clickbait image

Ricardo C Michel, 2022

BASEADO EM UM JOGO PARA ZX81

(recriação em FORTH de um jogo em BASIC publicado na revista)

(**MICROSISTEMAS** n. 53, de fev. 1986, pág. 32,)

(de nome "Estratégia",)

(de autoria de Jorge A. C. Bettencourt Soares)

(https://datassette.org/revistas/micro-sistemas/micro-sistemas-no-53)

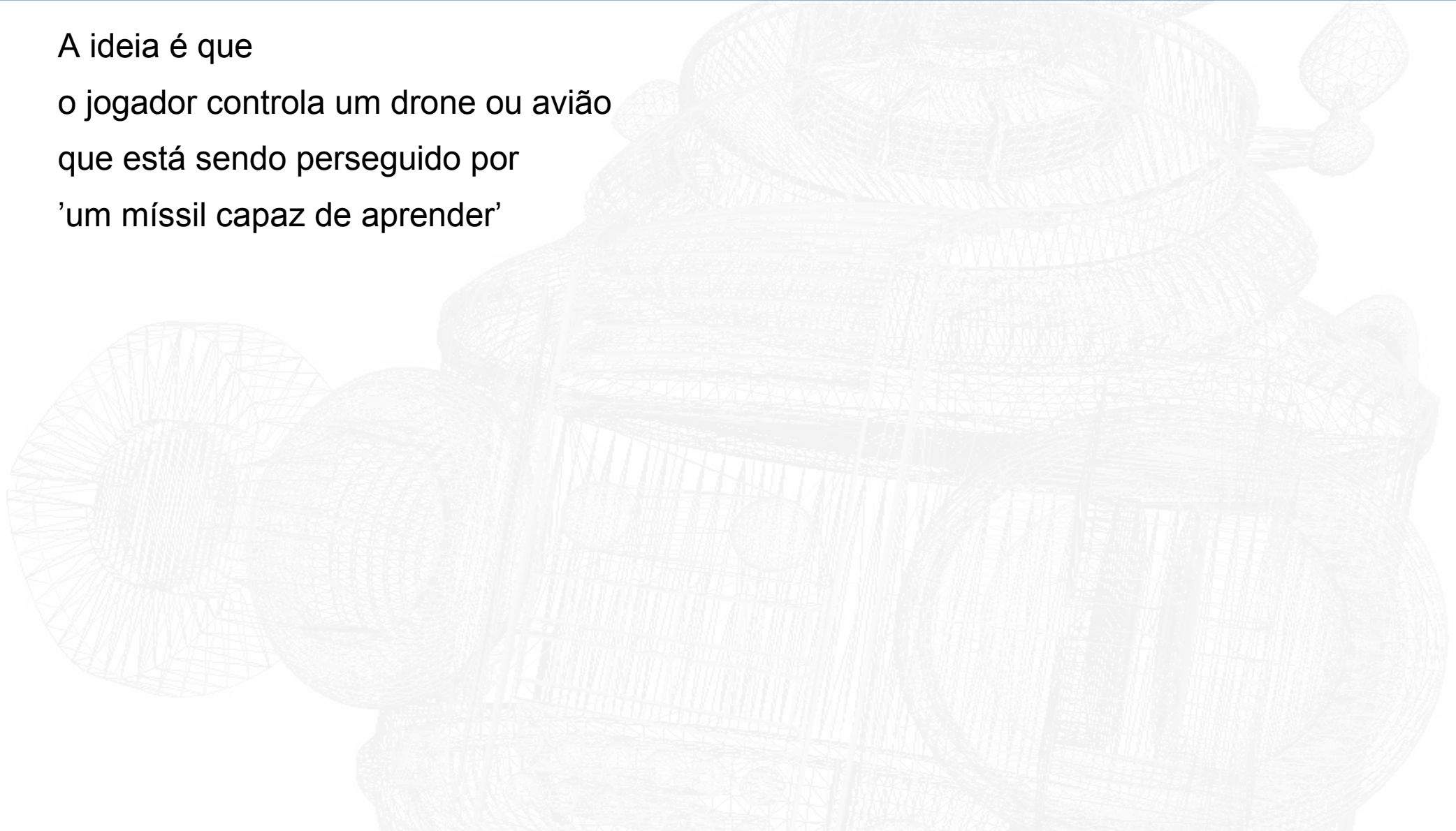
(Versão em esp32FORTH por Ricardo Cunha Michel, Brasil, 2022)

A ideia é que

o jogador controla um drone ou avião

que está sendo perseguido por

’um míssil capaz de aprender’



A ideia é que

o jogador controla um drone ou avião

que está sendo perseguido por

’um míssil capaz de aprender’

Uma vez que é baseado em um jogo dos anos 80,

em baixíssima resolução,

o drone é representado por uma letra “**A**”

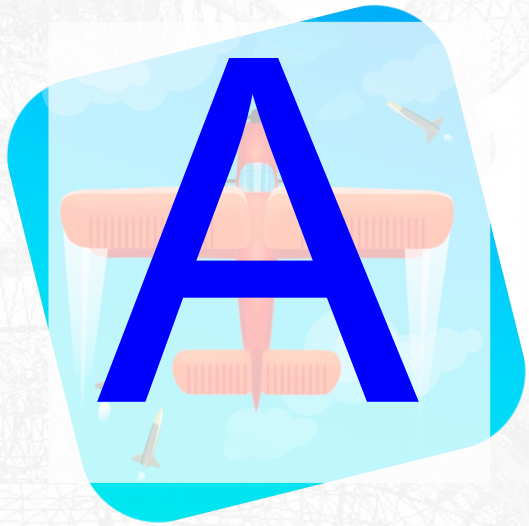
e o míssil por um perigosíssimo asterisco: “*****”

A ideia é que

o jogador controla um drone ou avião

que está sendo perseguido por

’um míssil capaz de aprender’

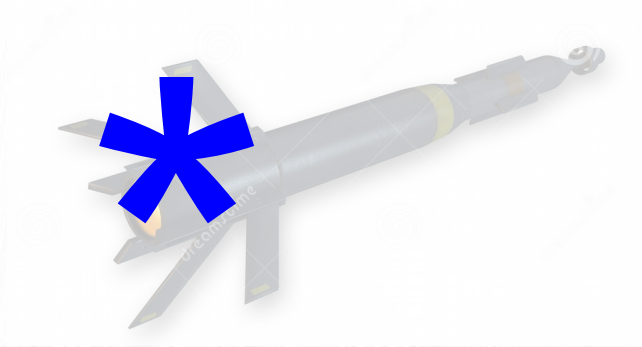


Uma vez que é um jogo dos anos 80,

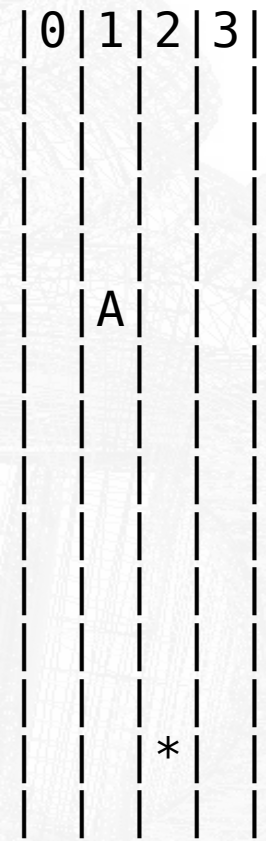
em baixíssima resolução,

o drone é representado por uma letra “**A**”

e o míssil por um perigosíssimo asterisco: “*”



O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.



O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

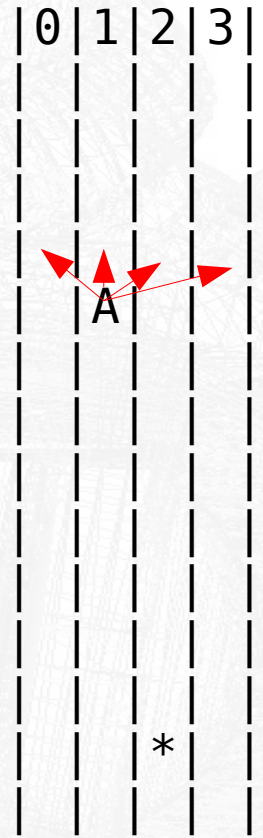
A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o espaço seguro, o piloto ganha aquela rodada.



O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

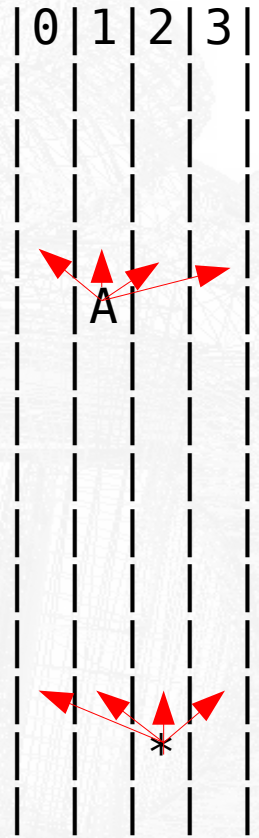
A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o espaço seguro, o piloto ganha aquela rodada.



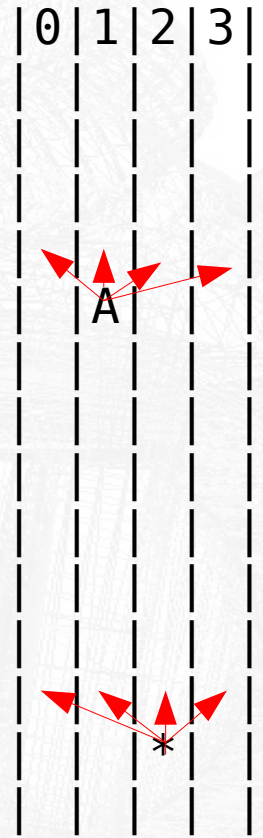
O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

O míssil considera que o piloto não é capaz de realizar mudanças realmente aleatórias,
isto é: o piloto empregará um padrão de troca de caminhos, reconhecível pelo míssil.



O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

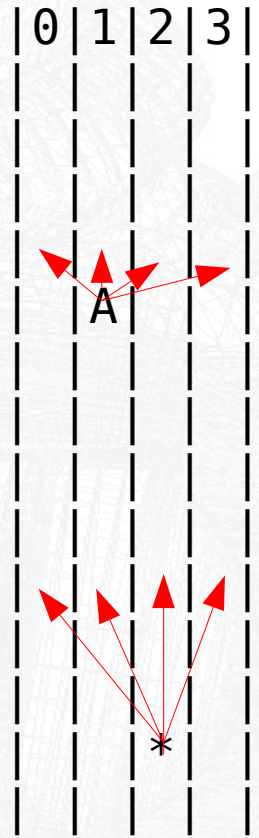
A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o espaço seguro, o piloto ganha aquela rodada.

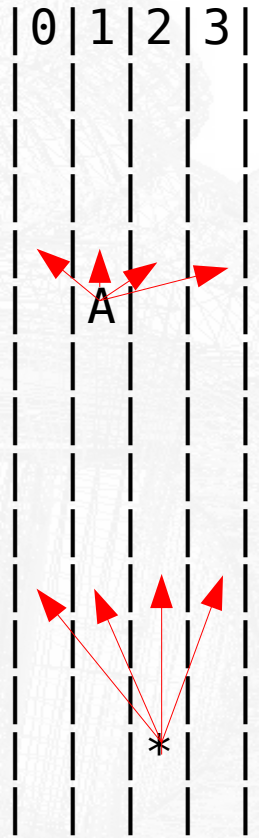


O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

O míssil tem “uma chance em quatro” de acertar, por puro acaso,
avançando três posições.
Em cada uma das outras três tentativas frustradas,
o jogador avança uma posição (ou seja: também avança três posições).

Este arranjo manteria drone e míssil na mesma distância um do outro, em média,
se o jogador não manifestasse um padrão reconhecível.

Como resultado, esse arranjo permitiria que o jogador ganhasse 50% das rodadas, se jogasse aleatoriamente.



* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o espaço seguro, o piloto ganha aquela rodada.

O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

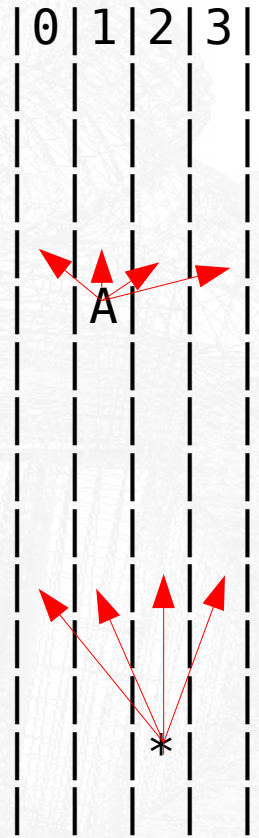
A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o espaço seguro, o piloto ganha aquela rodada.



O jogador pode controlar o drone em quatro ‘caminhos’ pelos céus.

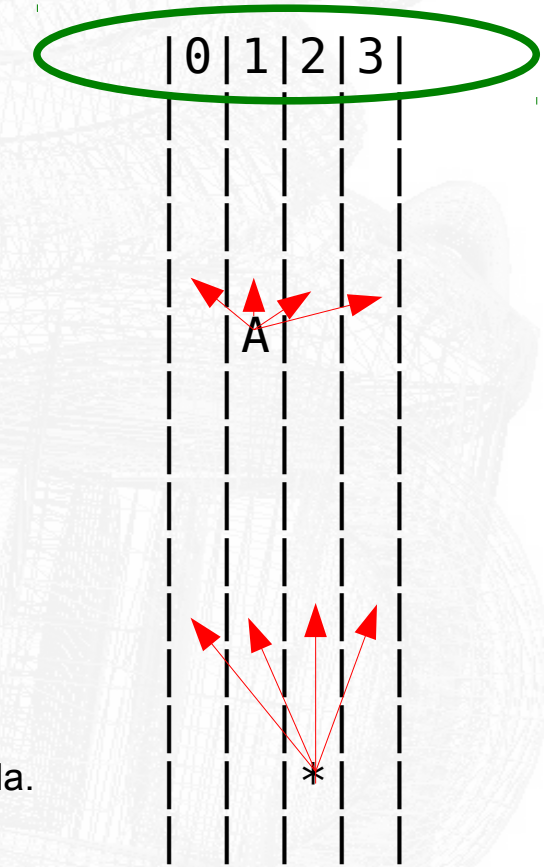
A cada momento,

* o jogador pode mover o drone
do caminho atual para qualquer outro,
ou mantê-lo no caminho atual;

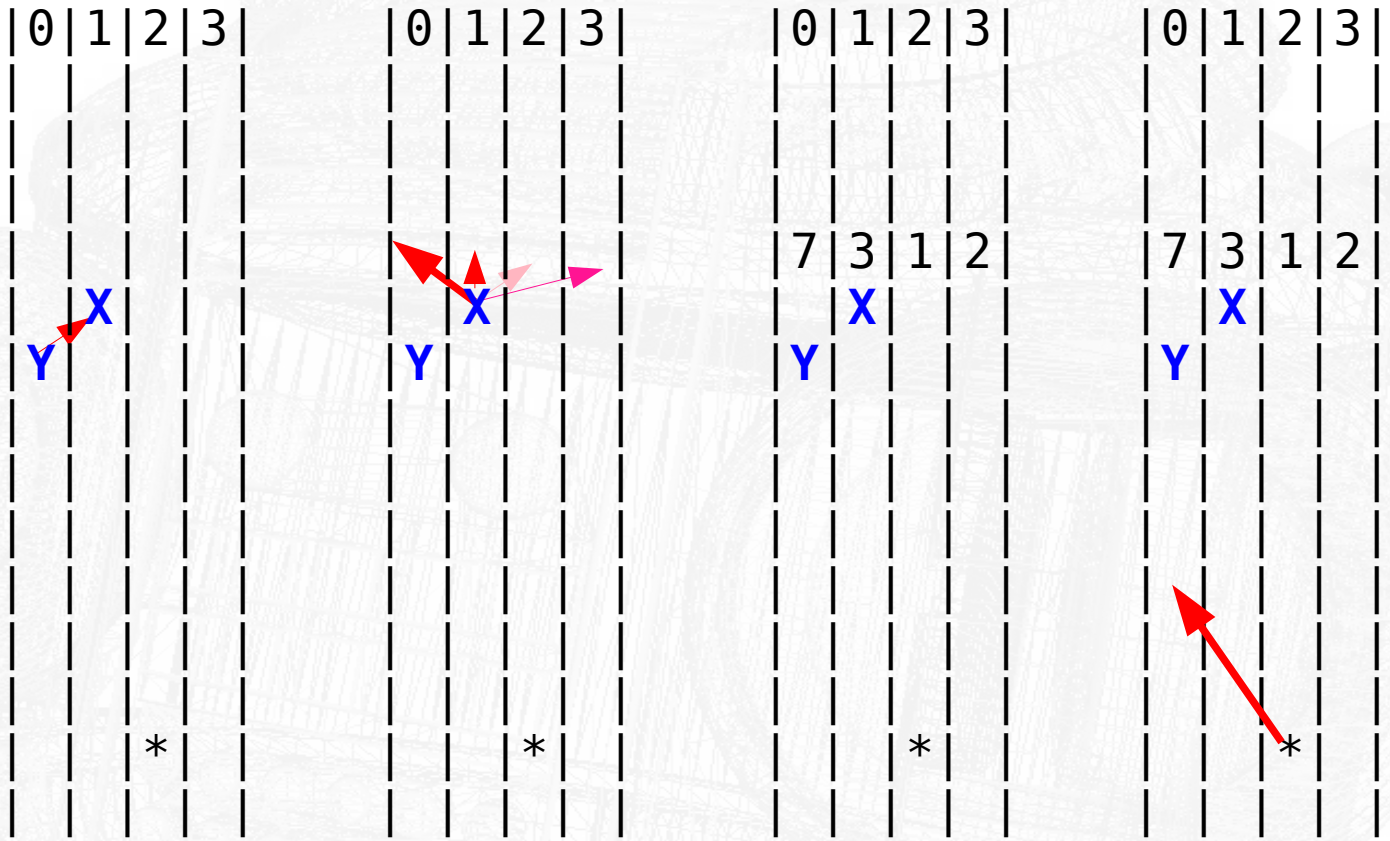
* o míssil tentará se posicionar no mesmo caminho do drone,
usando sua memória de eventos passados (*1);

* se os caminhos coincidirem,
o míssil avança três posições,
caso contrário, o drone avança uma posição (*2).

*Se o drone alcançar o **espaço seguro**, o piloto ganha aquela rodada.

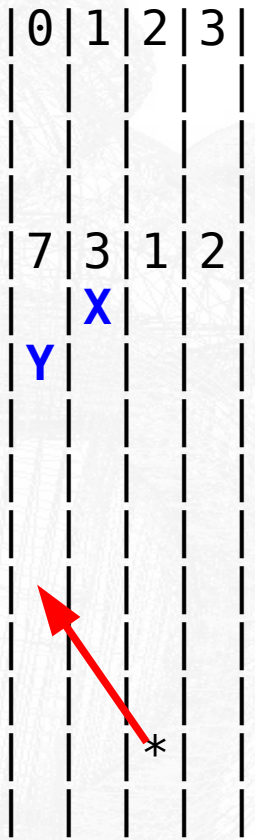


O míssil lembra quantas vezes o piloto se moveu para cada uma das quatro posições
nas vezes anteriores que ele esteve na posição atual, **X**,
tendo vindo da posição anterior, **Y**.



Ou seja:

em cada momento que o jogador executar uma ação,
o míssil, sabendo onde o drone está e de onde veio,
localiza na memória a *situação* correspondente
e verifica qual foi, no passado, o movimento mais frequente (*3) e o executa.



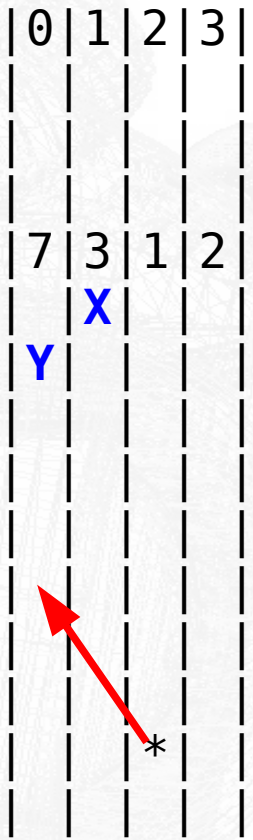
(*3):
o míssil poderia
distribuir a probabilidade posicionamento, em função da frequência de cada uma,
ao invés de ir apenas para a situação mais frequente.

Operacionalizando:

em cada momento que o jogador executar uma ação,
o míssil, sabendo onde o drone está e de onde veio,
localiza na memória a situação correspondente
e verifica qual foi, no passado, o movimento mais frequente (*3) e o executa.

Depois,
ocorre a atualização de posições
(com o míssil aproximando-se três posições ou o drone afastando-se uma posição),
e o míssil atualiza sua memória, em função da mudança realizada pelo piloto do drone
(independentemente de o míssil ter ou não ter 'adivinhado' corretamente).

(*3):
o míssil poderia
distribuir a probabilidade posicionamento, em função da frequência de cada uma,
ao invés de ir apenas para a situação mais frequente.



“ O míssil lembra quantas vezes o piloto se moveu para cada uma das quatro posições
nas vezes anteriores que ele esteve na posição atual, x,
tendo vindo da posição anterior, y. ”

Assim, **se**
para cada posição X (onde X pode valer 0, 1, 2, ou 3),
o piloto veio da posição anterior Y (onda Y também pode valer 0, 1, 2, ou 3),
então existem 16 situações possíveis (de 00 até 15).

Para cada uma das 16 situações, existem 4 possibilidades de movimento (0,1,2 ou 3)
e o míssil armazena
o número de vezes que o drone foi movido para cada uma delas
(usando 64 posições de memória).

Variáveis do míssil (nomes provisórios, de antes de escrever o programa)**:**

drone_atual (armazena o caminho atualmente seguido pelo drone – inteiro entre 0 e 3)

drone_anterior (armazena o caminho seguido pelo drone antes de mudar para o caminho atual – inteiro entre 0 e 3)

situacao (determina o número da situação atual, uma das 16 possíveis – inteiro entre 0 e 15)

situacao = drone_anterior * 4 + drone_atual

0*4+0=0	1*4+0=4	2*4+0=8	3*4+0=12
0*4+1=1	1*4+1=5	2*4+1=9	3*4+1=13
0*4+2=2	1*4+2=6	2*4+2=10	3*4+2=14
0*4+3=3	1*4+3=7	2*4+3=11	3*4+3=15

O míssil determina a situação e depois verifica qual a posição (0, 1, 2 ou 3) para onde o drone mais frequentemente foi.

freq (armazena a frequência com que o piloto foi para cada posição, em cada uma destas situações)

freq(0,1,2,3) = situacao * 4 + possivel_posicao_futura(0, 1, 2, ou 3)

O míssil irá para a posição com maior frequência para esta 'situação'.

para_onde_foi (armazena a posição para onde o piloto efetivamente se moveu)

(inteiro entre 0 e 3, variável usada para atualizar freq)

Plano:

<INICIO GLOBAL>

Zera as 64 posições de memória e todas as variáveis do míssil

<INICIO>

Define posição inicial do drone e do missil para começar nova rodada

Desenha caminhos aéreos, míssil e drone

<CICLO>

Jogador move o drone

O míssil, percebendo o movimento, mas sem conhecê-lo, calcula a '**situação**'

O míssil verifica as frequências de deslocamento do jogador para as posições 0, 1, 2 e 3

Míssil seleciona a posição para onde irá (a mais frequente, na primeira versão)

Caso posição do míssil == posição do drone : míssil aproxima três linhas

Caso posição do míssil <> posição do drone : drone afasta uma linha

'refresh' da tela (Desenha caminhos aéreos, míssil e drone)

Atualiza variáveis do míssil (drone_anterior = drone_atual; drone_atual=para_onde_foi)

Se (linha_drone <> linha_missil) & (linha_drone > linha_seguro), vai para **<CICLO>**

Se (linha_drone <= linha_seguro), vai para **<VITORIA>**

Se (linha_drone == linha_missil), vai para **<FIM DE JOGO>**

<VITORIA>

Congratulações

Vai para **<INICIO>**, preservando o que aprendeu deste piloto.

<FIM DE JOGO>

Vitória da IA

Vai para **<INICIO>**, preservando o que aprendeu deste piloto.

Implementação:

No original, a tela é mapeada, i.e.: pode-se escrever em qualquer coordenada da tela.

No esp32Forth, tem-se um terminal, não sendo possível escrever em linhas já transmitidas.

Solução:

transmitir tudo de novo a cada vez.

MAS…

como mudar a posição do drone e do míssil a cada vez que redesenhar a tela?

Como incluir drone e míssil na 'imagem', em suas posições corretas?

```
variable dr_altura      ( o quão perto o drone está da regiao segura )
variable dr_pos         ( posição do drone em um dos 4 caminhos aéreos )
variable dr_char        ( caracter que representa o drone, "A")

variable mi_altura      ( onde está o missil )
variable mi_pos         ( posição do míssil em um dos 4 caminhos aéreos )
variable mi_char        ( caracter que representa o míssil, "^" )


        : zona1   ." ########################" cr ;
        : zona2   ." #### FORA DE ALCANCE ####" cr ;
        : zona3   ." ########|0|1|2|3|########" cr ;
        : zona4   ." ////////| | | | |////////" cr ;

        : zona_dr s" ////////| | | | |////////" 2dup drop 9 dr_pos @ 2 * + + dup dr_char @ swap c! rot rot TYPE 32 swap c! cr ;
        : zona_mi s" ////////| | | | |////////" 2dup drop 9 mi_pos @ 2 * + + dup mi_char @ swap c! rot rot TYPE 32 swap c! cr ;


                : desenha
                    cls
                    zona1 zona1 zona2 zona1 zona3
                    29 0 do
                    i dr_altura @ = if zona_dr else
                    i mi_altura @ = if zona_mi else
                    zona4 then then
                    loop ;
```

Como o jogador move o drone?

```
: MOVE_dr begin key? until key 48 - dr_pos ! ;  ( o código das teclas numéricas menos 48 é o valor do dígito )
```

Como o míssil se moveria, aleatoriamente (isto é, sem IA)?

```
: MOVE_mi_alea  4 CHOOSE mi_pos ! ;    ( 'n CHOOSE' gera um número aleatório inteiro entre 0 e n-1 )
```

Depois que drone e míssil se movem, como atualizar as suas 'alturas'? (essa atualização depende do acerto do míssil)

```
: ATUALIZA_posicoes  mi_pos @ dr_pos @ = if 3 mi_altura var- else 1 dr_altura var- then ;
```

onde

```
: var- dup @ rot - swap ! ;  ( n var_name -- )  ( subtrai "n" unidades do valor armazenado na variável )
```

Poderia ter usado "-!", mas não lembrei dela na época. Depois, usei "+!"

Como determinar que o jogo terminou?

```
: TESTA_FIM
    mi_altura @ dr_altura @ <=
        if ." >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>MORREU" 1 derrotas +! cr 0 dr_ok?  ! then
    dr_altura @ 0 <=
        if ." <><><><><><><><> V I T O R I A <><><><><><><><> " 1 vitorias +! cr 0 dr_ok? ! then
    ;
```

"**0 dr_ok?**" serve para sinalizar a saída do loop da rodada.

Como o míssil se move usando a IA?

Definições iniciais

```
( IA ************************************************************ )
variable ia_dr_atual    ( a posição atual que o míssil vê o drone, X )
variable ia_dr_anterior ( onde o drone estava antes de estar na posição atual, Y )
variable situacao       ( situação do drone, uma das 16 situações possíveis )

: init_IA  ( como não há informação sobre como o drone chegou nesse posição, assume ambos os valores como sendo iguais )
    dr_pos @ dup ia_dr_atual ! ia_dr_anterior ! ;

: tabela CREATE 64 cells allot DOES> swap cells + ;   ( 'posição nome_da_tabela' deixará no stack o endereço da posição )
tabela ia_tab_freq      ( cria a tabela com 64 posições, com as 4 frequências de ocorrência para cada uma das 16 situações )

: zera_tab_freq 64 0 do 0 i ia_tab_freq ! loop ;
zera_tab_freq


( UTILIDADES: )

: mostra_tab_freq
    16 0 do
        i . ." : "
        i 4 * 0 + ia_tab_freq ? ." : "
        i 4 * 1 + ia_tab_freq ? ." : "
        i 4 * 2 + ia_tab_freq ? ." : "
        i 4 * 3 + ia_tab_freq ?
        cr loop ;

: max rot > if swap then drop ;
```

Como o míssil se move usando a IA?

A implementação da busca e atualização das frequências de movimento, que simula Inteligência (e é efetiva!):

```
: MOVE_mi_IA
      ( calcula e armazena o valor de 'situação' )
    ia_dr_anterior @ 4 * ia_dr_atual @ + situacao !
      ( agora, recordar as 4 frequências de movimento... )
    situacao @ 4 * 0 + ia_tab_freq @
    situacao @ 4 * 1 + ia_tab_freq @
    situacao @ 4 * 2 + ia_tab_freq @
    situacao @ 4 * 3 + ia_tab_freq @
      ( . . . e compará-las )
    3 mi_pos !
    3 0 do
    2dup > if 2 i - mi_pos ! swap then nip
    loop
    drop
    cr mi_pos ? Cr ;

< STYLE WARNING >
< para um FORTH bem fatorado, a palavra acima deveria ter sido dividida em três palavras menores, uma para cada operação >


: ATUALIZA_IA
    ia_dr_atual @ ia_dr_anterior !  ( atualiza posicoes )
    dr_pos @ ia_dr_atual !          ( coloca a posição detectada como sendo a nova 'posição atual' )
    situacao @ 4 * ia_dr_atual @ + ia_tab_freq dup @ 1 + swap ! ;  ( atualiza a tabela de frequencias da situação ocorrida )

( ************************************************************ FIM da IA )
```

Como controlar uma rodada?

```
( cada execução do jogo )
: UM_JOGO
init_vars ( nao zera memoria )
init_IA
desenha
begin
    dr_ok? @
    while
        MOVE_dr
        MOVE_mi_IA
        ATUALIZA_posicoes
        ATUALIZA_IA
        desenha
        TESTA_FIM
    repeat
;
```

Como controlar as diversas rodadas?

```
( a chamada inicial >> AQUI começa o jogo )
: JOGO
      cls
      ABERTURA
      0 vitorias !
      0 derrotas !
      begin
          key? key 27 -
          while
              UM_JOGO
              CR ." VITORIAS: " vitorias @ .
              CR ." DERROTAS: " derrotas @ .
              CR CR CR
              3000 ms
              ." <ESC> interrompe, outra tecla recomeça"
          repeat
;
( ********************************************************** FIM )
```

O resto do código são firulas, detalhes e limpeza de stack (e esta limpeza ainda está incompleta, sobrando um dígito na pilha após cada rodada)

O texto do jogo pode ser baixado de

https://www.dropbox.com/s/5d51whrm0v83vl5/JET.fth?dl=0

mas estará, em breve, também nos arquivos de exemplos do esp32Forth
(precisará de atualização dos comentários para o inglês, a língua do grupo)

Vamos jogar??

Muito agradecido pela atenção!