Departamento de Ciência da Computação – IME-USP

MAC2166 Introdução à Computação
Escola Politécnica - Turmas 13 e 14 (Web)
Primeiro Semestre de 2018

Segundo Exercício-Programa (EP2)

Data máxima para entrega: 19/5/2018

versão 1.0

20 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* (ou do tipo *fliperama*) que revolucionou a indústria de jogos eletrônicos e que se tornaria um ícone da cultura popular alguns anos depois: *Space Invaders* ("invasores do espaço"). 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 seu canhão (por simplicidade implementaremos apenas um 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 naves alienígenas começam na parte superior da tela, se movimentam na horizontal ou descem uma linha quando uma delas atinge uma das laterais 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/vetores e funções

em **Python** ou em **C**. A cada iteração do jogo, o usuário poderá mover o canhão (emissor de *laser*) 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 programa que resolve um problema (no caso, função a função) e aprender a seguir padrões pré-definidos de 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 da linguagem (C ou Python) que já foi apresentado em alguma aula remota. Além disso, sua nota dependerá de seu programa conseguir imprimir *ipsis-litteris* como nos casos-de-teste (após estar tudo funcionando faça os "ajustes finos" para imprimir como o esperado).

As únicas informações passadas ao programa pelo usuário no início da partida são: as dimensões do **tabuleiro/espaço** (colunas e linhas entre `2 x 2` e `19 x 53`); e a quantidade de naves alienígenas (um número inteiro entre 1 e menor que 53). O canhão do jogador ocupará a linha inferior da área do jogo e estará inicialmente na coluna do meio (se número de colunas for par, o primeiro inteiro "à esquerda"). As naves sempre começarão a partida alinhadas ao canto superior esquerdo da área do jogo, ocupando 2 linhas (se total de naves for ímpar, a primeira linha terá uma nave a mais), sendo que na horizontal elas terão 1 espaço em branco entre cada par. Na tabela abaixo são ilustradas as configurações iniciais das naves (nos cantos superiores esquerdos) para 2, 3, 6 e 7 naves. Cada nave é representada pelo caractere `V`.

2 naves	3 naves	6 naves	7 naves
V	V V	V V V	V V V V
V	V	V V V	V V V

As naves podem se mover para a *direita*, para *esquerda* e *baixo*, sendo que pode realizar seguidos movimentos horizontais mas um único vertical para baixo, O movimento para baixo ocorre quando uma das naves alcança a primeira ou última coluna do *tabuleiro/espaço*, nesse momento todas as naves movem-se uma linha para baixo e invertem a direção de movimentação (se faziam movimento para a esquerda, passam a moverem-se para a direita).

Muita **atenção** à regra de atualização do *estado do jogo*, resumida nos 6 passos abaixo. A regra deixa de apresentar a primeira posição de cada disparo do canhão do usuário, mas isso é necessário para simplificar a implementação do jogo. O estado do jogo será representada por uma **matriz de caracteres**.

Cada iteração (também chamada no restante deste enunciado de **rodada**) `i` do jogo prossegue da seguinte forma, considerando que no início, que `i=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, nesse caso

   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 a rodada for par (ou seja, `i`), 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 depois 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 caractere * 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 um *tabuleiro/espaço* com 19 linhas e 53 colunas, se em algum momento o canhão se encontra na linha 18 (primeira linha tem índice 0), coluna 0, e move-se para a esquerda, então ele reaparece na linha 18, coluna 52. De forma análoga, se ele se encontra na linha 18, coluna 52 e move-se para a direita, então ele reaparece na linha 18, 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 (e.g., implemente . 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 curso EP2 e descarregue o arquivo com o "esqueleto" do código `ep2_mac2166_web.c` ou `ep2_mac2166_web.py`[1]. Complete esse arquivo em seu editor preferido ou diretamente dentro do SAW. Preencha o cabeçalho declarando que você não praticará qualquer plágio e nem facilitará que o seu código seja copiado.

Por simplicidade, na sequência deste enunciado usaremos apenas o termo `ep2_mac2166_web` para designar o arquivo `ep2_mac2166_web.c` ou `ep2_mac2166_web.py`.

O arquivo `ep2_mac2166_web` é o arquivo que contém o `main` e as diversas funções para controle do jogo. Você precisará editar o `ep2_mac2166_web` para codificar as funções e, ao finalizar, submetê-lo no *SAW*. Não se esqueça que você deve conseguir fazê-lo passar nos testes automáticos.

No arquivo `ep2_mac2166_web`, 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 $num\_linhas \times num\_colunas$ (sendo $1 < num\_linhas < 20$ e $1 < num\_colunas < 54$). Cada elemento do jogo é representado por um caractere diferente:

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

```
|                          ^                |
|                          ^                |
| V V V V V V V V V V V    ^ V V V V VV     |
| V V V V V V V V V V V      V . V V VV     |
| . .          .     . .     ^   .  .  .    |
|  .        . .   . . .      ^   .   .  .   |
|     .   . .   .   .   . .     ^   . .  . . |
|      .                   .            .  .|
|      . .     . .   .     .    A    .       |
```

[1] Mais detalhes sobre os passos, as implementações e funções a serem escritas, podem ser encontradas diretamente no código ou nos comentários do referido arquivo.

Note que, da forma como o arquivo `ep2_mac2166_web` 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 caractere `|`. 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 **diretamente dentro do *SAW***, isso é essencial! Não deixe para testar no último dia, pois se tiver erros, não haverá tempo para correções.

Usuário **C**: se deseja compilar em seu computador, procure usar o `GCC`.

Usuário **Python**: se deseja executar em seu computador, use a versão 2 do **Python**, pois é esta a instalada no *SAW*.

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_mac2166_web`. 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 `num_linhas`+1 e a quantidade de colunas pelo valor `num_colunas`+1 e ambas as variáveis estão declaradas/definidas no início do código.

Convenções.

Nas descrições a seguir, para simplificar a notação, explicaremos as funções de cada passo destacando os valores que a função deve devolver de alguma forma para quem a invocou, indicando que a mesma "devolve ou altera" uma lista do tipo `[...]`. Por exemplo, na primeira função explicada (`moveCanhao`), ela devolve/altera variáveis do tipo `[bool, int, int]`, indicando que existem 3 valores que essa função precisa computar e devolver de alguma forma para quem a invocou (o primeiro representando *booleano* e os dois seguintes inteiros). Isso é feito de modo distinto em **C** e em **Python**.
No caso da linguagem **Python** isso é feito com um único comando (como `return [val1, val2, val3];`), mas no caso da linguagem **C**, na qual é impossível devolver mais que um valor simples, deve-se usar os parâmetros passando-os por referência. Assim, se a funćão precisa alterar 3 valores (mesmo que fossem todos simples - matriz NÃO é simples), precisaríamos de 2 deles por referência e o terceiro poderia ser devolvido com `return`.

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_mac2166_web` 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 tem código, complete-a e faça testes. 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_mac2166_web` 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 caractere referente à movimentação e a segunda ser chamada apenas nas rodadas pares, a entrada (para o movimento do canhão) 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 devolvem valores. A função `moveCanhao` devolve um **booleano**[2] que vale `True` se o canhão foi destruído e `False` caso contrário. Já a função `moveNaves` devolver ou alterar 3 valores, sendo dos tipos:

```
[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:

[2] Constantes equivalentes aos valores lógicos verdadeiro ou ao falso, No caso de **C**, seria qualquer valor não nulo e o valor nulo. No caso de **Python**, seriam as constantes `True` e `Fase`.

```
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 devolvidos 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_mac2166_web` 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 caractere referente à emissão de um *laser*, a entrada de dados 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 caractere 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 `proximo_pseudo`[3]. Com o número "aleatório" atual, fique com o valor 1 se ele for ímpar e 0 em caso contrário.

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

A função `emiteLaserCanhao` devolve ou altera dois valores, sendo dos tipos:

```
[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.

[3] Assim como feito no EP1, o gerador de números (pseudo) aleatórios deve ser inicializado com alguma semente. Isso já está sendo feito no início do arquivo modelo com `num_pseudoaleatorio = 127;`.

A função `emiteLasersNaves` devolve ou alterar dois valores, sendo dos tipos:

```
[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 devolvidos 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.

Passo 4: movimentação dos *lasers*

Busque no `ep2_mac2166_web` 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* do canhão 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 devolvem essas informações da mesma forma que as funções do passo anterior.

A função `moveLasersCanhao` devolve ou altera 2 valores, sendo dos tipos:

```
[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` devolve ou altera 2 valores, sendo dos tipos:

```
[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 devolvidos 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 laço principal do jogo.

Passo 5: laço 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_mac2166_web` 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 devolve ou altera 2 valores, sendo dos tipos:

```
[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 com a seguinte mensagem:

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

Não altere a cadeia de caracteres (*string* acima. Utilize exatamente da forma como apresentada. Como a própria mensagem 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`.

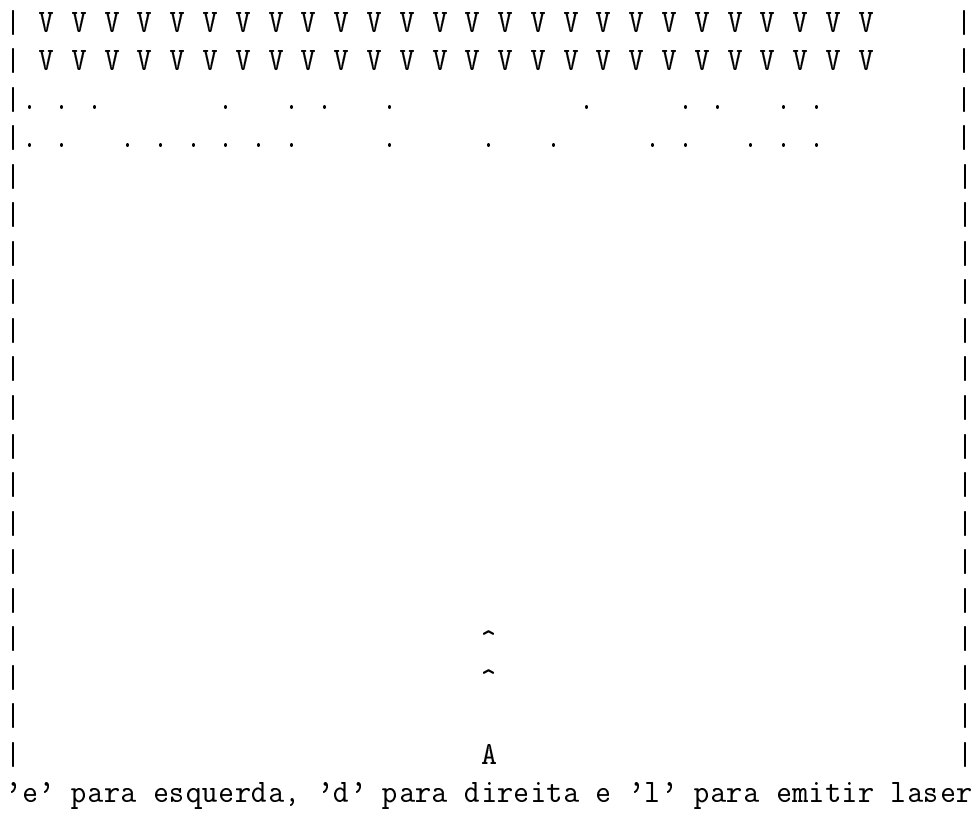
4 Entrega do EP

O arquivo `ep2_mac2166_web` deverá ser submetido na área especificada na *SAW*.

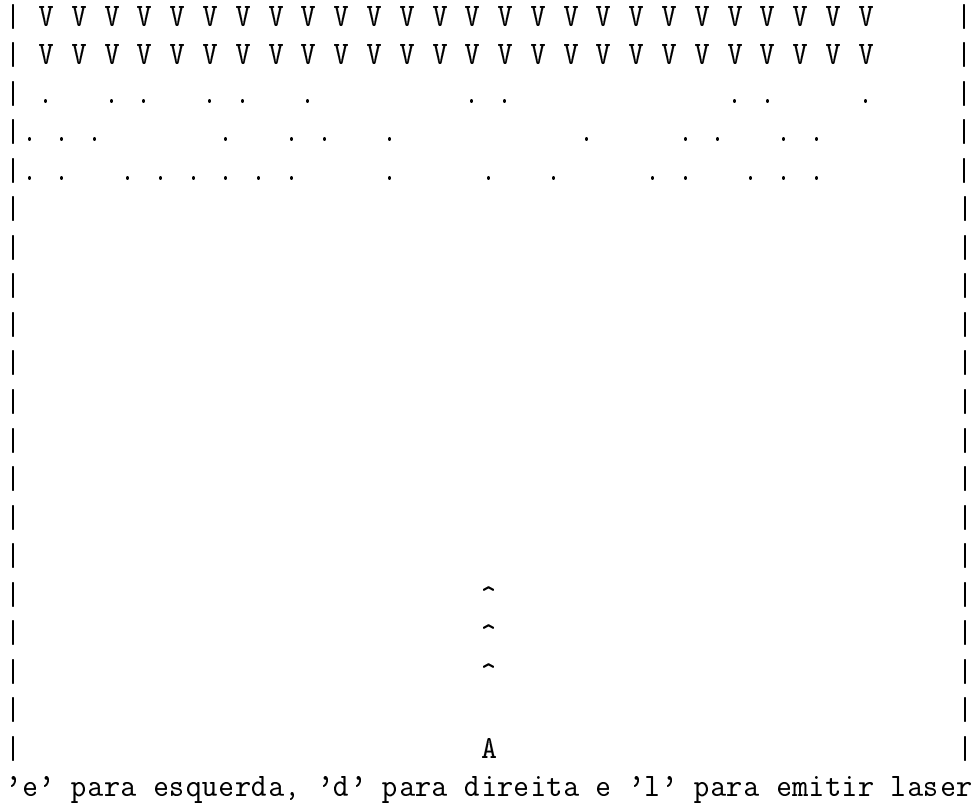
Lembre que **na implementação deste EP você só poderá usar recursos do python apresentados em sala de aula pelo seu professor. Leia as instruções gerais para entrega de EPs em** https://www.ime.usp.br/~mac2166/infoepsPy/

Apêndice – Uma partida com a saída esperada do EP

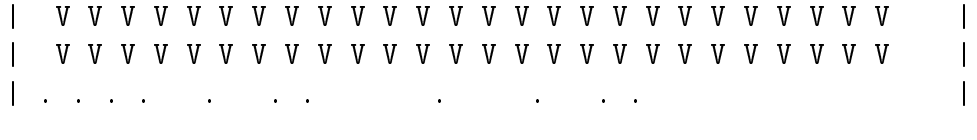
```
Digite o numero de naves (inteiro maior que 1 e menor que 53): 52
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|V V V V V V V V V V V V V V V V V V V V V V V V V V      |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                                                         |
|                            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
```

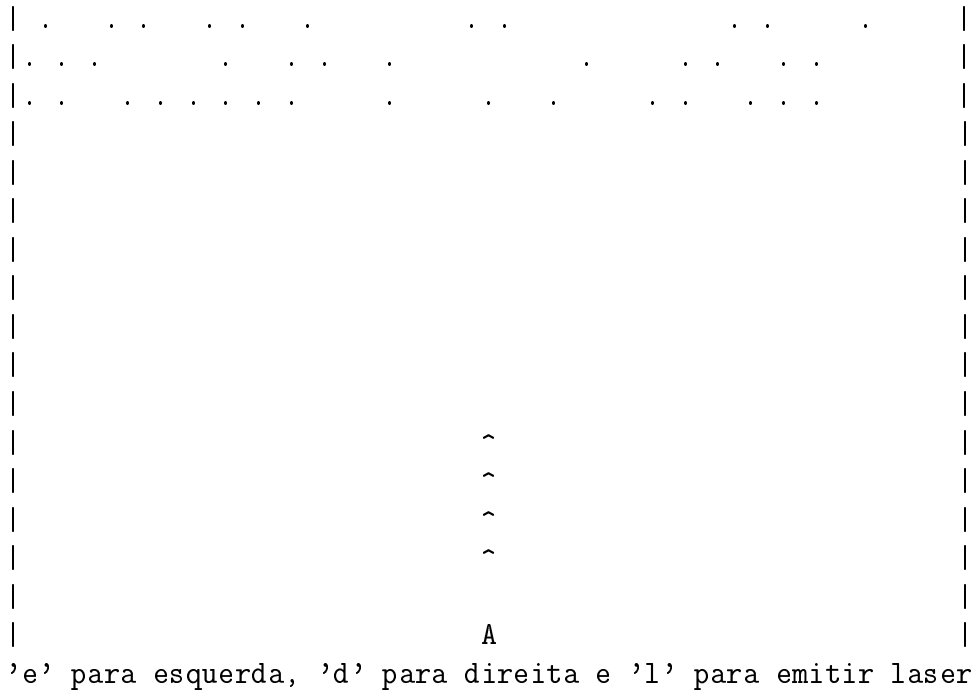


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

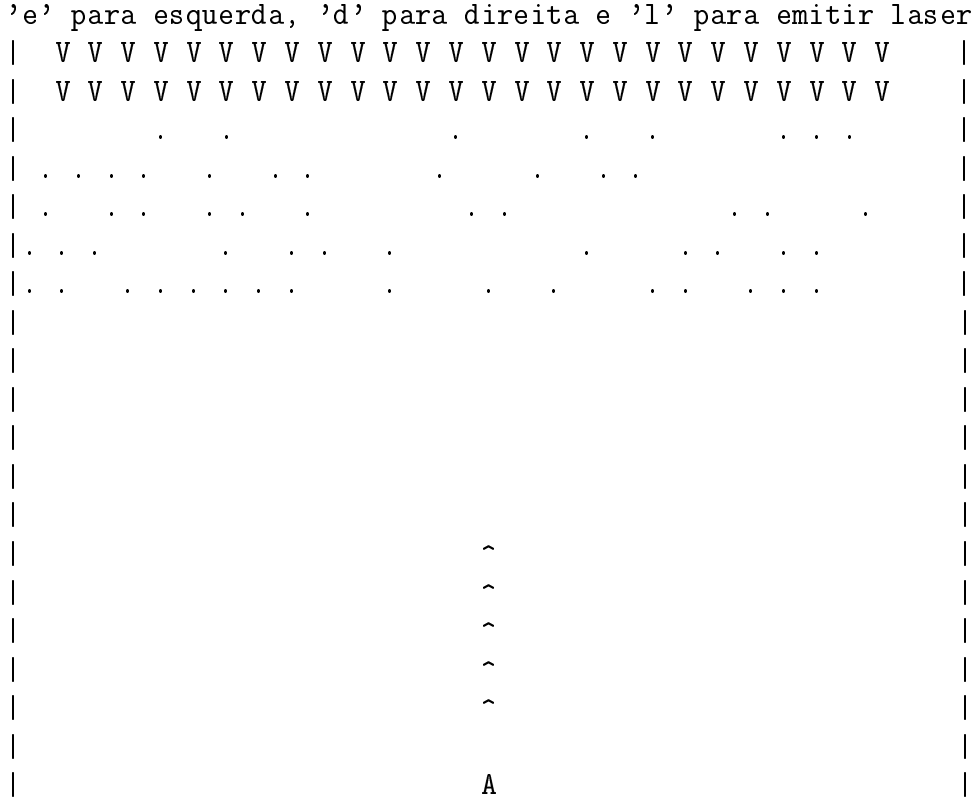


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

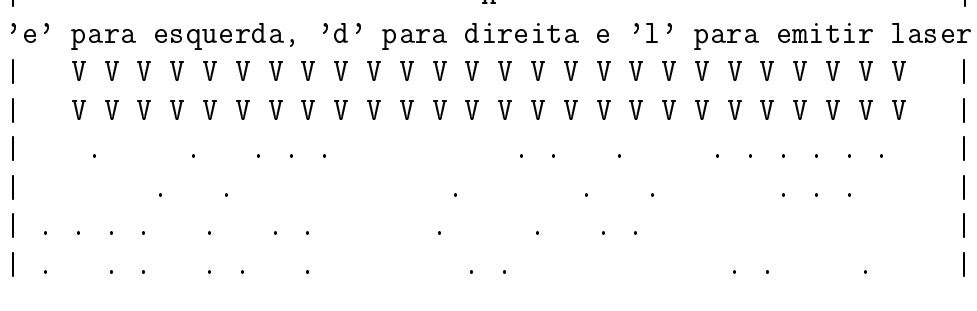


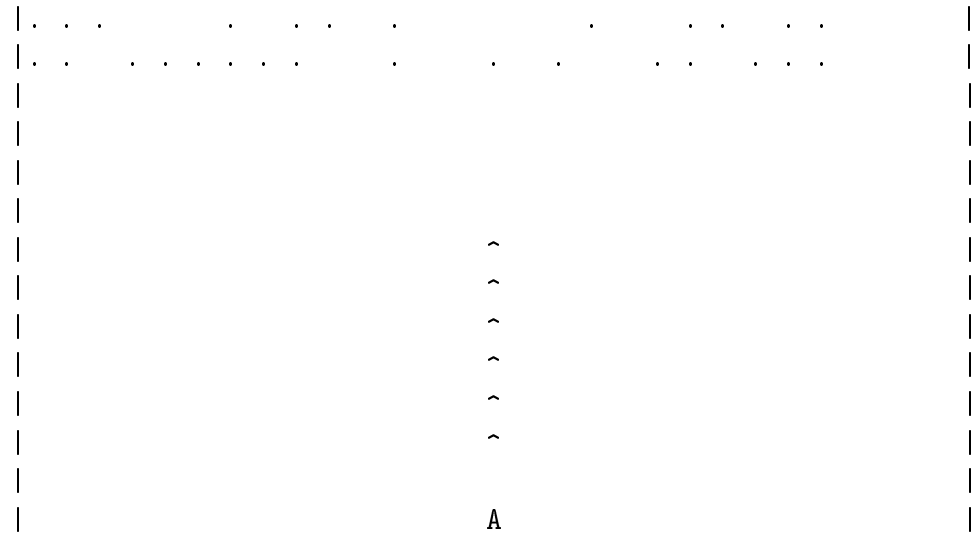


'e' para esquerda, 'd' para direita e 'l' para emitir laser: l

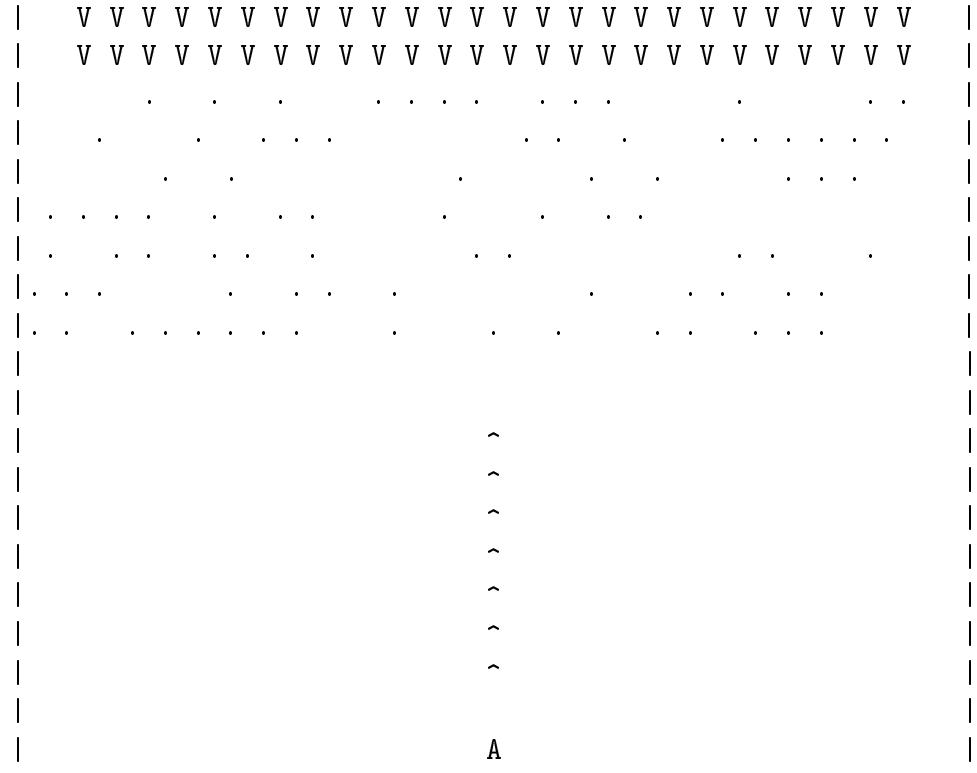


'e' para esquerda, 'd' para direita e 'l' para emitir laser: l

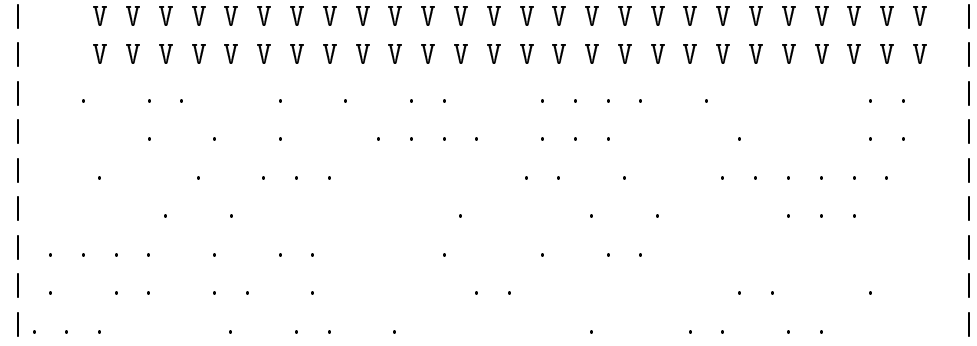


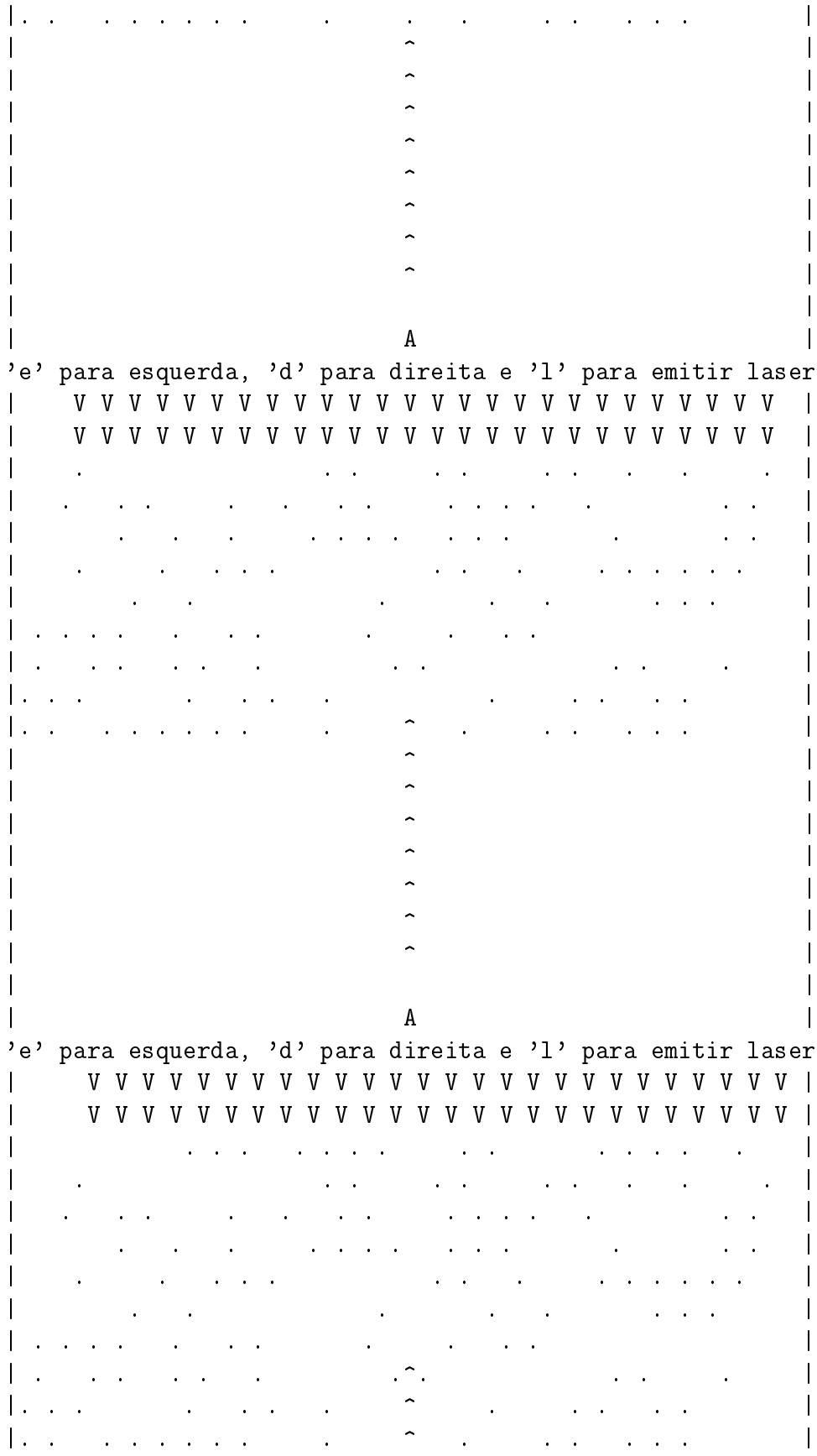


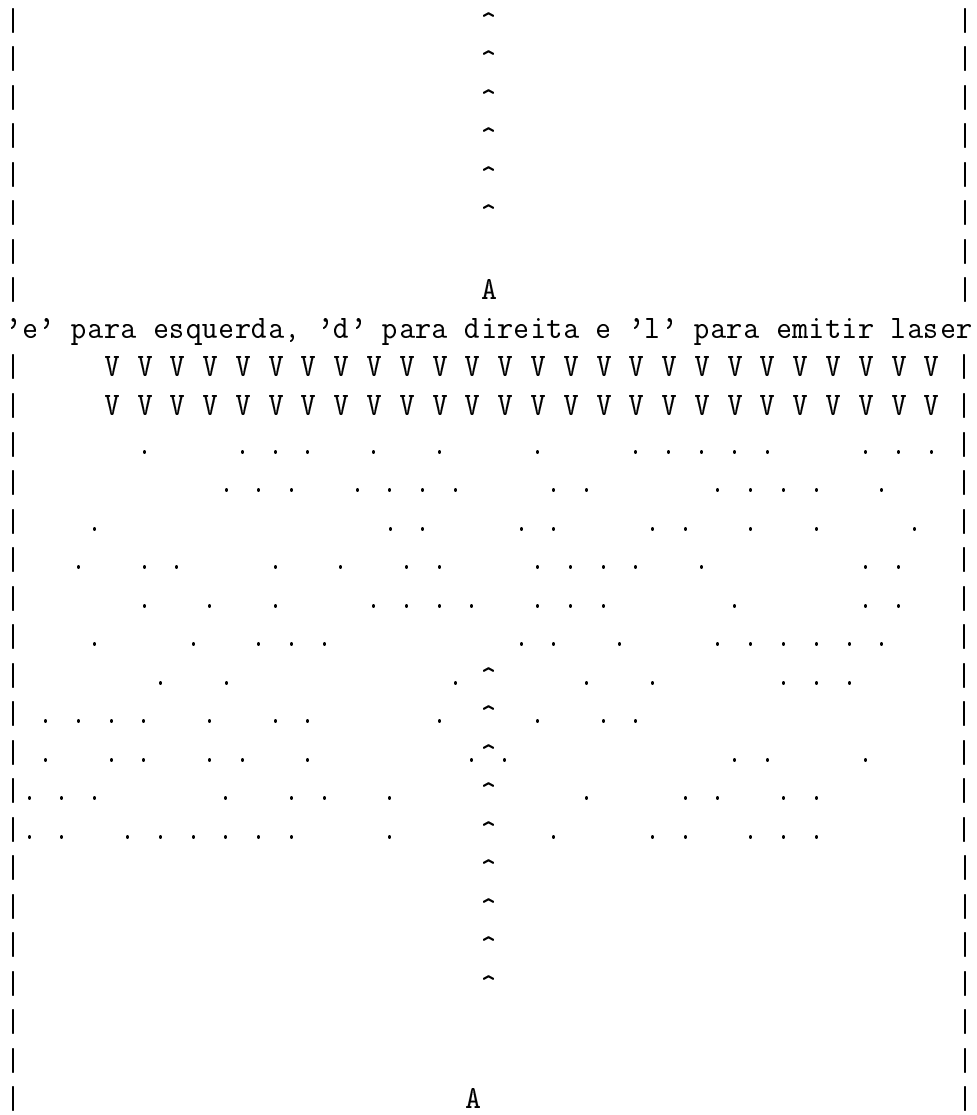
```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

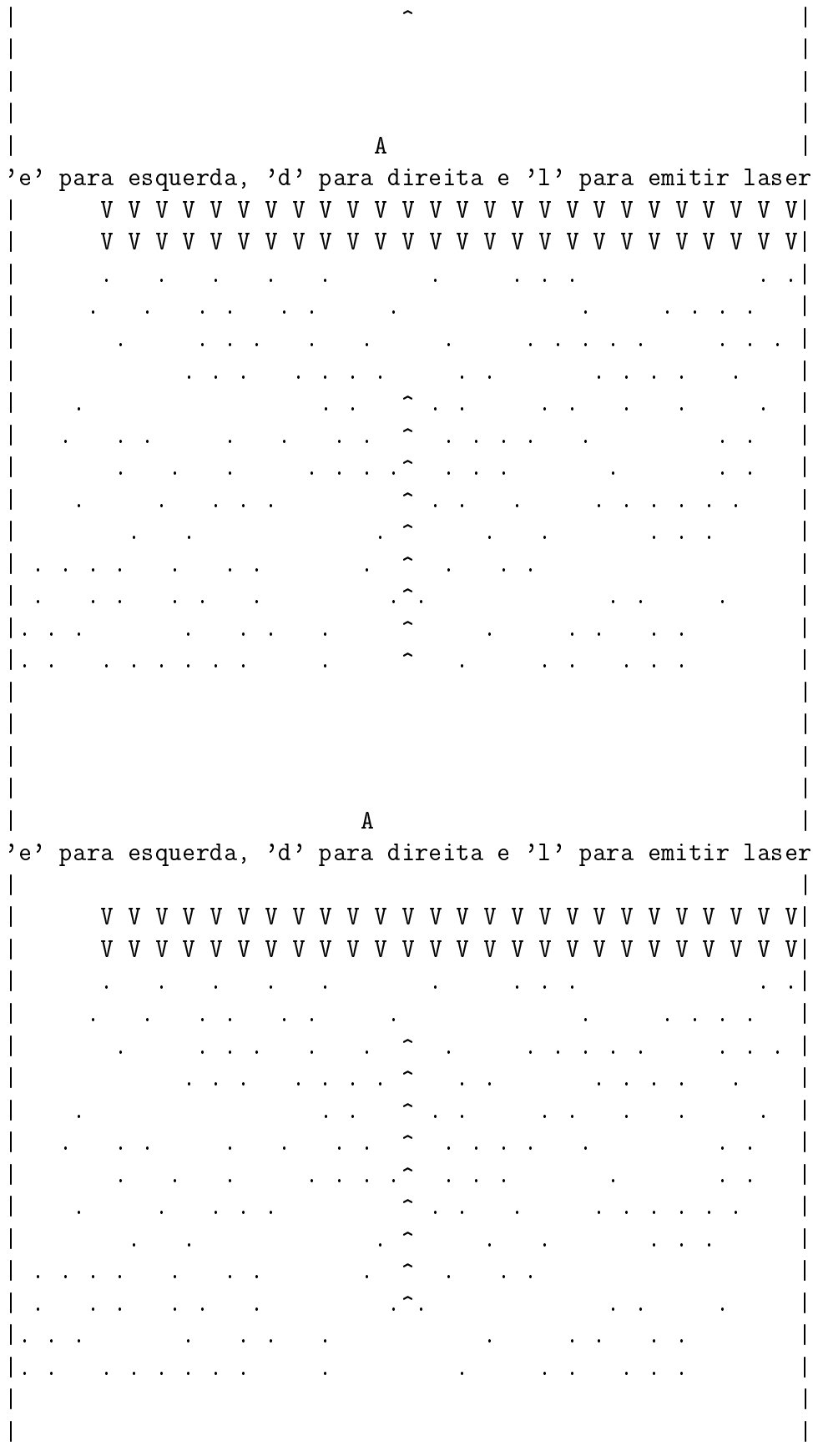


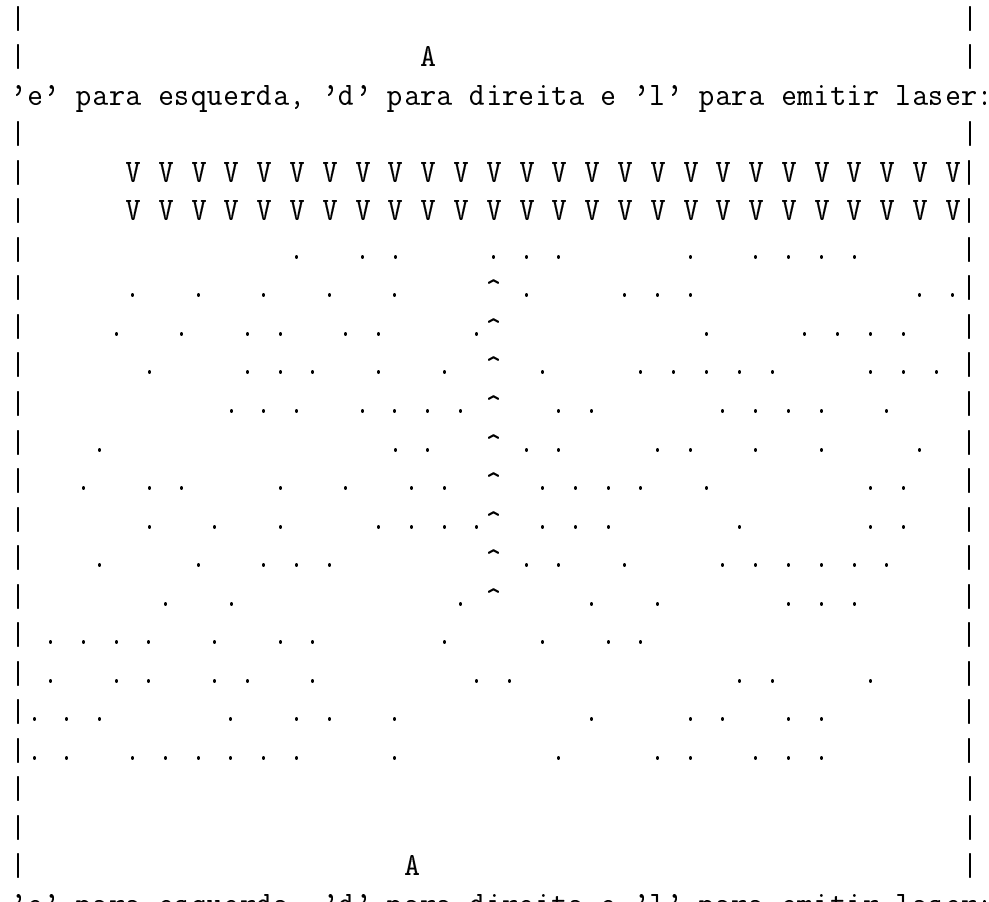
```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

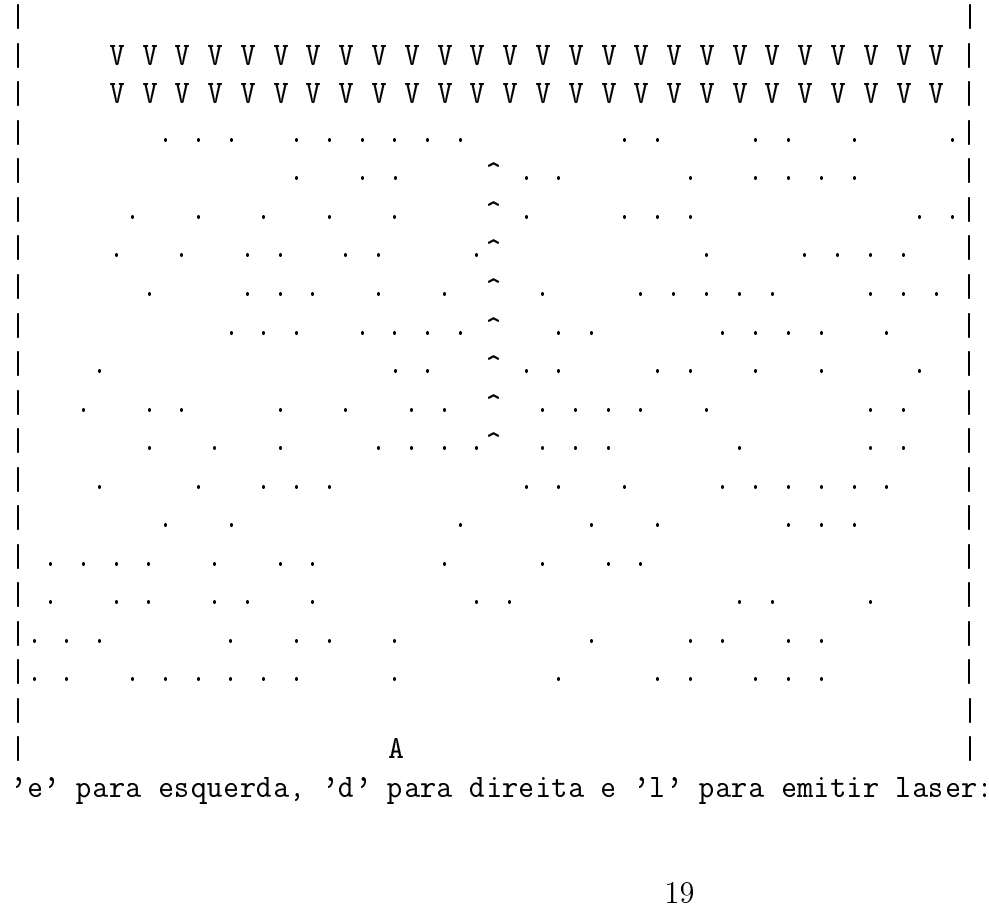



'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l

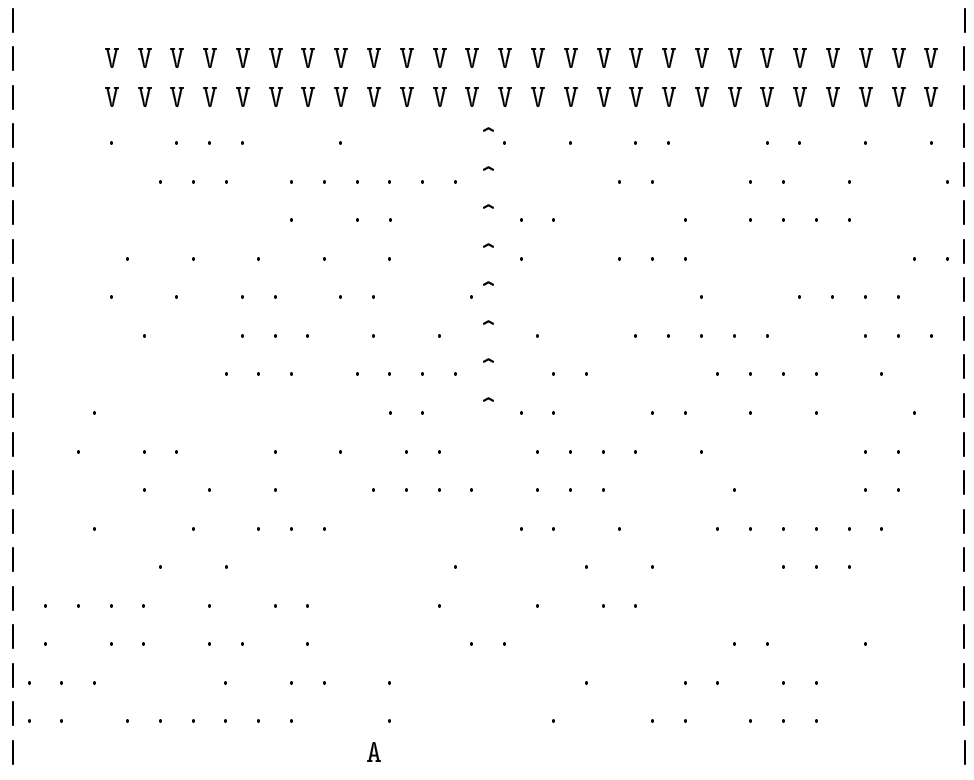

'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e


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

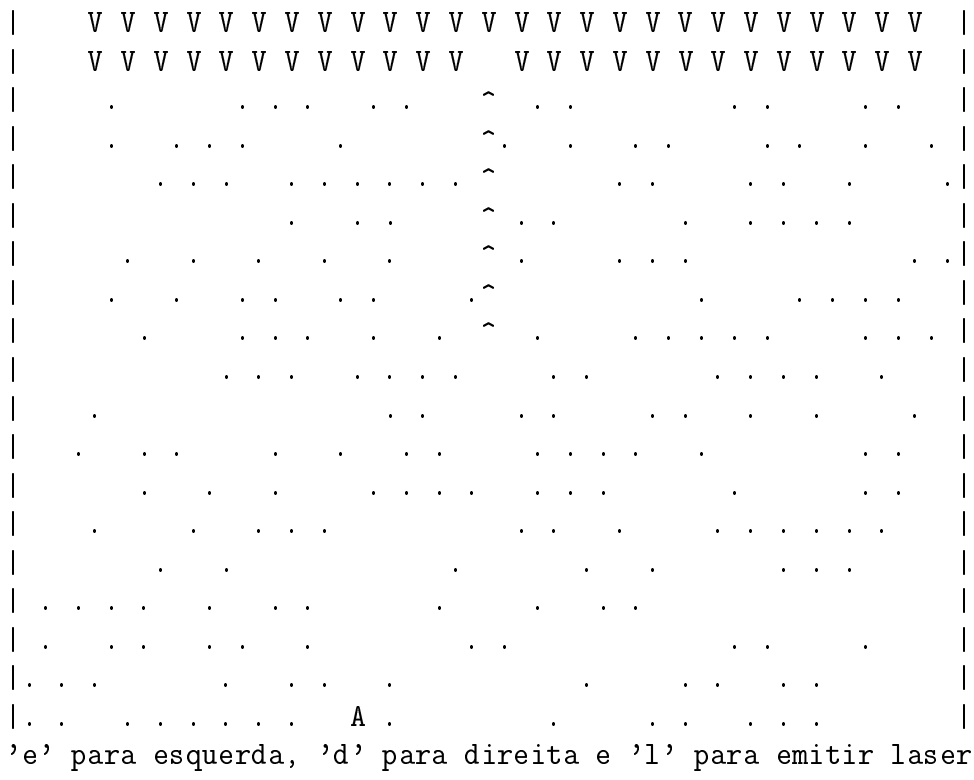

A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e


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

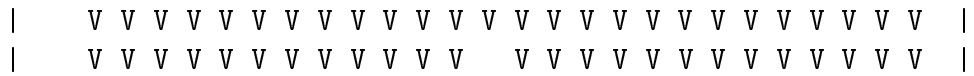
A
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e

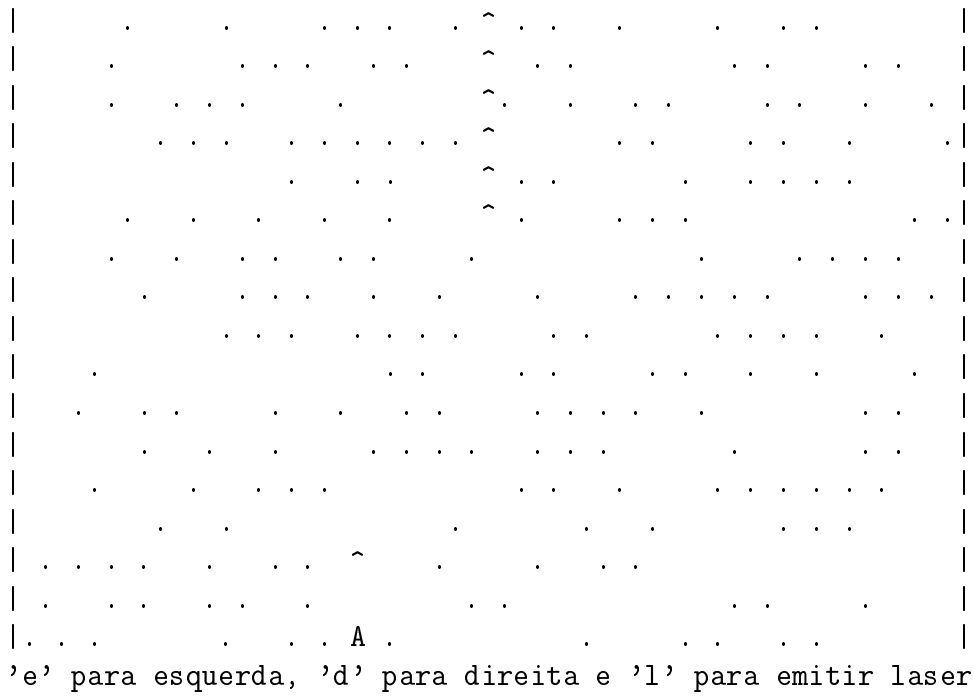


’e’ para esquerda, ’d’ para direita e ’l’ para emitir laser: e

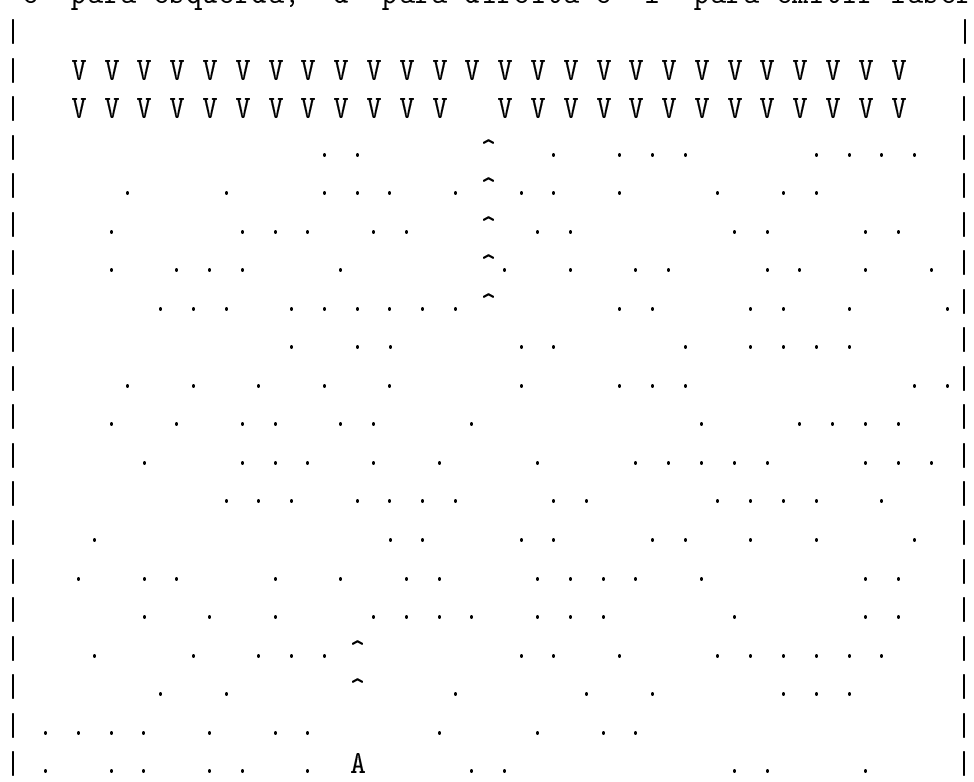


’e’ para esquerda, ’d’ para direita e ’l’ para emitir laser: l

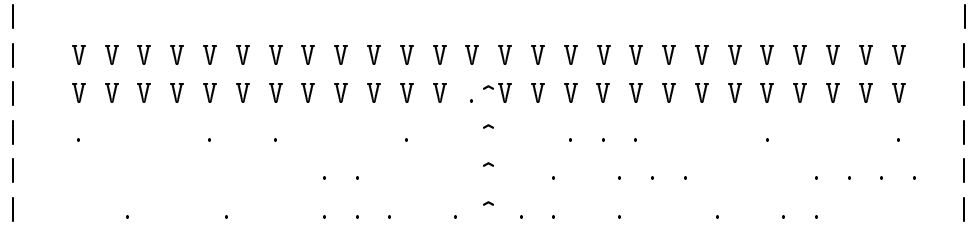


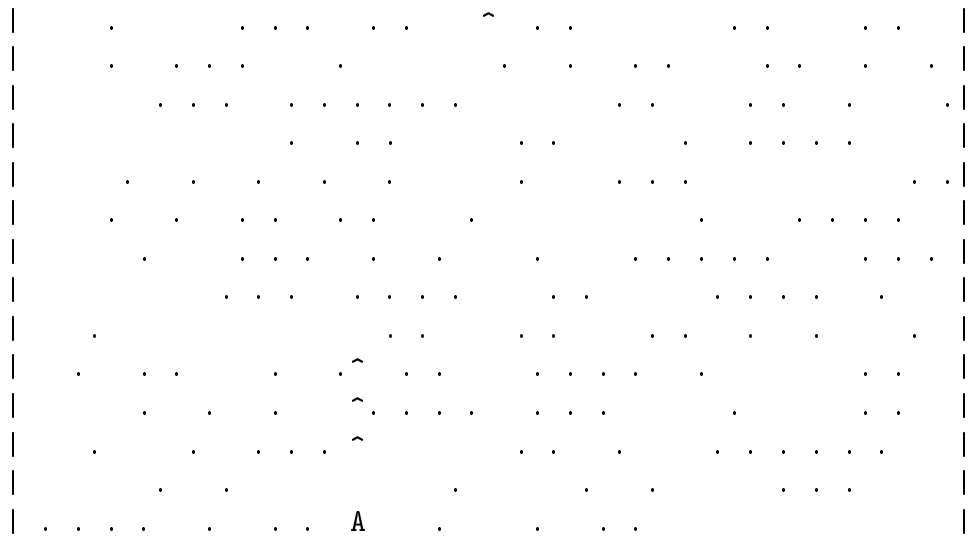


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

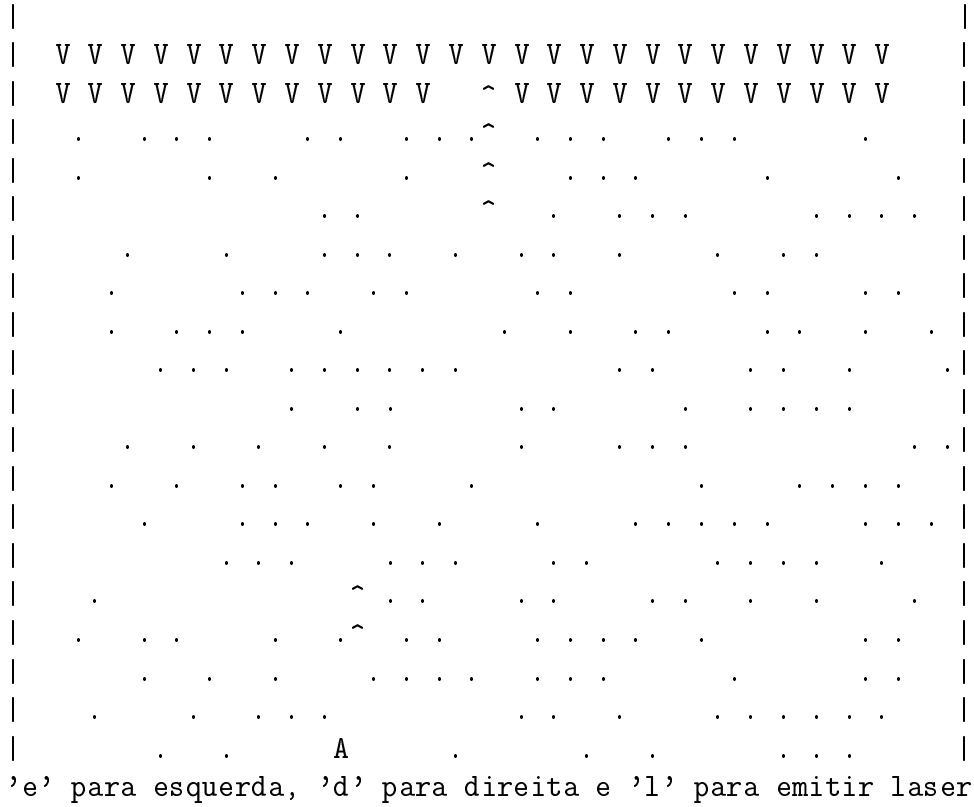


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

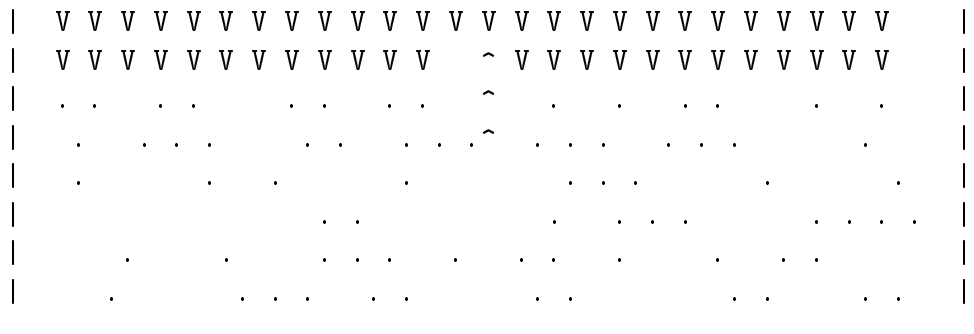


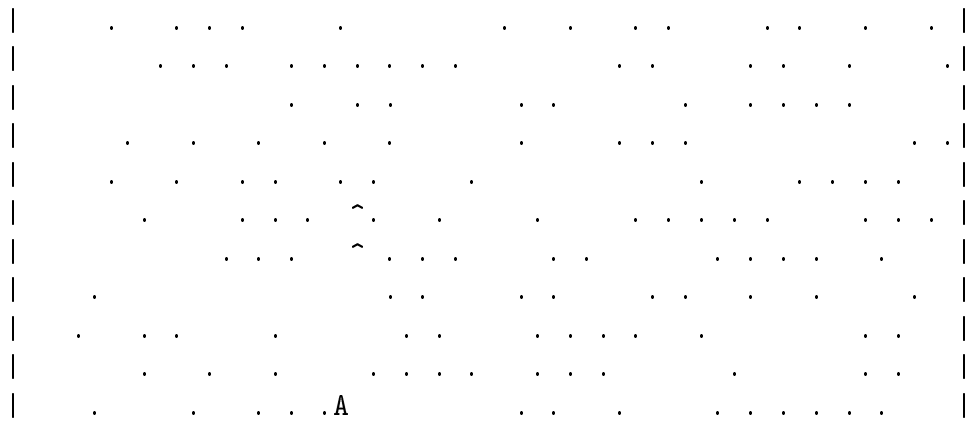


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: e
```

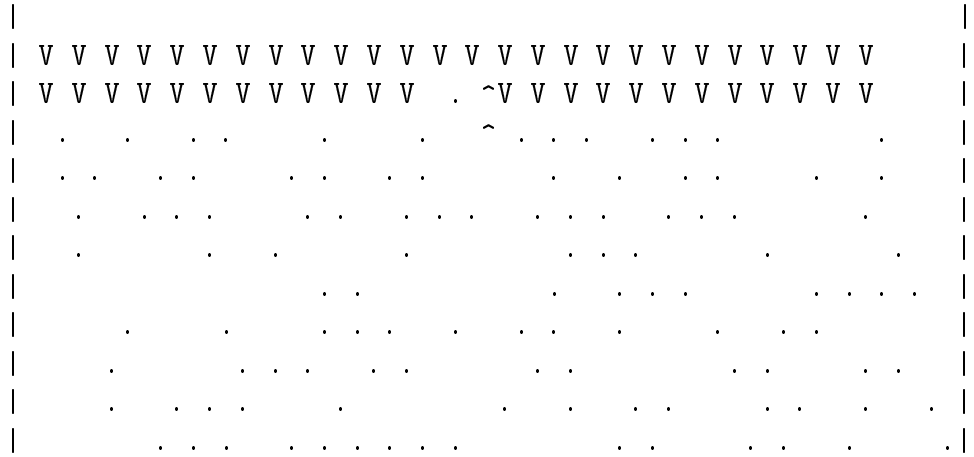


```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```

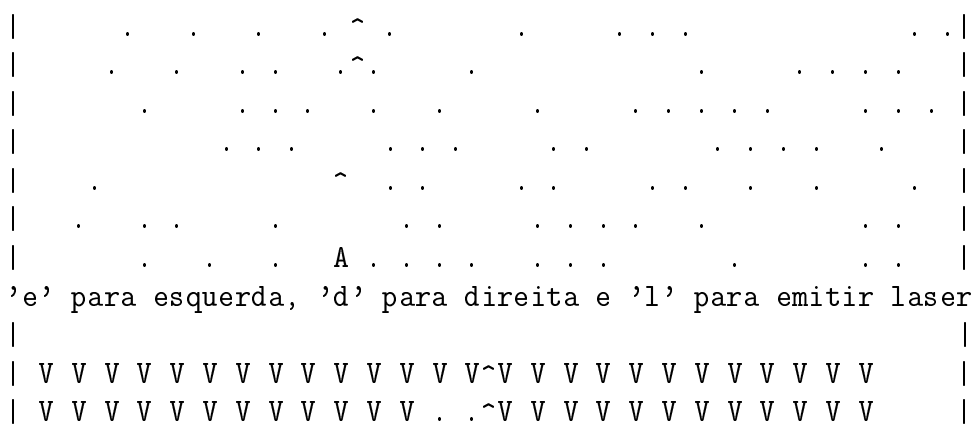


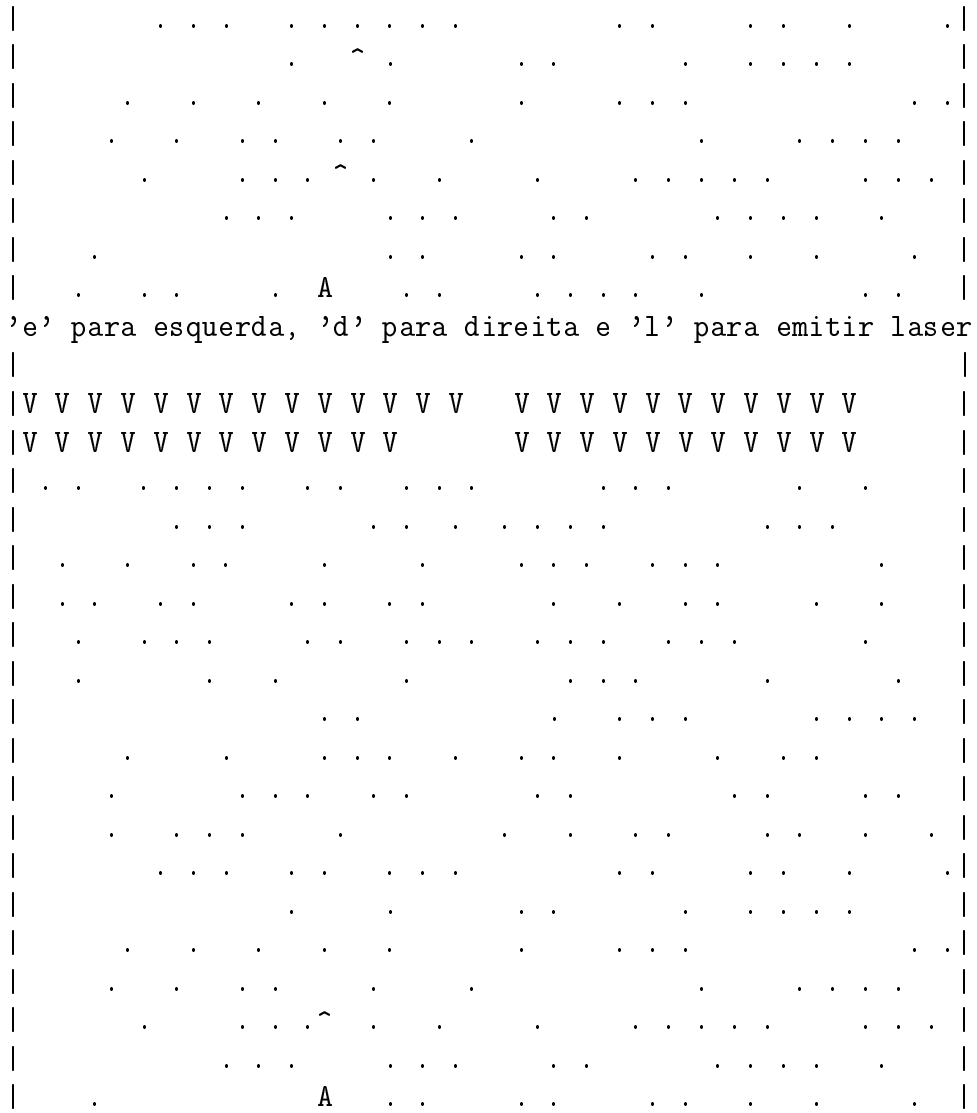


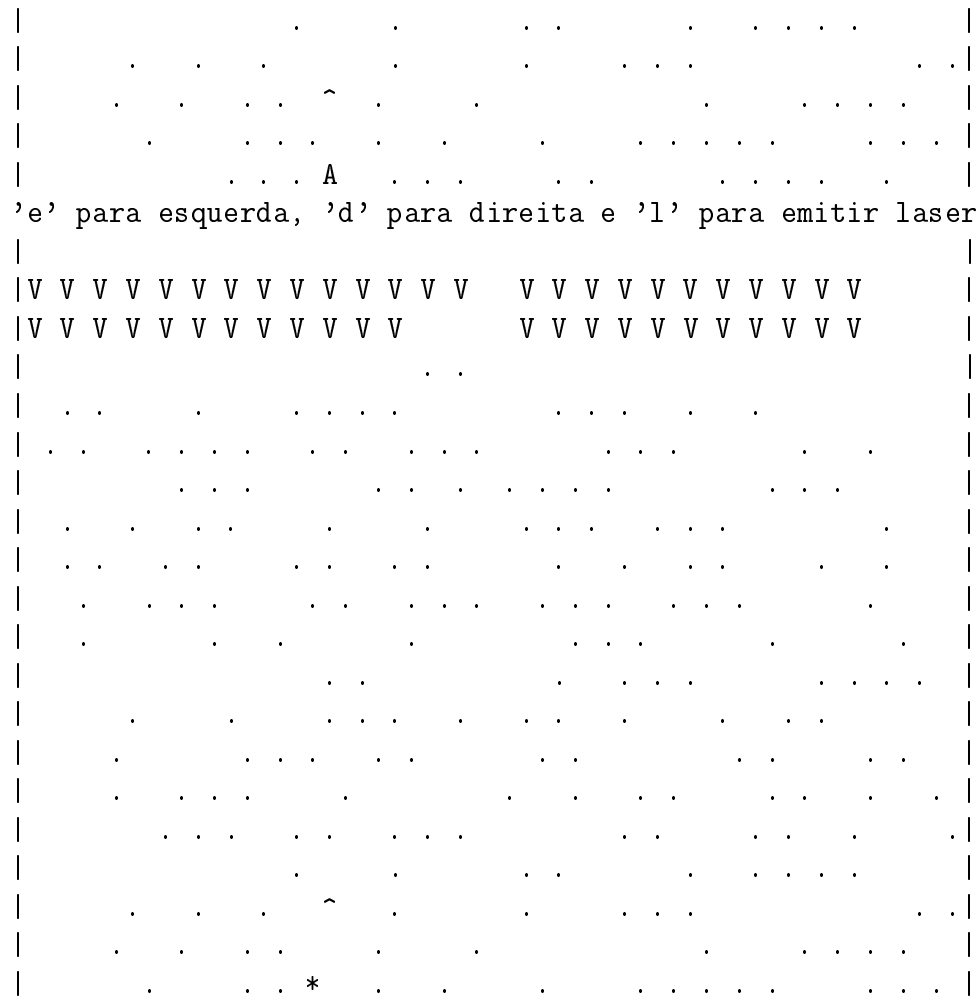
```
'e' para esquerda, 'd' para direita e 'l' para emitir laser: l
```



```
'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