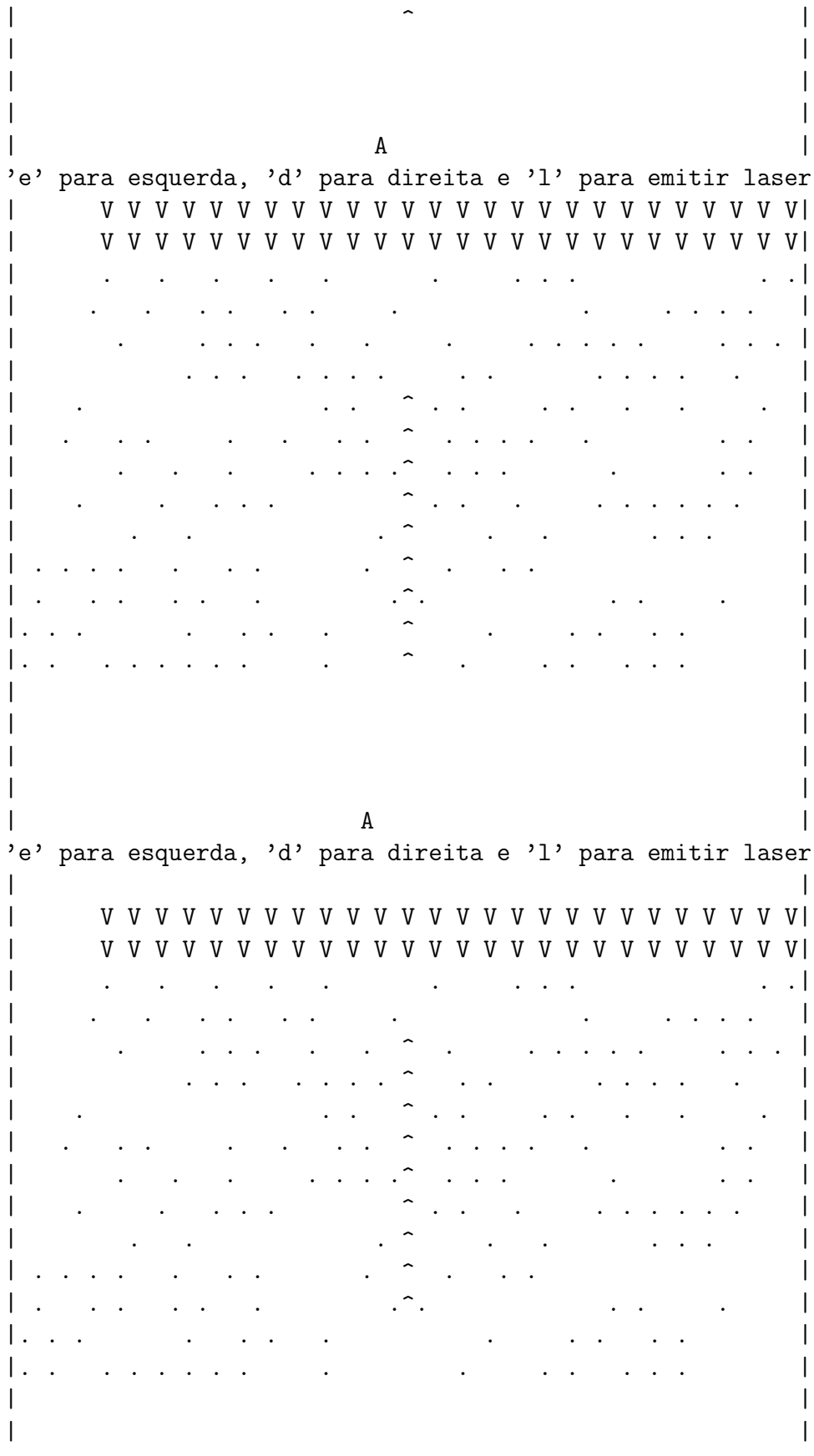
```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

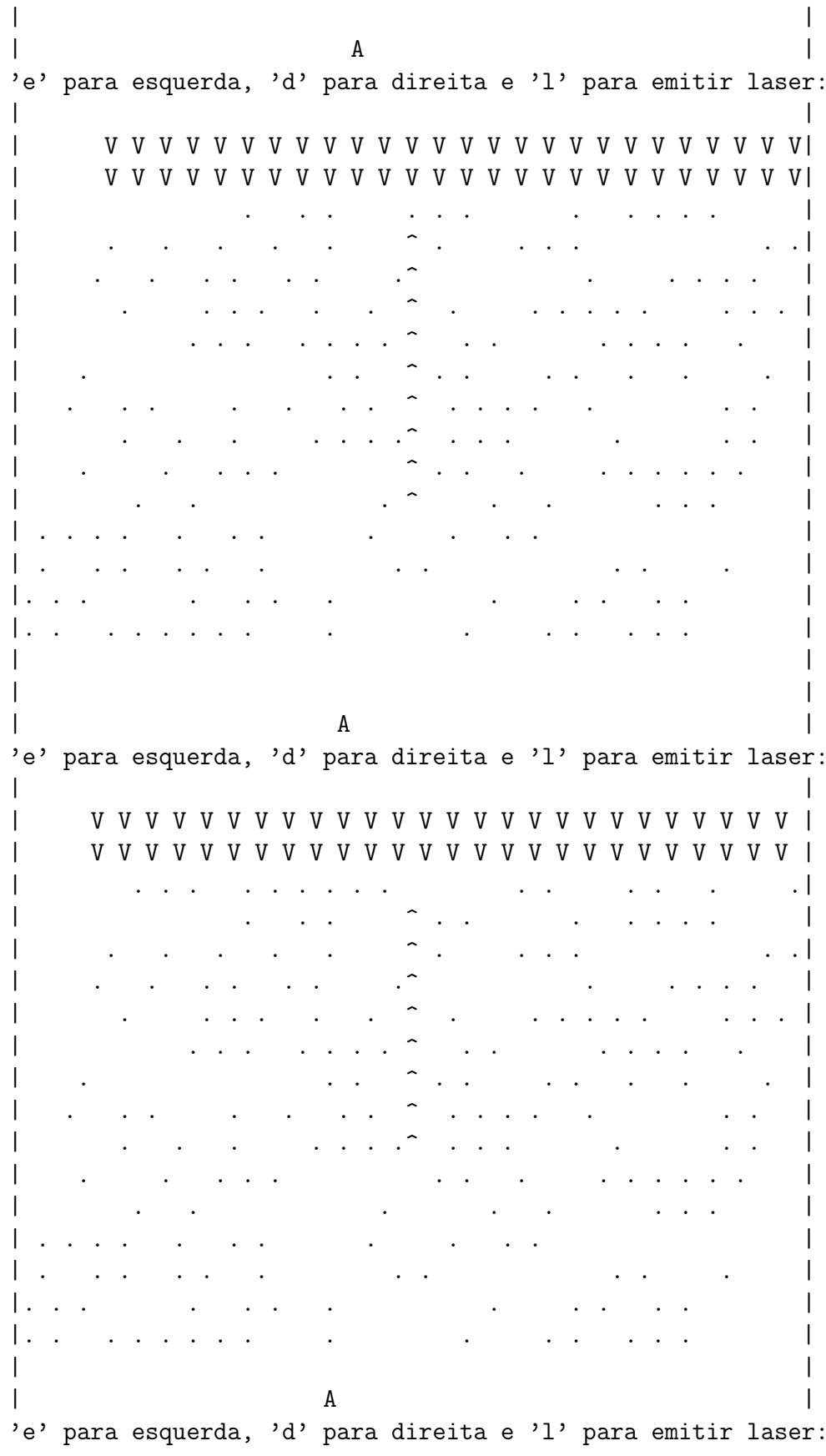


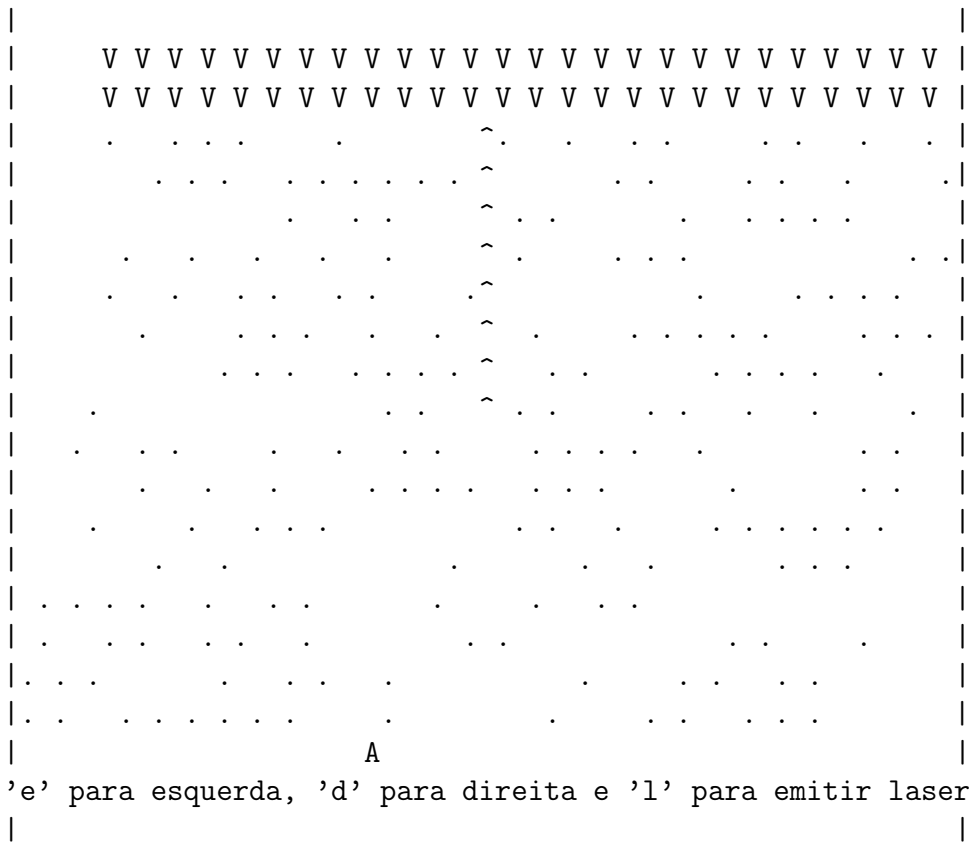
```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```



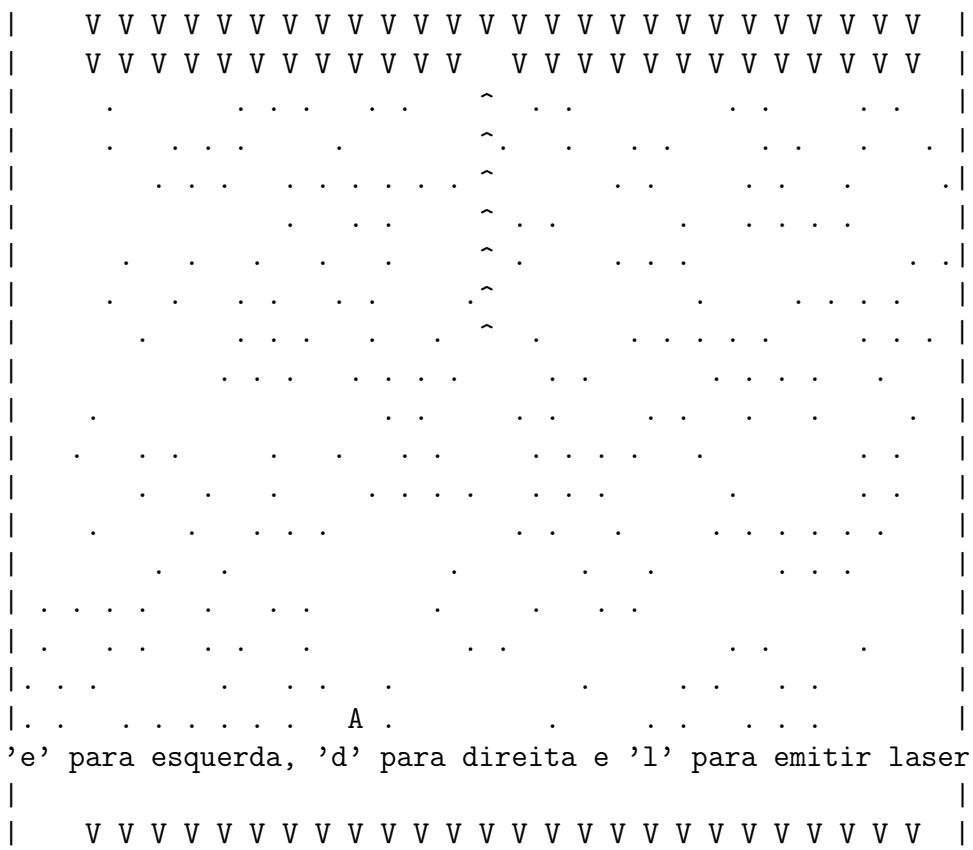

'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e


A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e

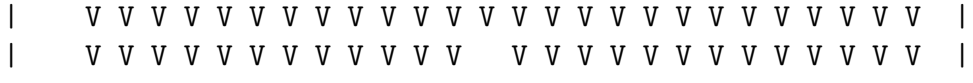

A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e

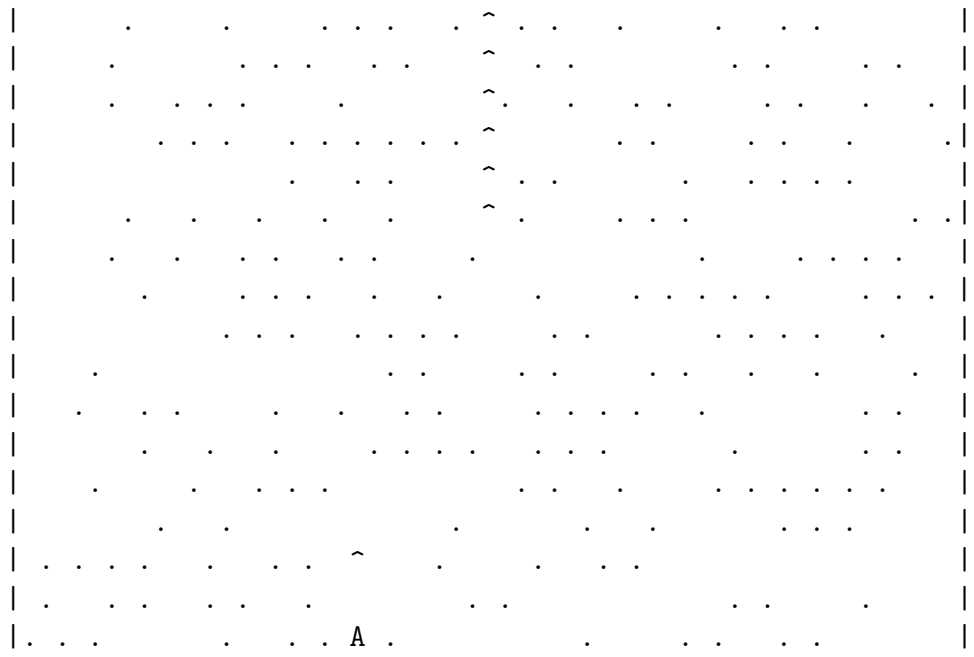


```
                    A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
```

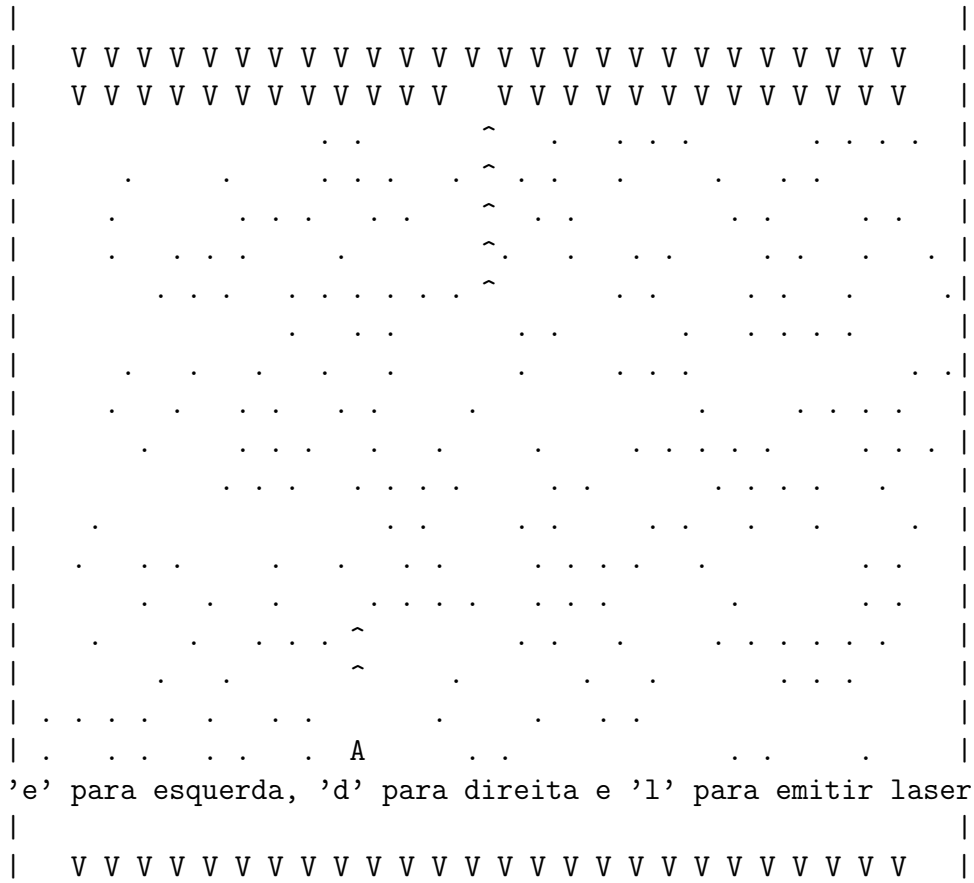


```
                  A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

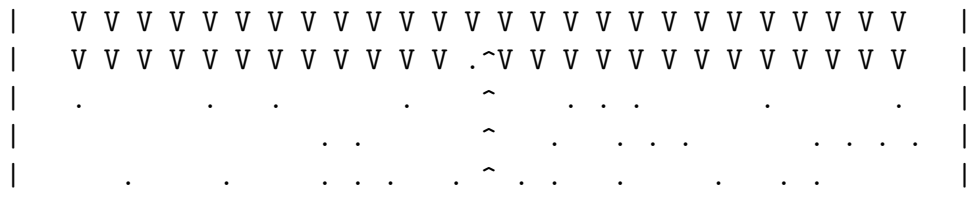


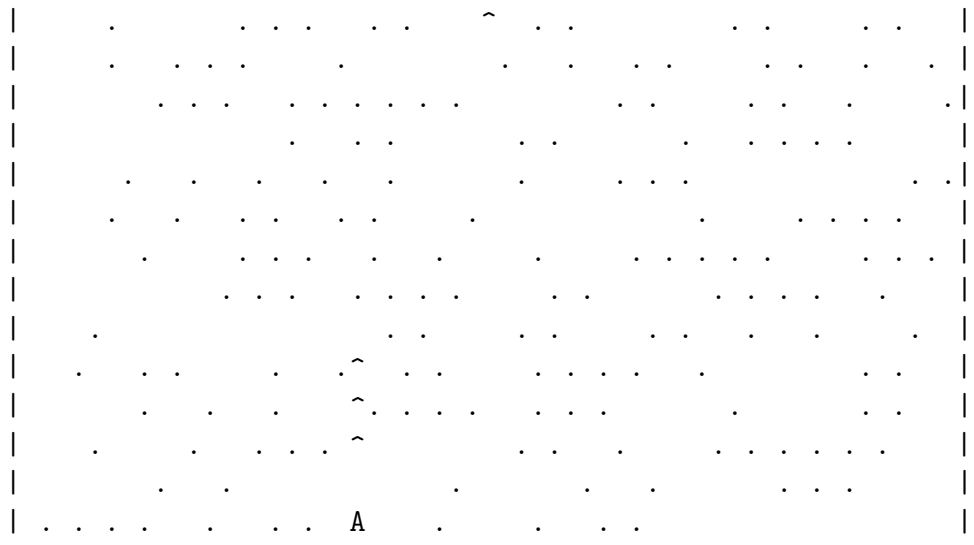


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

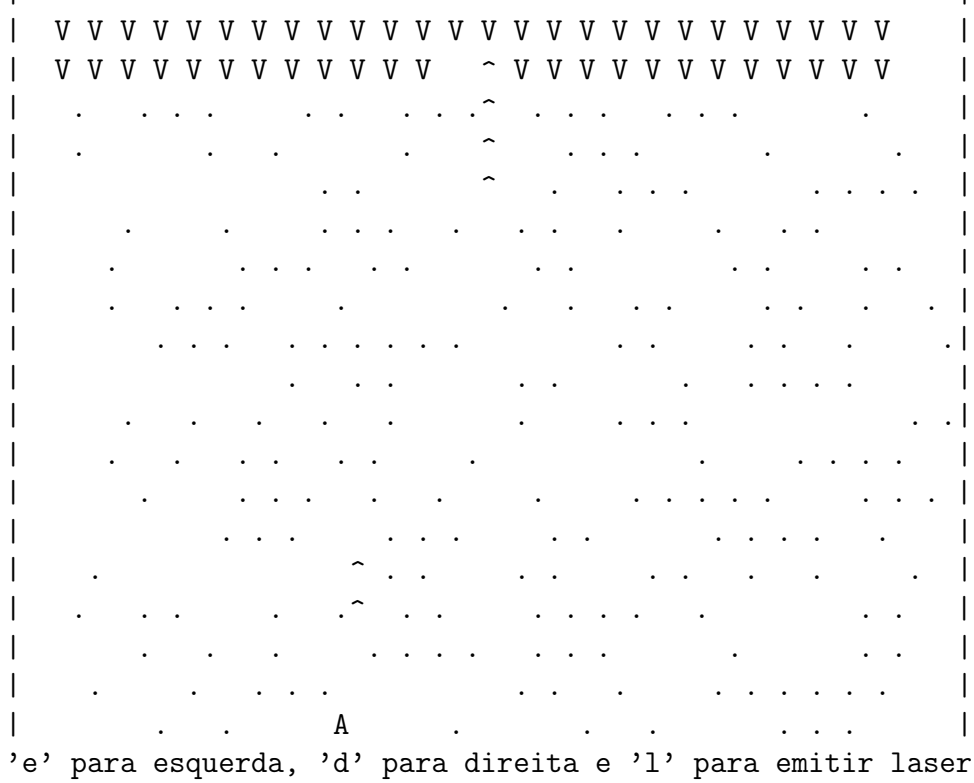


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

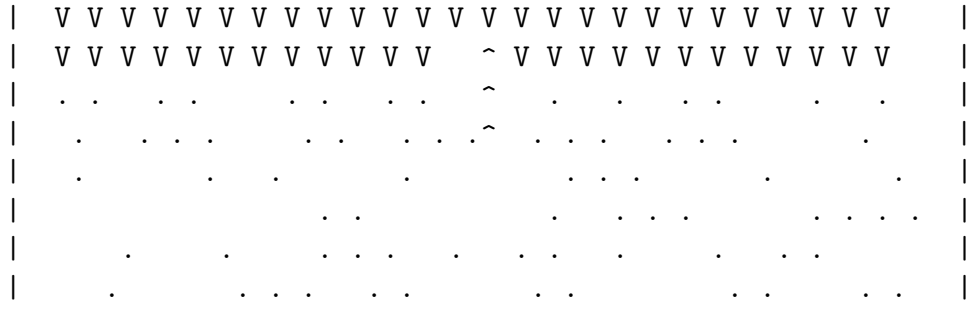


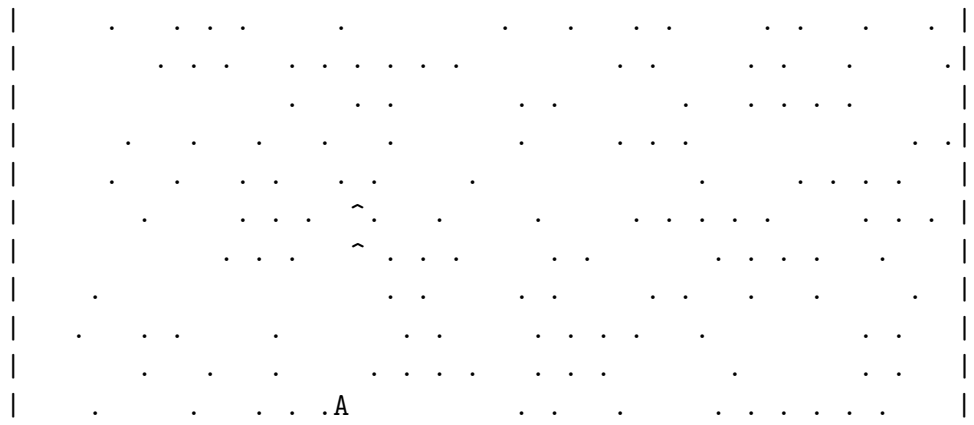


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
```

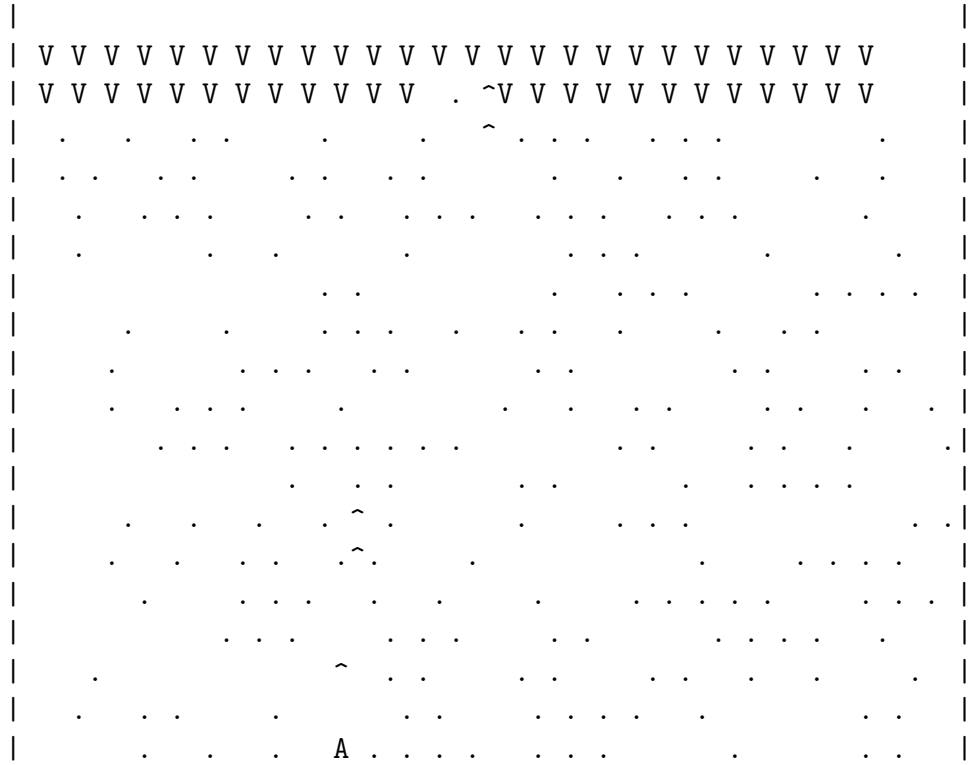


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

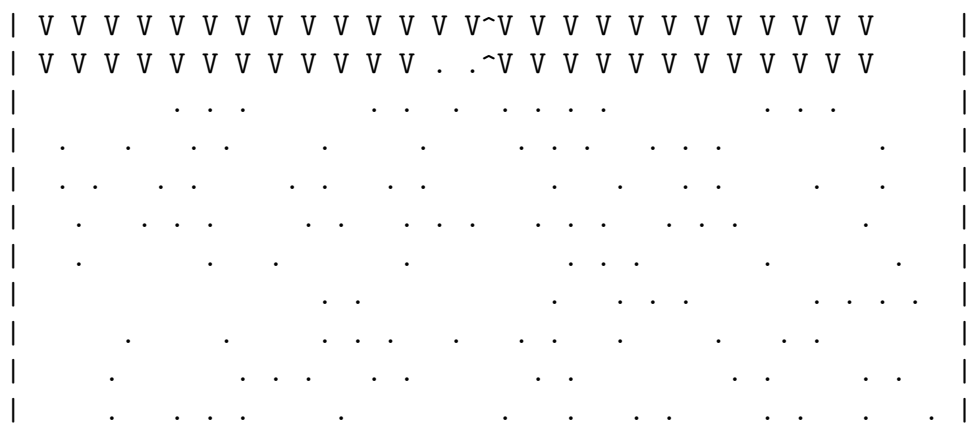


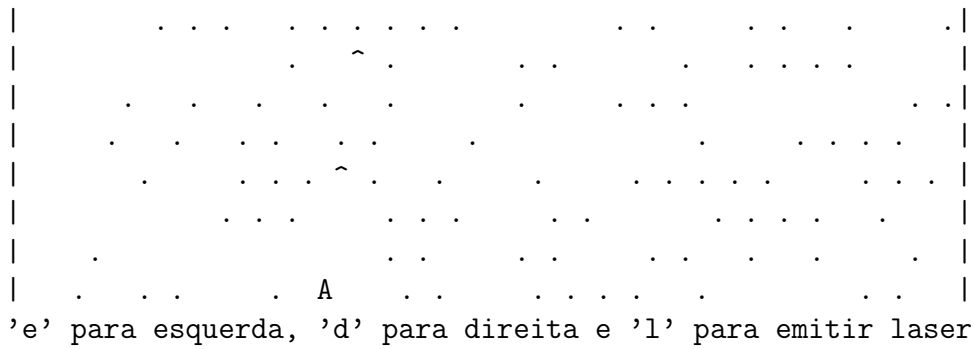


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

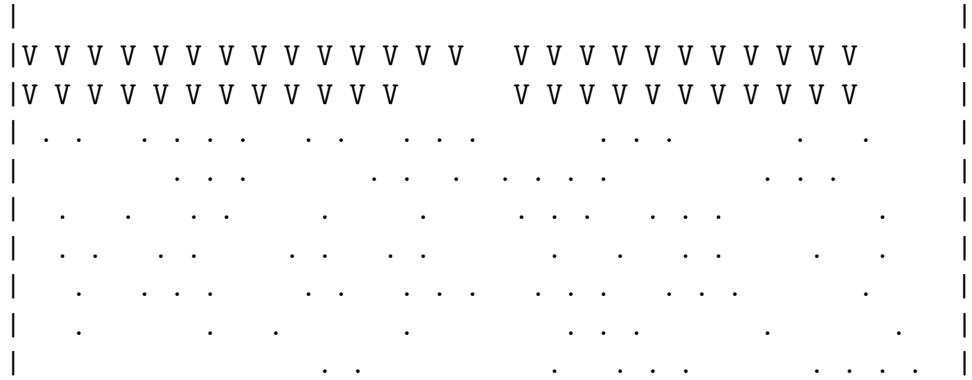


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
```

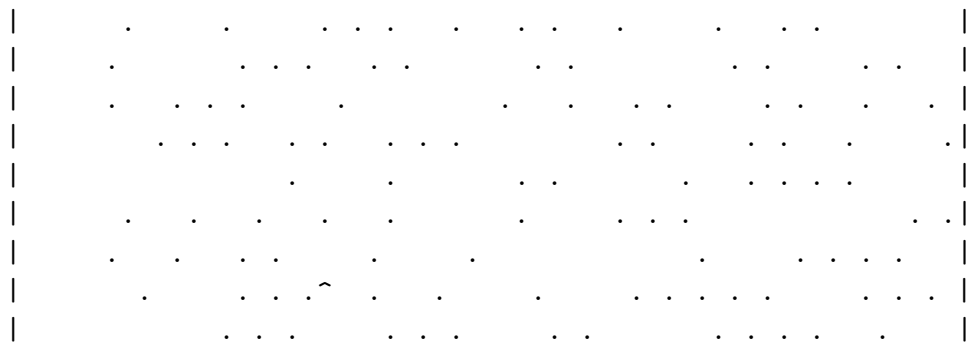


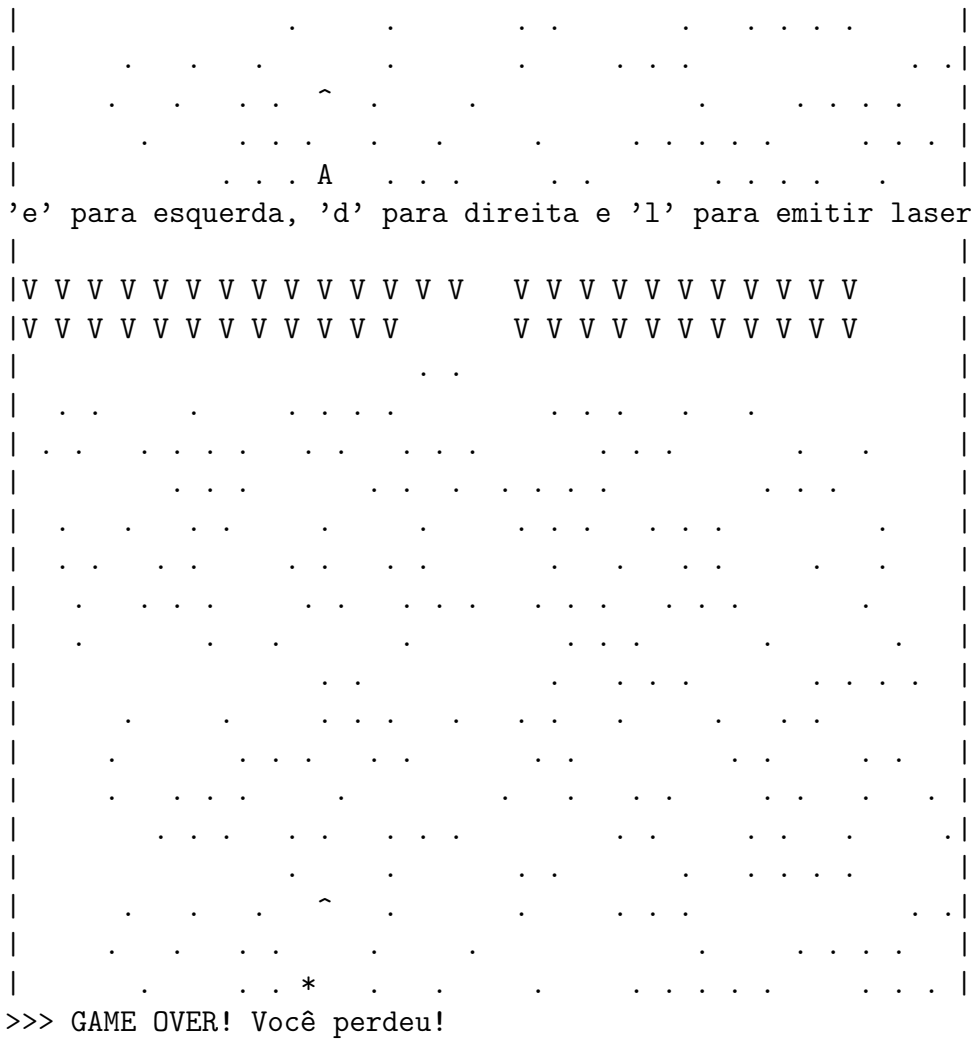


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```



```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```




'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
>>> GAME OVER! Você perdeu!
>>> Pontuação: 24