Small Star Empires

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Small_Star_Empires_3:
José Aleixo Peralta da Cruz - up201403526
José Carlos Alves Vieira - up201404446

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

10 de Outubro de 2016

O Jogo Small Star Empires

História

A empresa que criou o jogo (Archona Games) recorreu ao Kickstarter para tentar fazer com que o mesmo fosse possível, e conseguiu, adquirindo fundos suficientes para levar o projecto avante.

4 das maiores civilizações da galáxia adquiriram a habilidade de viajar à velocidade da luz, e estão prontas para impôr o seu domínio pelo vasto oceano de estrelas!

A civilização dos jogadores é uma dessas 4, e é o seu dever fazer com que a mesma seja a mais poderosa e com mais domínios.

Esse domínio será adquirido pelas viagens por entre planetas, sistemas e nebulosas com as naves da civilização.

Será preciso poder estratégico, portanto prepara-te para seres o império mais poderoso da galáxia!

Objectivo

Este é um jogo para 2 a 4 jogadores. O objetivo é colonizar a galáxia usando as naves, que serão movidas num tabuleiro hexagonal, sendo cada hexágono um sistema.

A colonização é realizada através da colocação de colónias ou de estações de comércio, que equivale ao controlo dentro daquele sistema.

No final do jogo, os jogadores calculam os pontos de cada um, dependendo esses da quantidade de sistemas que cada um controla.

O jogador com mais pontos é anunciado vencedor!

Regras

Tipos de Sistemas:

1. Sistemas Mãe: cada jogador começa no seu sistema mãe. Inicialmente, estes sistemas já estão ocupados pelo jogador correspondente, não sendo necessário colonizá-lo nem construir uma estação de comércio. No entanto, não dão pontos ao jogador que o controla no final da partida, mas contam como bónus de território e para os jogadores que tiverem algo adjacente a este sistema.
2. Sistemas de Estrelas: são colonizáveis. Cada sistema tem entre 1-3 planetas e oferecem 1-3 pontos no final da partida, dependendo do número de planetas.
3. Sistemas Nebulosos: quantos mais destes sistemas da mesma cor forem controlados pelo mesmo jogador, mais pontos esse jogador recebe no final da partida. 1 sistema equivale a 2 pontos, 2 sistemas a 5 pontos e 3 sistemas a 8 pontos.

4. Sistemas Vazios: são colonizáveis. Contam como bónus de território caso estejam colonizados, , mas não fornecem pontos no final da partida, exceto se tiver sido construída uma estação de comércio e houver sistemas controlados pelos oponentes adjacentes a este.
5. Buraco da Minhoca: podem ser usados para rapidamente viajar para diferentes cantos da galáxia.
6. Buracos Negros: são muitos perigosos, razão suficiente para que as naves não possam passar por cima nem mover-se para dentro de buracos negros.

Componentes dos jogadores:

1. Naves: cada jogador usa 2-4 naves. Podem sobrevoar sistemas controlados pelo jogador correspondente, mas buracos negros e sistemas controlados por outros jogadores impedem as naves de se deslocarem desse ponto adiante.
2. Colónias: usadas para marcar controlo por um jogador sobre um sistema.
3. Estações de Comércio: igual às colónias, mas no final da partida podem dar pontos bónus.

Flow da partida:

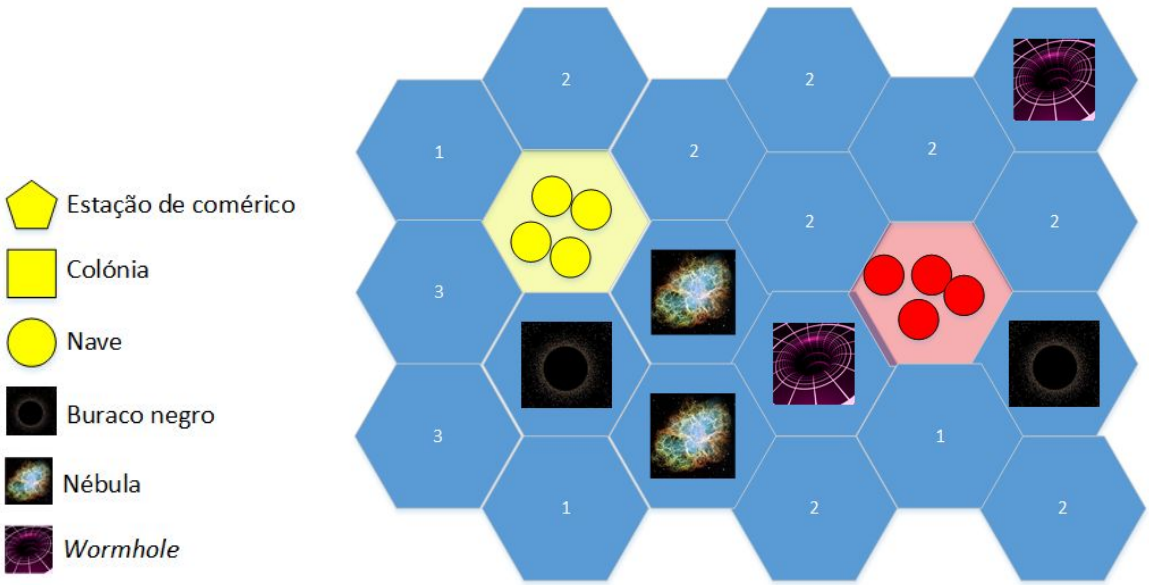
1. Escolha da cor do jogador (vermelho, verde, azul ou amarelo).
2. Posicionamento das suas naves no correspondente sistema mãe.
3. Se algum dos jogadores tiver vencido a ronda anterior, é esse mesmo que começa a partida. Se ainda não tiver sido feito nenhuma partida, o jogador que começa é o mais novo.
4. Sistema de funcionamento por turnos, avançando de jogador para jogador usando o método dos ponteiros do relógio (sentido horário).
5. O jogador move uma das suas naves, e onde pousar tem de construir uma colônia ou uma estação de comércio, ficando com o controlo do sistema. É proibido alterar o que já está construído numa fase posterior do jogo.
6. Se algum jogador não conseguir mover pelo menos uma das suas naves, o turno passa para o próximo jogador.
7. Realiza-se este ciclo até que mais nenhuma nave possa viajar ou todos os sistemas estiverem ocupados.
8. A partida acaba quando todos os sistemas estiverem ocupados ou todas as naves ficaram impossibilitadas de viajar.
9. É realizado o cálculo dos pontos:
    a. Sistemas de Estrelas: 1 ponto por cada planeta no sistema.
    b. Sistemas Nebulosos: 2 pontos por 1 nebulosa de uma cor, 5 pontos por 2 nebulosas da mesma cor, e 8 pontos por 3 nebulosas da mesma cor.
    c. Estações de comércio: 1 ponto por cada estação de comércio, colónia ou sistema mãe de outro jogador adjacente a esta estação.
    d. Pontos bônus: 3 pontos para o jogador com o maior território (o sistema mais conectado num único território).
10. No caso de empates (ordem de desempates):

a. jogador com mais colónias por colonizar.
b. jogador com mais estações de comércio por construir.
c. jogador com mais planetas controlados.
d. todos os jogadores empatados.

Representação do Estado do Jogo

O tabuleiro de jogo é representado internamente como uma lista de listas, denominada *board*. Cada elemento do *board*, será uma lista que representa uma linha do tabuleiro de jogo. Cada elemento de uma linha será outra lista, contendo toda a informação de uma casa do tabuleiro.

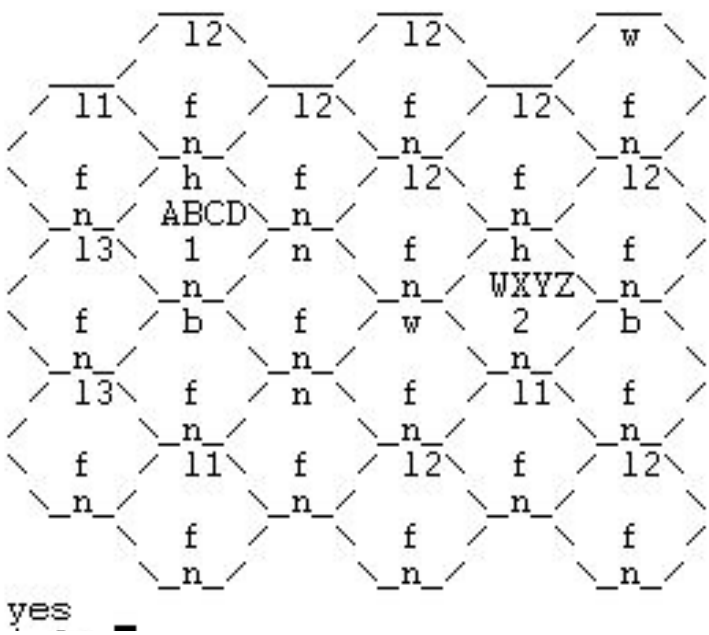
Um exemplo da composição do tabuleiro para o início de um jogo é a seguinte, em que todas as naves de cada jogador começam nos respetivos sistemas-mãe:



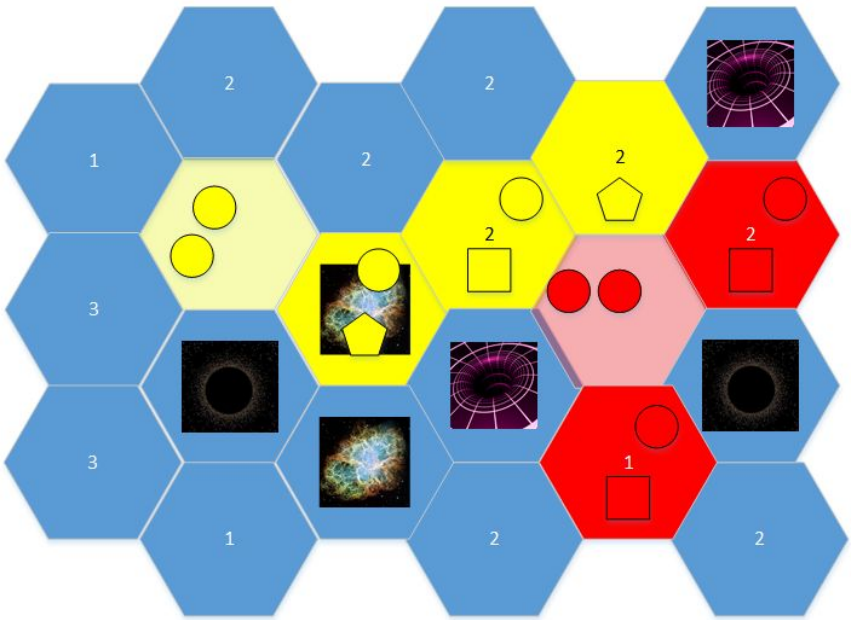
Legenda das casas e resultado em Prolog

```
%%% Case example: [<type>, <owner>, [<listOfShips], <building>]
/*
TYPES OF CASE                          2 - player two
h - home system                        f - free
b - blackhole
w - wormhole                         LIST OF SHIPS
l1 - level one system (1               Player one: ABCD or abcd
point)                               (damaged)
l2 - level two system (2               Player two: WXYZ or wxyz
points)                              (damaged)
l3 - level three system (3
points)
n - nebula (5 points)                  BUILDING
                                       t - trade station
OWNER                                  c - colony
1 - player one                         n - none
```

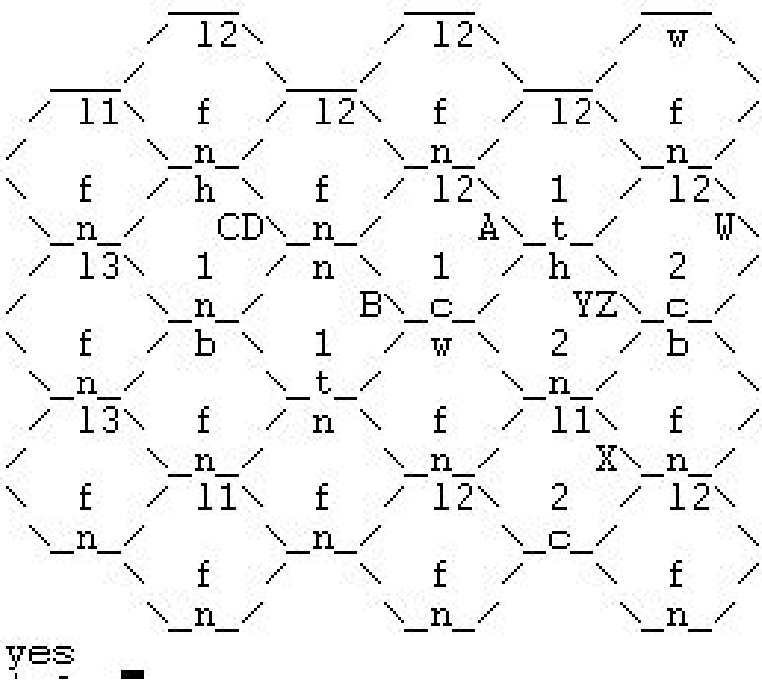
*/



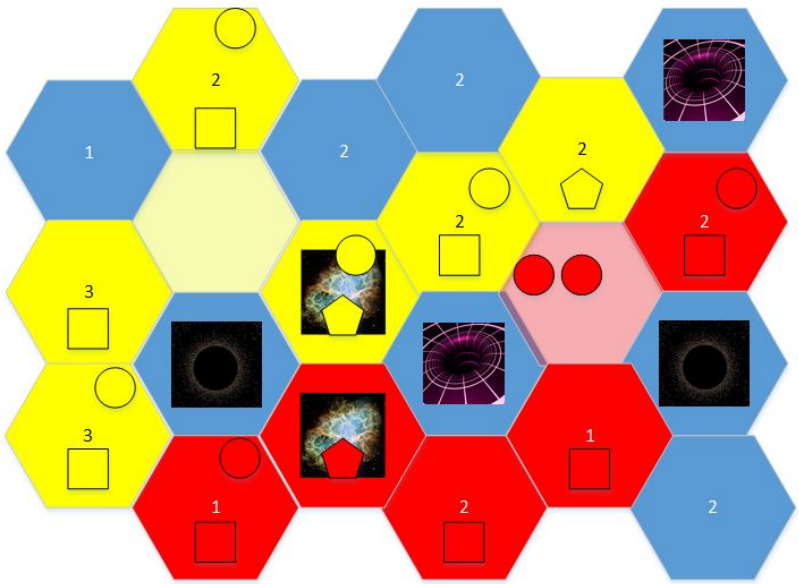
De seguida mostra-se uma situação para a qual o jogo poderá desenvolver-se, começando o amarelo primeiro. À medida que as naves avançam e conquistam sistemas e nébulas, constroem-se colónias ou estações de comércio.



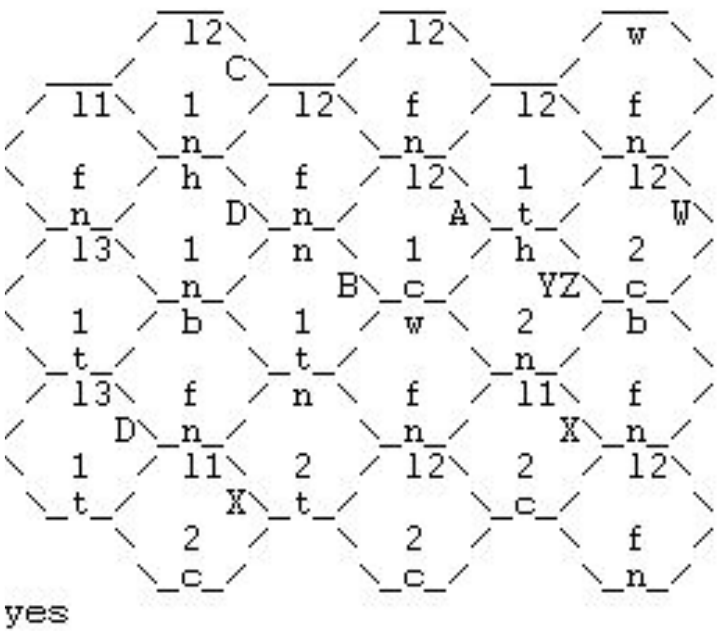
Resultado em Prolog:



Uma possível situação de fim de jogo resulta da falta de opções de jogada pelo jogador vermelho, que não consegue movimentar nenhuma das suas naves, enquanto o jogador amarelo consegue. Assim, perde o jogador vermelho. Na imagem de exemplo, as naves ainda estão presentes para mostrar as suas últimas posições:



Resultado em Prolog:



yes

Visualização do Tabuleiro

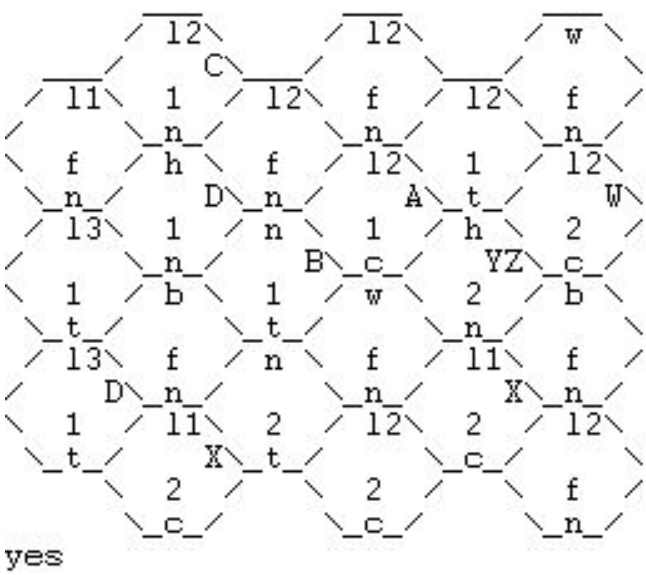
Para visualizar o tabuleiro, são geradas formas hexagonais recursivamente, que representam cada casa do tabuleiro. É possível determinar a dimensão do tabuleiro e ele gera-se automaticamente. Assim, a dimensão do nosso tabuleiro depende da dimensão da lista de listas que representa o jogo.

Se o número de elementos da lista *board* for 3, o tabuleiro terá 3 linhas. O número de colunas é determinado pelo número de elementos que uma linha tem.

A informação de cada casa é declarada na matriz de representação de jogo como uma lista dos diversos elementos que uma casa pode conter e é escrita recursiva e automaticamente a partir da matriz, preservando o espaçamento entre hexágonos. Caso uma haja um elemento do tipo vazio (‘x’), a coordenada correspondente não vai conter informação.

A visualização do tabuleiro é feita recorrendo ao predicado `?- display`. Segue-se um exemplo de lista de listas que representa o estado de jogo e respetiva visualização:

```
board([
    [[12, 1, ['C'], n], [12, f, [], n], [w, f, [], n]],
    [[11, f, [], n], [12, f, [], n], [12, 1, [], t]],
    [[h, 1, ['D'], n], [12, 1, ['A'], c], [12, 2, ['W'], c]],
    [[13, 1, [], t], [n, 1, ['B'], t], [h, 2, ['Y', 'Z'], n]],
    [[b, f, [], n], [w, f, [], n], [b, f, [], n]],
    [[13, 1, ['D'], t], [n, 2, [], t], [11, 2, [], c]],
    [[11, 2, ['X'], c], [12, 2, [], c], [12, f, [], n]]
    ]).
```



Movimentos

No Small Star Empires só existe um tipo de peça que o jogador movimenta pelo tabuleiro: a nave. Esta pode viajar um qualquer número de casas, mas apenas numa das 6 direções possíveis, a partir da casa onde está.

Uma nave **pode:**

- mover-se para um sistema desocupado;
- passar por um sistema que tenha colonizado;
- passar por um *wormhole*.

Uma nave **não pode**:

- mover-se ou passar por um buraco negro;
- mover-se para um *wormhole*;
- mover-se para ou passar por um sistema controlado por um adversário.
- mover-se para um dos seus sistemas.

Há ainda o caso particular de, se a nave tiver passado por um campo de asteróides, tornar-se danificada e poder apenas mover-se no máximo 2 casas por jogada.

O predicado para mover uma nave de um jogador seria:

```
move(-player, -ship, -direction, -numOfCases, -oldBoard, +newBoard).
/* player: 1 ou 2, ship: A,B,C ou D, direction: N, S, NE, SE, NW ou SW */
```

Este predicado chamaria outros predicados, para obter a informação do tabuleiro e verificar se a jogada efetuada é válida:

```
getStartCase(-player, -ship, -board, +startCase).
getDestinyCase(-startCase, -direction, -numOfCases, +destinyCase).

destinyIsValid(-player, -ship, -board, -destinyCase).
caseIsBlackHole(-board, -case).
caseIsWormHole(-board, -case).
caseIsFree(-board, -case).
caseBelongsToOpponent(-player, -board, -case).
caseBelongsToSelf(-player, -board, -case).
shipIsDamaged(-player, -ship).
```

Ao chegar a um novo sistema, o dono da nave tem de a ocupar com uma colónia ou com uma estação de comércio.

```
buildColony(-player, -case, -oldBoard, +newBoard).
buildTradeStation(-player, -case, -oldBoard, +newBoard).
```

Anexo

Código-fonte:

```
%%% Case example: [<type>, <owner>, [<listOfShips], <building>]

/* TYPES OF CASE
h - home system
b - blackhole
w - wormhole
l1 - level one system (1 point)
l2 - level two system (2 points)
l3 - level three system (3 points)
n - nebula (5 points)
*/

/* OWNER
1 - player one
2 - player two
f - free
*/

/* LIST OF SHIPS
Player one: ABCD or abcd (damaged)
Player two: WXYZ or wxyz (damaged)
*/

/* BUILDING
t - trade station
c - colony
n - none
*/

translate(s10, '          ').
translate(s8, '        ').
translate(s7, '       ').
translate(s6, '      ').
translate(s5, '     ').
translate(s3, '   ').
translate(s2, '  ').
translate(s1, ' ').
translate(s0, '').
translate(t1, '_').
translate(t2, '__').
translate(t3, '___').
translate(ub, '/').
translate(db, '\\').
translate(nl, '\n').
translate(c, '1').
translate(x, '').

/* UTILITIES */

prefix([],Ys).
prefix([X|Xs], [X|Ys]):- prefix(Xs,Ys).

sufix(Xs, Xs).
sufix(Xs, [Y|Ys]):- suffix(Xs, Ys).
```

```
sublist(Xs, Ys):- prefix(Xs, Ys).
sublist(Xs, [Y|Ys]):- sublist(Xs,Ys).

length([],0);
length([X|Xs], s(N)):- length(Xs, N).

first(X, [X|_]).

last(X, [X]).
last(X, [_|Z]) :- last(X, Z).

/* END OF UTILITIES */


/* DISPLAY GAME CASE FUNCTIONS*/

%Verify errors example function

isTypeOfCase(C):-
    C == h;
    C == b;
    C == c;
    C == l1;
    C == l2;
    C == l3;
    C == n.

case_example([h, 1, ['A', 'B', 'C', 'D'], n]).

board_example([
    [h, 1, ['A', 'B', 'C', 'D'], n], x, [h, 2, ['W', 'X', 'Y', 'Z'], n]
]).

display_list([]).

display_list([E1|Es]):-
    write(E1),
    display_list(Es).

getLine(X, [X|_], 0).

getLine(X, [_|L], LineToSend):-
    getLine(X, L, K1), LineToSend is K1 + 1.

/* These next functions add space to fill the hexagon awhile displaying information */

display_type(X):-
    translate(X, A),
    translate(s3, SpaceBetweenHex),
    write(SpaceBetweenHex).

display_type([C,_,_,_|_]):-
    (C == 'h'; C == 'b'; C == 'n'; C == 'w') ->
        translate(s1, S),
        write(S),
        write(C),
        write(S);
    (C == 'l1'; C == 'l2'; C == 'l3') ->
        translate(s1, S),
        write(S),
        write(C).
```

```
display_owner(X):-
    translate(X, A),
    translate(s5, SpaceBetweenHex),
    write(SpaceBetweenHex).

display_owner([_,O,_,_|_]):-
    translate(s2, SpaceBetweenHex),
    write(SpaceBetweenHex),
    write(O),
    write(SpaceBetweenHex).

display_ships(X):-
    translate(X, A),
    translate(s5, SpaceBetweenHex),
    write(SpaceBetweenHex).

display_ships([_,_,S,_|_]):-
    length(S, L),
    L1 is 5-L,
    generate_empty_space(s1, L1),
    display_list(S).

display_building(X):-
    translate(X, A),
    translate(t2, SpaceBetweenHex),
    write(SpaceBetweenHex).

display_building([_,_,_,B|_]):-
    translate(t1, S),
    write(B),
    write(S).

empty_case(C):-
    C==x.

display_board_case([]).

display_board_case(C):-
    display_type(C),
    display_owner(C),
    display_ships(C),
    display_building(C).

/*** WRITE FUNCTIONS ***/

/* Write whole element */
write_element_coord([B1|Bs], X, Y):-
    Y == 0 -> write_element(B1, X);
    Y > 0 -> Y1 is Y-1, write_element_coord(Bs, X, Y1).

write_element([R1|Rs], X):-
    X == 0 -> display_board_case(R1);
    X > 0 -> X1 is X-1, write_element(Rs, X1).

/* Write element type */
write_type_coord([B1|Bs], X, Y):-
    Y == 0 -> write_element_type(B1, X);
    Y > 0 -> Y1 is Y-1, write_type_coord(Bs, X, Y1).

write_element_type([R1|Rs], X):-
    X == 0 -> display_type(R1);
    X > 0 -> X1 is X-1, write_element_type(Rs, X1).
```

```
/* Write element owner */

write_owner_coord([B1|Bs], X, Y):-
    Y == 0 -> write_element_owner(B1, X);
    Y > 0 -> Y1 is Y-1, write_owner_coord(Bs, X, Y1).

write_element_owner([R1|Rs], X):-
    X == 0 -> display_owner(R1);
    X > 0 -> X1 is X-1, write_element_owner(Rs, X1).

/* Write element ships*/

write_ships_coord([B1|Bs], X, Y):-
    Y == 0 -> write_element_ships(B1, X);
    Y > 0 -> Y1 is Y-1, write_ships_coord(Bs, X, Y1).

write_element_ships([R1|Rs], X):-
    X == 0 -> display_ships(R1);
    X > 0 -> X1 is X-1, write_element_ships(Rs, X1).

/* Write element building */

write_building_coord([B1|Bs], X, Y):-
    Y == 0 -> write_element_building(B1, X);
    Y > 0 -> Y1 is Y-1, write_building_coord(Bs, X, Y1).

write_element_building([R1|Rs], X):-
    X == 0 -> display_building(R1);
    X > 0 -> X1 is X-1, write_element_building(Rs, X1).

/*** END OF WRITE FUNCTIONS ***/

display_board_test([B1|Bs]):-
    display_board_case(B1),
    display_board_test(Bs).

test_display_board:-
    board_example(B), display_board_test(B).

generate_empty_space(Spaces, NumberOfTimes):-
    NumberOfTimes = 0,
    write('').

generate_empty_space(Spaces, NumberOfTimes):-
    N1 is NumberOfTimes - 1,
    translate(Spaces, EmptySpace),
    write(EmptySpace),
    generate_empty_space(Spaces, N1).

display_line_1_aux(NumberHexagons):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons - 1,
        translate(t3, A),
        translate(s7, SpaceBetweenHex),
        write(A),
        write(SpaceBetweenHex),
        display_line_1_aux(N1);
    NumberHexagons == 0 -> nl.

display_line_1(NumberEmptySpaces, NumberHexagons):-
    generate_empty_space(s7, 1),
```

```
    generate_empty_space(s10, NumberEmptySpaces),
    display_line_1_aux(NumberHexagons).

getCurrentPiece([X|Xs], X, Xs).

display_line_2_aux(NumberHexagons, FirstRow):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons-1,
        getCurrentPiece(FirstRow, CurrentPiece, RemainingPieces),
        translate(ub, OpenHex),
        translate(s3, SpaceInsideHex),
        translate(db, CloseHex),
        translate(s5, SpaceBetweenHex),
        write(OpenHex),
        display_type(CurrentPiece),
        write(CloseHex),
        write(SpaceBetweenHex),
        display_line_2_aux(N1, RemainingPieces);
    NumberHexagons == 0 -> nl.

display_line_2(NumberEmptySpaces, NumberHexagons, FirstRow):-
    generate_empty_space(s6, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    display_line_2_aux(NumberHexagons, FirstRow).

display_line_3_aux(NumberHexagons, FirstRow):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons-1,
        getCurrentPiece(FirstRow, CurrentPiece, RemainingPieces),
        translate(t3, A),
        translate(ub, OpenHex),
        translate(s5, SpaceInsideHex),
        translate(db, CloseHex),
        write(A),
        write(OpenHex),
        display_ships(CurrentPiece),
        write(CloseHex),
        display_line_3_aux(N1, RemainingPieces);
    NumberHexagons == 0 -> nl.

display_line_3(NumberEmptySpaces, NumberHexagons, FirstRow):-
    generate_empty_space(s2, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    display_line_3_aux(NumberHexagons, FirstRow).

display_line_4_aux(NumberHexagons, UpperMatrixLine, MiddleMatrixLine):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons - 1,
        getCurrentPiece(UpperMatrixLine, CurrentUpperPiece, RemainingUpperPieces),
        getCurrentPiece(MiddleMatrixLine, CurrentMiddlePiece, RemainingMiddlePieces),
        translate(ub, OpenHex),
        translate(s3, SpaceInsideHex),
        translate(db, CloseHex),
        translate(s5, SpaceInsideHex2),
        display_type(CurrentMiddlePiece),
        write(CloseHex),
        display_owner(CurrentUpperPiece),
        write(OpenHex),
        display_line_4_aux(N1, RemainingUpperPieces, RemainingMiddlePieces);
    NumberHexagons == 0 -> nl.

display_line_4(NumberEmptySpaces, NumberHexagons, UpperMatrixLine, MiddleMatrixLine):-
```

```
    generate_empty_space(s1, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    translate(ub, OpenHex),
    translate(s3, SpaceInsideHex),
    translate(db, CloseHex),
    write(OpenHex),
    display_line_4_aux(NumberHexagons, UpperMatrixLine, MiddleMatrixLine).

display_line_5_aux(NumberHexagons, UpperMatrixLine, MiddleMatrixLine):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons - 1,
        getCurrentPiece(UpperMatrixLine, CurrentUpperPiece, RemainingUpperPieces),
        getCurrentPiece(MiddleMatrixLine, CurrentMiddlePiece, RemainingMiddlePieces),
        translate(t1, A),
        translate(ub, OpenHex),
        translate(s5, SpaceInsideHex),
        translate(db, CloseHex),
        display_ships(CurrentMiddlePiece),
        write(CloseHex),
        write(A),
        display_building(CurrentUpperPiece),
        write(OpenHex),
        display_line_5_aux(N1, RemainingUpperPieces, RemainingMiddlePieces);
    NumberHexagons == 0 -> nl.

display_line_5(NumberEmptySpaces, NumberHexagons, UpperMatrixLine, MiddleMatrixLine):-
    generate_empty_space(s0, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    translate(ub, OpenHex),
    translate(s5, SpaceInsideHex),
    translate(db, CloseHex),
    write(OpenHex),
    display_line_5_aux(NumberHexagons, UpperMatrixLine, MiddleMatrixLine).

display_line_6_aux(NumberHexagons, MiddleMatrixLine, LowerMatrixLine):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons - 1,
        getCurrentPiece(MiddleMatrixLine, CurrentMiddlePiece, RemainingMiddlePieces),
        getCurrentPiece(LowerMatrixLine, CurrentLowerPiece, RemainingLowerPiece),
        translate(ub, OpenHex),
        translate(s3, SpaceInsideHex),
        translate(db, CloseHex),
        translate(s5, SpaceInsideHex2),
        display_owner(CurrentMiddlePiece),
        write(OpenHex),
        display_type(CurrentLowerPiece),
        write(CloseHex),
        display_line_6_aux(N1, RemainingMiddlePieces, RemainingLowerPiece);
    NumberHexagons == 0 -> nl.

display_line_6(NumberEmptySpaces, NumberHexagons, MiddleMatrixLine, LowerMatrixLine):-
    generate_empty_space(s0, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    translate(ub, OpenHex),
    translate(s5, SpaceInsideHex),
    translate(db, CloseHex),
    translate(s5, SpaceInsideHex2),
    write(CloseHex),
    display_line_6_aux(NumberHexagons, MiddleMatrixLine, LowerMatrixLine).

display_line_7_aux(NumberHexagons, MiddleMatrixLine, LowerMatrixLine):-
    NumberHexagons > 0 ->
```

```
        N1 is NumberHexagons - 1,
        getCurrentPiece(MiddleMatrixLine, CurrentMiddlePiece, RemainingMiddlePieces),
        getCurrentPiece(LowerMatrixLine, CurrentLowerPiece, RemainingLowerPiece),
        translate(t1, A),
        translate(ub, OpenHex),
        translate(s5, SpaceInsideHex),
        translate(db, CloseHex),
        write(A),
        display_building(CurrentMiddlePiece),
        write(OpenHex),
        display_ships(CurrentLowerPiece),
        write(CloseHex),
        display_line_7_aux(N1, RemainingMiddlePieces, RemainingLowerPiece);
    NumberHexagons == 0 -> nl.

display_line_7(NumberEmptySpaces, NumberHexagons, MiddleMatrixLine, LowerMatrixLine):-
    generate_empty_space(s1, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    translate(db, CloseHex),
    write(CloseHex),
    display_line_7_aux(NumberHexagons, MiddleMatrixLine, LowerMatrixLine).

display_line_8_aux(NumberHexagons, LastRow):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons-1,
        getCurrentPiece(LastRow, CurrentPiece, RemainingPieces),
        translate(ub, OpenHex),
        translate(s5, SpaceInsideHex),
        translate(db, CloseHex),
        translate(s3, SpaceInsideHex2),
        write(CloseHex),
        display_owner(CurrentPiece),
        write(OpenHex),
        write(SpaceInsideHex2),
        display_line_8_aux(N1, RemainingPieces);
    NumberHexagons == 0 -> nl.

display_line_8(NumberEmptySpaces, NumberHexagons, LastRow):-
    generate_empty_space(s5, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    display_line_8_aux(NumberHexagons, LastRow).

display_line_9_aux(NumberHexagons, LastRow):-
    NumberHexagons > 0 ->
        N1 is NumberHexagons-1,
        getCurrentPiece(LastRow, CurrentPiece, RemainingPieces),
        translate(t1, A),
        translate(ub, OpenHex),
        translate(db, CloseHex),
        translate(s5, SpaceBetweenHex),
        write(CloseHex),
        write(A),
        display_building(CurrentPiece),
        write(OpenHex),
        write(SpaceBetweenHex),
        display_line_9_aux(N1, RemainingPieces);
    NumberHexagons == 0 -> nl.

display_line_9(NumberEmptySpaces, NumberHexagons, LastRow):-
    generate_empty_space(s6, 1),
    generate_empty_space(s10, NumberEmptySpaces),
    display_line_9_aux(NumberHexagons, LastRow).
```

```
display_num_linhas(NumLinhasAdicionais, NumOfCols, MatrixLineToStart, Board):-
    NumLinhasAdicionais > 0 ->
        N1 is NumLinhasAdicionais - 1,
        HexMiddleLine is MatrixLineToStart+1,
        HexLowerLine is MatrixLineToStart+2,
        getLine(UpperMatrixLine, Board, MatrixLineToStart),
        getLine(MiddleMatrixLine, Board, HexMiddleLine),
        getLine(LowerMatrixLine, Board, HexLowerLine),
        display_line_4(0, NumOfCols, UpperMatrixLine, MiddleMatrixLine),
        display_line_5(0, NumOfCols, UpperMatrixLine, MiddleMatrixLine),
        display_line_6(0, NumOfCols, MiddleMatrixLine, LowerMatrixLine),
        display_line_7(0, NumOfCols, MiddleMatrixLine, LowerMatrixLine),
        display_num_linhas(N1, NumOfCols, HexLowerLine, Board);
    NumLinhasAdicionais == 0.

display_start_lines(NumOfCols, FirstRow):-
    display_line_1(0, NumOfCols),
    display_line_2(0, NumOfCols, FirstRow),
    display_line_3(0, NumOfCols, FirstRow).

display_end_lines(NumOfCols, LastRow):-
    display_line_8(0, NumOfCols, LastRow),
    display_line_9(0, NumOfCols, LastRow).

display_board:-
    board(B),
    length(B, NumOfRows),

    1 is mod(NumOfRows, 2), /**** Until we can work with even rows ****/

    first(FirstRow, B),
    length(FirstRow, NumOfElementsFirstRow),

    last(LastRow, B),
    length(LastRow, NumOfElementsLastRow),

    display_start_lines(NumOfElementsFirstRow, FirstRow),
    display_num_linhas(NumOfRows//2 , NumOfElementsFirstRow, 0, B),
    display_end_lines(NumOfElementsLastRow, LastRow).

display:- display_board.

/**********
*  BOARD  *
***********/

/* Each element of the board is a line. Each element within the line is a piece.*/
initial_board([
    [[l2, f, [], n], [l2, f, [], n], [w, f, [], n]],
    [[l1, f, [], n], [l2, f, [], n], [l2, f, [], n]],
    [[h, 1, ['A', 'B', 'C', 'D'], n], [l2, f, [], n], [l2, f, [], n]],
    [[l3, f, [], n], [n, f, [], n], [h, 2, ['W', 'X', 'Y', 'Z'], n]],
    [[b, f, [], n], [w, f, [], n], [b, f, [], n]],
    [[l3, f, [], n], [n, f, [], n], [l1, f, [], n]],
    [[l1, f, [], n], [l2, f, [], n], [l2, f, [], n]]
    ]
    ).

mid_board([
    [[l2, f, [], n], [l2, f, [], n], [w, f, [], n]],
    [[l1, f, [], n], [l2, f, [], n], [l2, 1, [], t]],
```

```
    [[h, 1, ['C', 'D'], n], [l2, 1, ['A'], c], [l2, 2, ['W'], c]],
    [[l3, f, [], n], [n, 1, ['B'], t], [h, 2, ['Y', 'Z'], n]],
    [[b, f, [], n], [w, f, [], n], [b, f, [], n]],
    [[l3, f, [], n], [n, f, [], n], [l1, 2, ['X'], c]],
    [[l1, f, [], n], [l2, f, [], n], [l2, f, [], n]]
    ]
    ).

board([
    [[l2, 1, ['C'], n], [l2, f, [], n], [w, f, [], n]],
    [[l1, f, [], n], [l2, f, [], n], [l2, 1, [], t]],
    [[h, 1, ['D'], n], [l2, 1, ['A'], c], [l2, 2, ['W'], c]],
    [[l3, 1, [], t], [n, 1, ['B'], t], [h, 2, ['Y', 'Z'], n]],
    [[b, f, [], n], [w, f, [], n], [b, f, [], n]],
    [[l3, 1, ['D'], t], [n, 2, [], t], [l1, 2, [], c]],
    [[l1, 2, ['X'], c], [l2, 2, [], c], [l2, f, [], n]]
    ]
    ).
```