U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Programação em Lógica

2019/2020 – MIEIC

Turma: 4

Afonso Soares Mendonça
(up201706708)

Filipe Carlos de Almeida Duarte da Cunha Nogueira
(up201604129)

Xero-G:

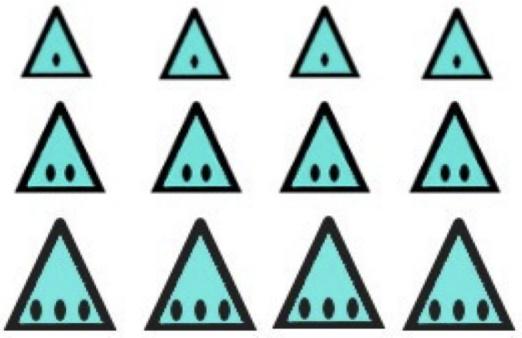
Xero-G é um jogo de estratégia que tem como objetivo levar uma nave espacial até à base do oponente. Tem duração entre 5 a 20 minutos e são necessários 2 jogadores. Este jogo é uma re-implementação do jogo “Gyges”, criado por Claude Leroy em 1985.

História:

A guerra entre humanos e aliens já se alonga há anos. Como tentativa de controlar as baixas militares, ambos os lados começaram a levar a cabo produção de naves espaciais chamadas *fighter ships*, controladas por inteligência artificial. Contudo, gradualmente, exposição a raios cósmicos causam danos que podem levar a avarias conhecidas como *bit-flips* no software das naves. Devido a estas avarias as naves tornam-se obsoletas e sem qualquer tipo de reação, ficando simplesmente à deriva no espaço. Tu és o último piloto treinado do teu planeta, o teu treino deu-te as competências necessárias para conseguires controlar as naves e é o teu trabalho, com qualquer nave que encontres, acabar com o inimigo e, consequentemente, com a guerra. Muitas vezes terás necessidade de saltar de nave em nave com o intuito de reprogramar as suas coordenadas para uma posição que te favorece estrategicamente. Planeia bem as tuas decisões e irás conseguir atingir a base do inimigo, acabando assim esta guerra inaudita.

Componentes:

- Tabuleiro 8x6, sendo que tanto a primeira como a última linha representam exclusivamente a base de cada jogador
- 12 naves, sendo que há 3 de cada nível. Não é necessário haver distinção entre um certo número de naves pois todas podem ser controladas por ambas jogadores.



Setup:

Cada jogador começa com 6 naves na linha imediatamente a seguir à sua base. Nesta linha estão representadas naves dos diferentes tipos: três pintas (nível 3), duas com duas pintas (nível 2) e duas com uma pinta (nível 1). Os jogadores podem organizar as suas naves como preferirem ao longo desta mesma linha.

Início:

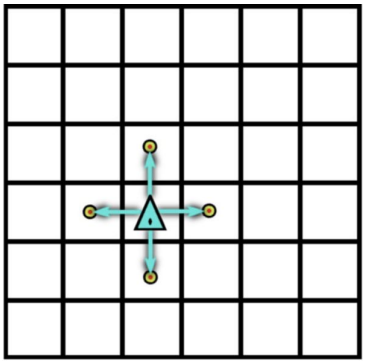
Começa o jogador que cuja base é a no topo do tabuleiro. Daqui em diante, a vez de jogar é alternada entre os jogadores

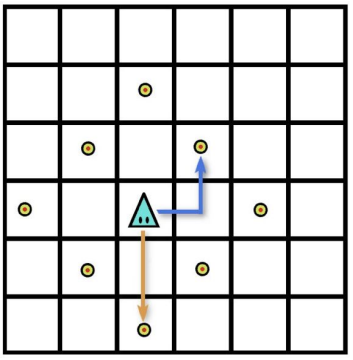
Como jogar:

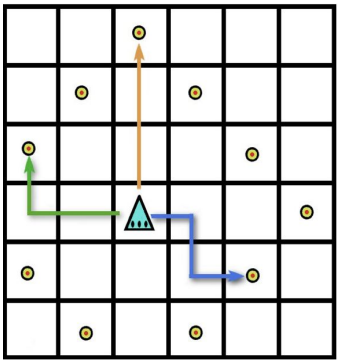
Na tua vez, o jogador encontra a linha mais próxima de si que contenha uma ou mais naves. Esta é a sua *home row.* (No início do jogo esta linha será a linha imediatamente a seguir à base.) O jogador escolhe uma nave da *home row* e dá início à sua jogada.

Movimentos:

1) O nível de cada nave (na figura o número de pintas em cada pirâmide) representa o número de movimentos para casas adjacentes que essa mesma nave pode realizar. As naves apenas se podem movimentar horizontalmente ou verticalmente. Contudo, é possível mudar de direção em cada casa.



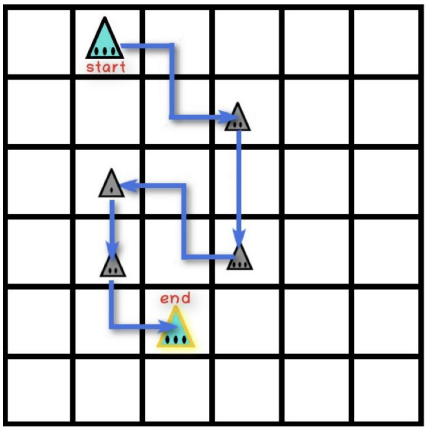




2) Os espaços onde as naves passam têm que estar vazios. Contudo, a casa de destino de cada jogada pode estar ocupada. Aterrar a nave numa casa vazia termina a jogada.

3) Caso a nave aterre numa casa ocupada por outra nave, é possível efetuar uma de duas jogadas:

- *Rocket Boost* - De acordo com o nível da nave que já se encontrava na casa, a nave recebe um *boost* (movimentos extra) de exatamente esse mesmo nível. Tal como acontece num movimento normal, todas as casas por onde a nave passa durante o *boost* têm que estar vazias. Novamente, a casa onde a nave aterra pode estar ocupada por outra nave. Neste último caso, esta jogada pode ser encadeada com outro *Rocket Boost* ou acabar com um *Reprogram Coordinates*.
- *Reprogram Coordinates* - Deixa a nave com que se efectuou a última jogada na posição onde aterrou e muda agora de posição a nave que estava a ocupar essa casa. Pode-se posicionar essa nave em qualquer uma das casas vazias do tabuleiro de acordo com as pintas da tua nave anterior. Contudo, não podes posicioná-la na base do teu oponente. Com esta jogada a tua vez termina.

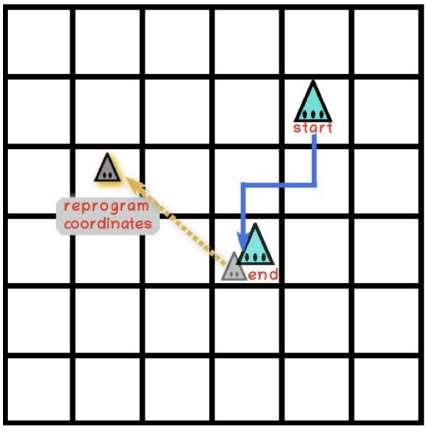


Exemplo: Rocket Boost

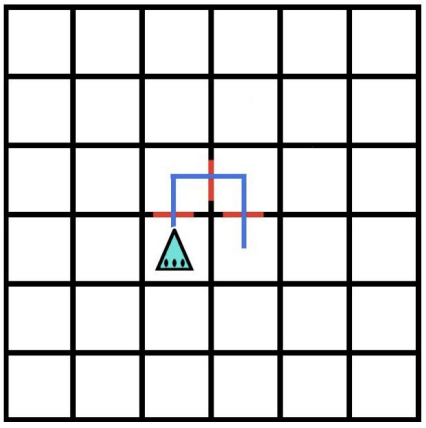
Na figura ao lado, este jogador efetua 4 *Rocket Boosts* consecutivos com a nave de nível 3 assinalada com *start*.

Exemplo: Reprogram Coordinates

No exemplo à direita, a nave nível 3 aterra numa outra nave de nível 2. O jogador pode então reprogramar as coordenadas desta última e movê-la para outra casa.

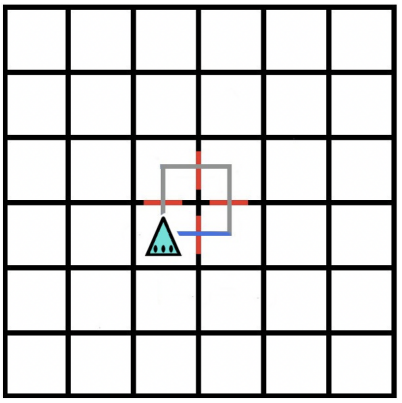


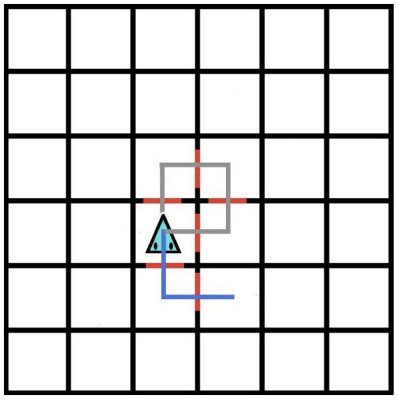
4) Uma nave pode passar pelo mesma casa no mesmo turno, mas nunca pode ir para esta mesma casa pela mesma combinação de casas partida-destino se já o fez anteriormente (no mesmo turno). Para auxiliar a compreensão, é útil imaginar cada lado de cada quadrado (que representa a sua respectiva casa) como um portão, sendo que cada portão só pode ser atravessado uma vez por turno. Ver exemplos abaixo (assume-se que cada nave não tem limite de movimento para facilitar a compreensão dos diagramas e desta regra do jogo).



A nave à esquerda viajou para Norte, Este e Sul, sendo que tornou os “portões” vermelhos à medida que os atravessa. Na posição atual, a nave pode voltar à casa por onde começou visto que o “portão” não está vermelho. Supondo que volta então à casa de partida…

A nave, na figura à direita, entrou então na casa de origem. Agora, não pode nem ir para Norte nem para Este. Consegue então ir apenas para Oeste ou Sul..





A nave à esquerda voou para Sul e depois Este. A este ponto a nave pode escolher ir para Norte, mesmo depois de já ter estado naquela casa, pois nunca passou por aquele “portão” ou por aquela combinação de casas origem-destino. Caso siga para Norte, poderá então apenas voar para Este depois.

Fim de Jogo:

Para vencer, é necessário começar o turno na *home row* e acabar na base do teu oponente numa só jogada. É permitido encadear tantos *boosts* quantos forem preciso para o jogador conseguir aterrar na base oponente sem sobrar nenhum movimento. Desta forma o jogo termina imediatamente.

Especificações extra

● Qualquer movimento para fora da grelha interior 6x6 (bases excluídas) com o intuito de aterrar na base inimiga é contabilizado como apenas um movimento.

● Caso seja impossível o movimento de qualquer uma das naves situadas na tua *home row*, o jogador escolhe a nave que se encontra na linha mais próxima dele.

● Ao fim do turno de cada jogador, o tabuleiro tem que ter mudado.

Representação interna do estado de jogo

Estado inicial:

```
startBoard([
    ['A', 'A', 'A', 'A', 'A', 'A'],
    [ 3 ,  2 ,  1 ,  3 ,  1 ,  2 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  0 ],
    [ 1 ,  3 ,  2 ,  2 ,  3 ,  1 ],
    ['B', 'B', 'B', 'B', 'B', 'B']
    ]).
```

Estado intermédio:

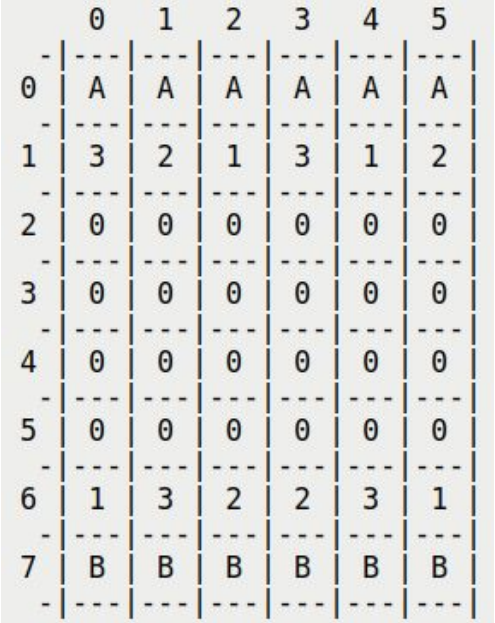
```
midBoard([
    ['A', 'A', 'A', 'A', 'A', 'A'],
    [ 0 ,  0 ,  1 ,  0 ,  0 ,  2 ],
    [ 3 ,  0 ,  0 ,  3 ,  1 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  3 ],
    [ 0 ,  0 ,  3 ,  0 ,  2 ,  0 ],
    [ 0 ,  2 ,  0 ,  0 ,  0 ,  1 ],
    [ 1 ,  0 ,  2 ,  0 ,  0 ,  0 ],
    ['B', 'B', 'B', 'B', 'B', 'B']
    ]).
```

Estado final:

```
endBoard([
    ['A', 'A',  2, 'A', 'A', 'A'],
    [ 0 ,  0 ,  1 ,  0 ,  0 ,  2 ],
    [ 3 ,  0 ,  0 ,  3 ,  1 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  0 ,  0 ],
    [ 1 ,  0 ,  3 ,  0 ,  3 ,  1 ],
    [ 0 ,  2 ,  0 ,  0 ,  0 ,  0 ],
    [ 0 ,  0 ,  0 ,  0 ,  2 ,  0 ],
    ['B', 'B', 'B', 'B', 'B', 'B']
    ]).
```

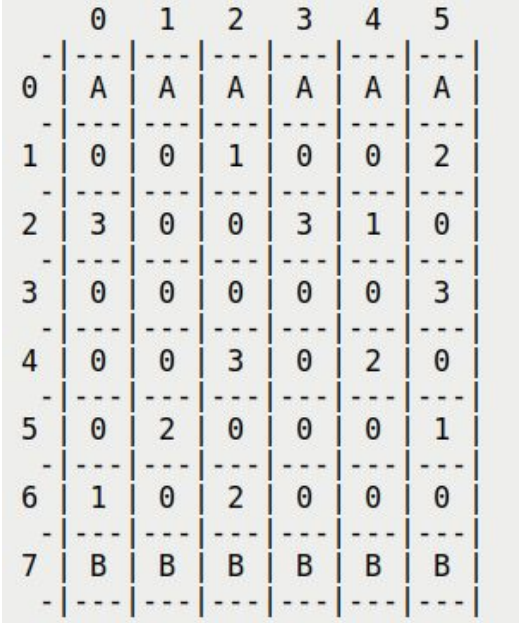
Visualização do tabuleiro em modo texto

Estado inicial:



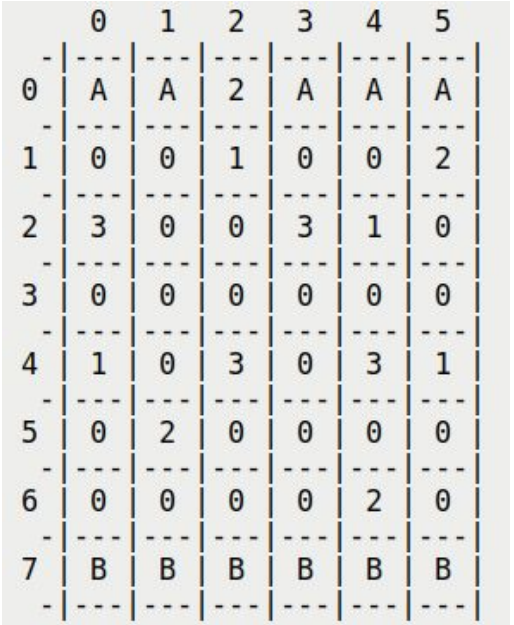
	0	1	2	3	4	5
0	A	A	A	A	A	A
1	3	2	1	3	1	2
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	1	3	2	2	3	1
7	B	B	B	B	B	B

Estado intermédio:



	0	1	2	3	4	5
0	A	A	A	A	A	A
1	0	0	1	0	0	2
2	3	0	0	3	1	0
3	0	0	0	0	0	3
4	0	0	3	0	2	0
5	0	2	0	0	0	1
6	1	0	2	0	0	0
7	B	B	B	B	B	B

Estado final:



	0	1	2	3	4	5
0	A	A	2	A	A	A
1	0	0	1	0	0	2
2	3	0	0	3	1	0
3	0	0	0	0	0	0
4	1	0	3	0	3	1
5	0	2	0	0	0	0
6	0	0	0	0	2	0
7	B	B	B	B	B	B

Predicados responsáveis pela visualização em texto

```
display_game(B, P):-
    nl,
    write('TURN: '),
    write(P),
    nl,
    write('     0   1   2   3   4   5 \n'),
    write('  -|---|---|---|---|---|---|'),
    display_matrix(B, 0).

display_matrix([], _).
display_matrix([H|T], Nrow):-
    write('\n '),
    write(Nrow),
    write(' | '),
    Nrow1 is Nrow+1,
    display_row(H, Nrow1),
    nl,
    write('  -|---|---|---|---|---|---|'),
    display_matrix(T, Nrow1).

display_row([], _).
display_row([H|T], Nrow):-
    write(H),
    write(' | '),
    display_row(T, Nrow).
```

A representação do tabuleiro começa com o predicado display_game, recebendo como argumento um estado de jogo, tabuleiro representado por 8 listas de listas de 6 elementos, e o jogador cuja vez é de jogar. Começa por imprimir o jogador que deverá fazer a próxima jogada. Depois, imprime no ecrã a numeração das colunas e, no fim, chama um outro predicado display_matrix, recebendo o estado de jogo a imprimir e o número da linha. Daqui em diante, o funcionamento é recursivo, imprimindo linha a linha com auxílio do predicado display_row. É impresso, como é visto acima, o número da linha e o seu conteúdo, sendo que ‘0’ é usado para representar uma casa vazia e os caracteres ‘1’, ‘2’ e ‘3’ usados para representar, respetivamente, naves de nível 1, nível 2 e nível 3.