FEUP 2019/2020

PLOG

SQUEX

TP 1

Intermediary Report

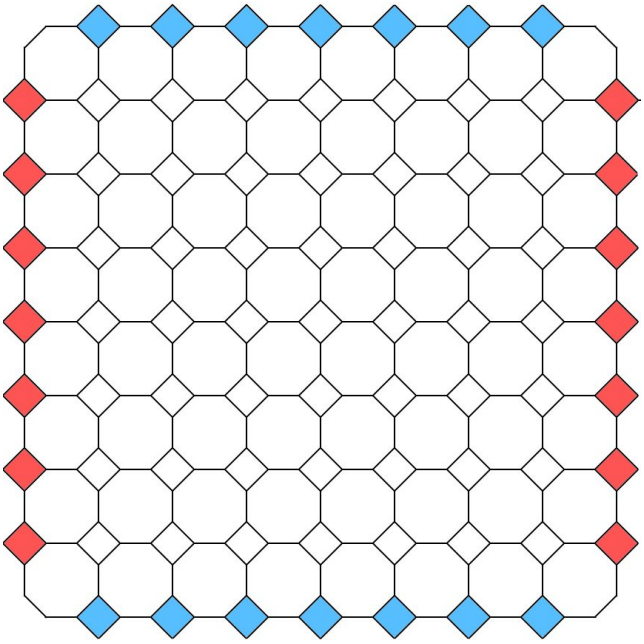
David Luís Dias da Silva
up201705373@fe.up.pt

Luís Pedro Pereira Lopes Mascarenhas Cunha
up201706736@fe.up.pt

1. Game description

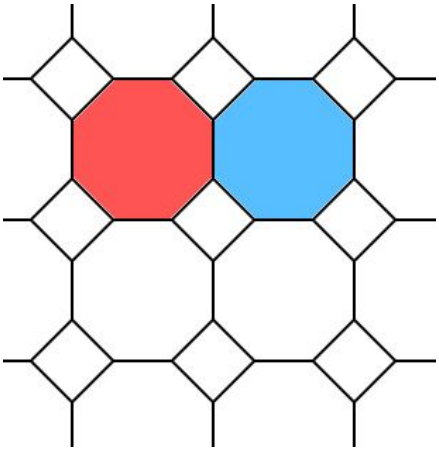
The game Squex (Square Hex) is a two-player board game, similar to Hex, that is played on a 8x8 square board(or any other size) of octagons connected by squares, as seen in Fig. 1.

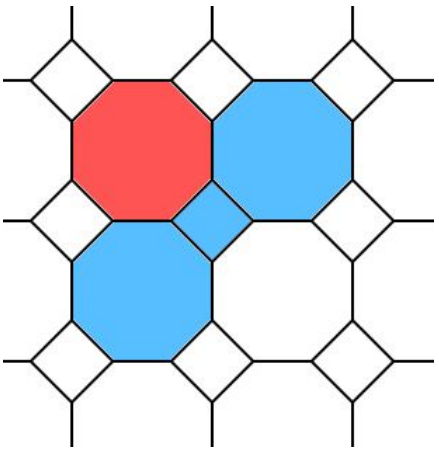
Opposite edges of the board are colored the same. Each player chooses a color and plays by placing an octagonal piece in the board aiming to connect the corresponding edges with a continuous line of octagons and squares.



2. Rules

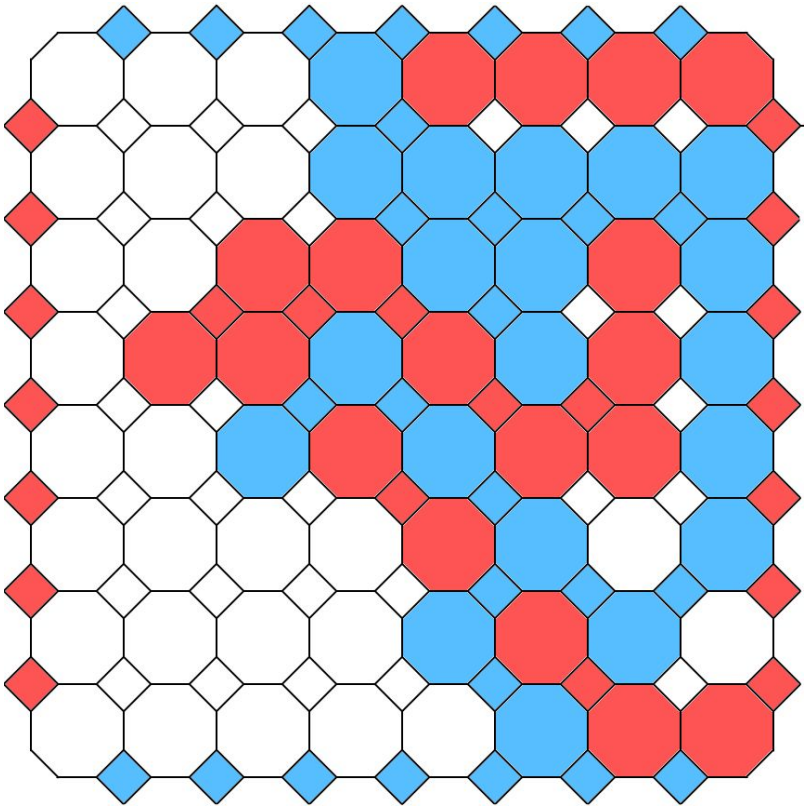
1. The players play by turns, placing a piece of the designed color in any of the free octagons.
2. When a piece is placed diagonally to another of the same player, a square piece of the corresponding color is placed in the square in between, as seen below.





3. A cut occurs when rule 2 happens and a square of the other player is already occupying that position, the currently placed piece is replaced by one of the player who just played.

4. In the event of a cut, the player who suffered it gets two consecutive turns. This is true also if a player performs a cut in the first turn after suffering one, in which the player loses its second turn and the other player gets to play twice.

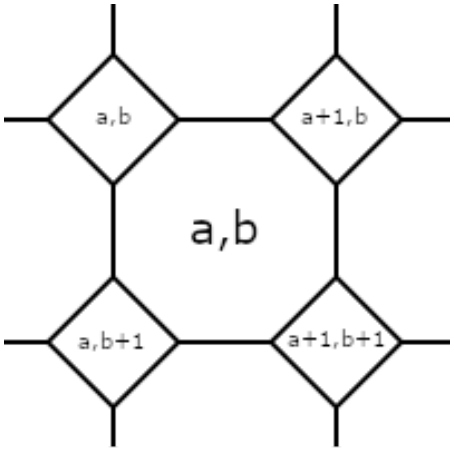


5. A player wins only if it was able to establish a connection between the respective color's sides of the board that can't be broken with a cut from the opposite player. An example of this is shown in the above figure, in which the blue player still hasn't won the game, because the red player can cut the blue square in the bottom.

3. Internal representation

For the internal representation of the game, namely the board, we considered two approaches (both using lists of lists). An approach where the state of the board was stored in two matrices, one for the octagons and one for the squares, and another where all the cells were stored in a single matrix. The latter would consist of a matrix with dimensions of 17 x 9 (storing the squares and octagons in alternating rows) or 8x8 (each cell stores data about an octagon and surrounding squares). The reasoning behind this strategy was that it would be easier to display the board if it was stored in one matrix. However we chose the strategy of using two matrices, one 8x8 that holds the information about the octagon cells and one 9x9 that stores the state of the squares, which also proved simple display the board.

This decision was made because of the modularity that is provided by separating the different concepts and because of the simplicity of calculating the adjacent squares to a given octagon. In fact, for the octagon with the indices (a, b) in the octagon matrix, the adjacent squares have the indices (a, b), (a + 1, b), (a + 1, b + 1) and (a, b + 1), counting clockwise starting at the top-left square.



Each cell of the board matrices stores an integer that signifies the type of piece that occupies it in the board at a given moment. The value 0 corresponds to an empty cell, the values 1 and 2 to cells with pieces played by player 1 and player 2, respectively.

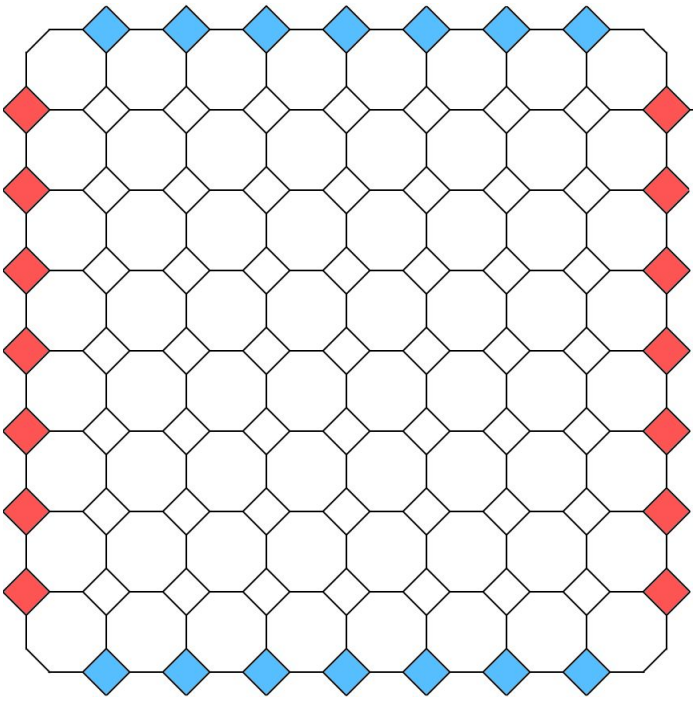
```
init_board([[ %octagon cells
              [0, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 2, 0, 0, 0, 0, 0],
              [0, 1, 2, 0, 1, 0, 0, 0],
              [0, 2, 0, 0, 2, 0, 0, 0],
              [0, 0, 1, 0, 1, 0, 0, 0],
              [0, 0, 0, 0, 1, 0, 2, 0],
              [0, 0, 0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 0, 0, 0]
             ],

             [ %square cells
              [0, 1, 1, 1, 1, 1, 1, 0],
              [2, 0, 0, 0, 0, 0, 0, 2],
              [2, 0, 0, 0, 0, 1, 0, 2],
              [2, 0, 1, 0, 1, 0, 2, 2],
              [2, 0, 0, 0, 2, 0, 0, 2],
              [2, 0, 2, 0, 0, 0, 0, 2],
              [2, 0, 0, 1, 0, 0, 0, 2],
              [2, 0, 0, 0, 0, 0, 0, 2],
              [0, 1, 1, 1, 1, 1, 1, 0]
             ]]).
```

Another important aspect of the state of the game is whose turn is next and for how many consecutive turns the player will play. This is necessary because of rule 4.

3.1 The board at different stages of the game

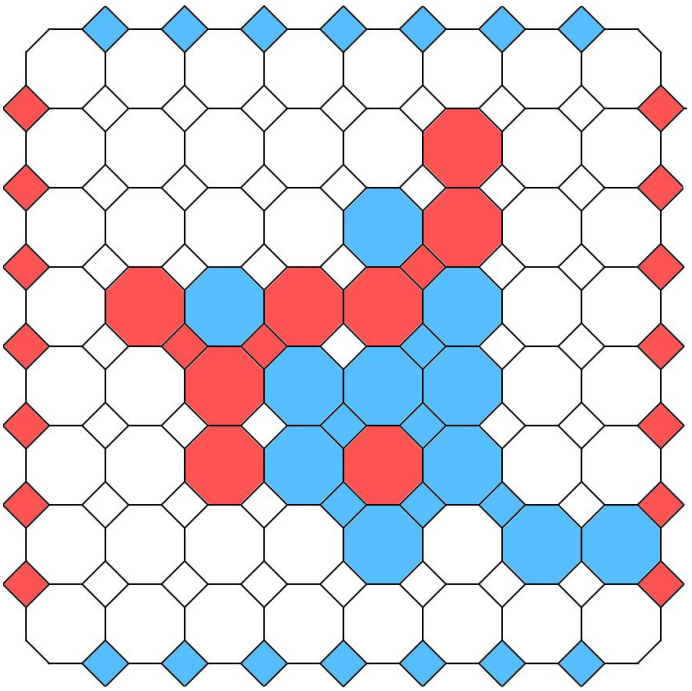
Initially the the octagon-matrix is empty and the square-matrix has the “blue-values” for the first and last row, and the “red-values” for the first and last column. It is important to note that the values of the first and last rows/columns of the square matrix don’t change throughout the game as per the game rules. Also, the corners of the matrix aren’t relevant to the game and are always valued at zero:

```
[[
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 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, 1, 1, 1, 1, 1, 0],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [2, 0, 0, 0, 0, 0, 0, 2],
 [0, 1, 1, 1, 1, 1, 1, 0]
]]
```



During the game the values of both matrices will change according to the player-moves and the rules of the game:

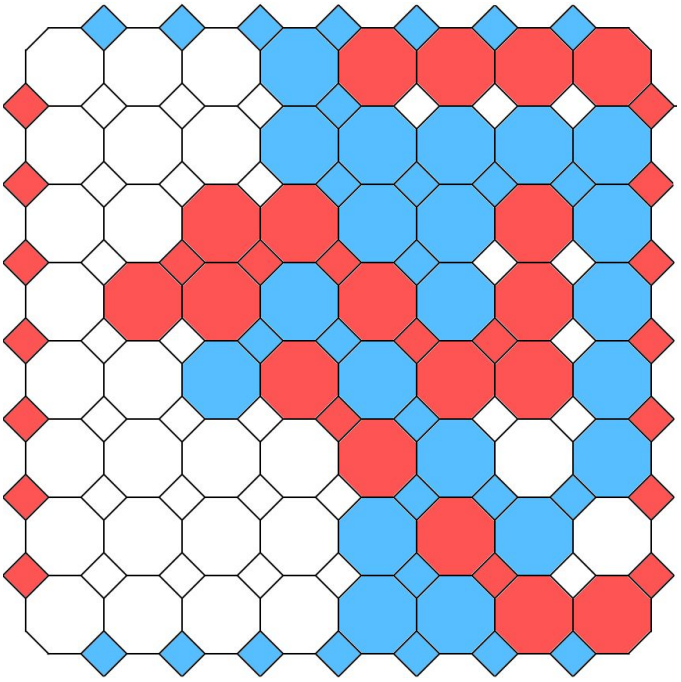
```
[[
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 2, 0, 0],
 [0, 0, 0, 0, 1, 2, 0, 0],
 [0, 2, 1, 2, 2, 1, 0, 0],
 [0, 0, 2, 1, 1, 1, 0, 0],
 [0, 0, 2, 1, 2, 1, 0, 0],
 [0, 0, 0, 0, 1, 0, 1, 1],
 [0, 0, 0, 0, 0, 0, 0, 0]
],
[
 [0, 1, 1, 1, 1, 1, 1, 0],
 [2, 0, 0, 0, 0, 0, 0, 2],
```



[2, 0, 0, 0, 0, 0, 0, 0, 2],
[2, 0, 0, 0, 0, 2, 0, 0, 2],
[2, 0, 2, 2, 0, 1, 0, 0, 2],
[2, 0, 0, 0, 1, 1, 0, 0, 2],
[2, 0, 0, 0, 1, 1, 1, 0, 2],
[2, 0, 0, 0, 0, 0, 0, 0, 2],
[0, 1, 1, 1, 1, 1, 1, 1, 0]
]]

The end state of the game is very similar to an intermediate state:

[[
[0, 0, 0, 1, 2, 2, 2, 2],
[0, 0, 0, 1, 1, 1, 1, 1],
[0, 0, 2, 2, 1, 1, 2, 1],
[0, 2, 2, 1, 2, 1, 2, 1],
[0, 0, 1, 2, 1, 2, 2, 1],
[0, 0, 0, 0, 2, 1, 0, 1],
[0, 0, 0, 0, 1, 2, 1, 0],
[0, 0, 0, 0, 1, 1, 2, 2]
],
[
[0, 1, 1, 1, 1, 1, 1, 1, 0],
[2, 0, 0, 0, 1, 0, 0, 0, 2],
[2, 0, 0, 0, 1, 1, 1, 1, 2],
[2, 0, 0, 0, 0, 1, 0, 0, 2],
[2, 0, 0, 1, 1, 0, 0, 0, 2],
[2, 0, 0, 0, 0, 1, 0, 0, 2],
[2, 0, 0, 0, 0, 1, 1, 1, 2],
[2, 0, 0, 0, 0, 1, 0, 0, 2],
[0, 1, 1, 1, 1, 1, 1, 1, 0]
]]



4. Game display

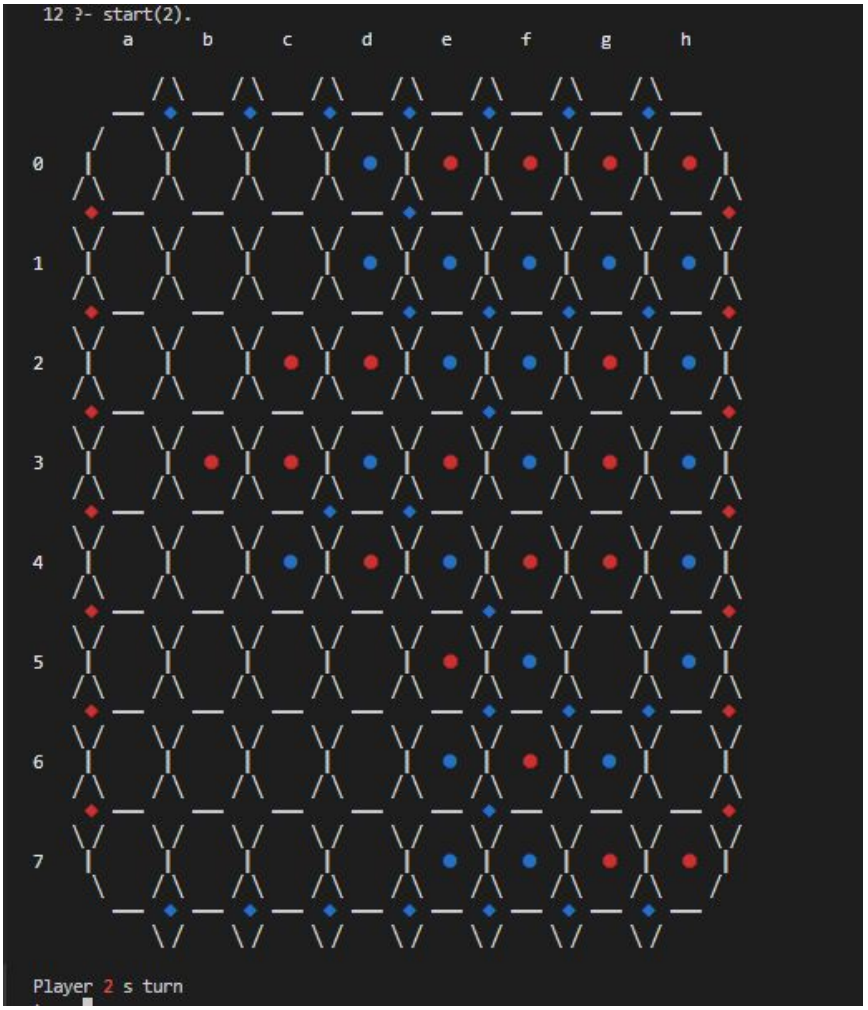
To display the game the predicate '`display_game(Board, Player).`' issues, in order, calls to draw the horizontal coordinates ('`display_horizontal_coordinates(a, 8).`'), draw the board ('`display_board(OctagonBoard, SquareBoard, 0).`') and display which player as the active play('`write("Player "), write_player(Player), write(" s turn").`').

The *'display_board/3'* predicate recursively iterates through both matrices drawing them row by row, while also displaying the row-coordinates (usage of the third argument) and the separators that make up the shape of the cells and, collectively, the whole board. To accomplish this, predicates such as *'display_square_row_borders/2'* , *'display_octagon_row/1'* , *'display_octagon_hor_separator/0',*

'display_octagon_ver_separator/0', etc. are used. The row drawing predicates call predicates like *'display_octagon_piece/1'* and *'display_square_piece/1'* that, given their argument, draw the correct unicode character with the correct color.

To make development easier we made a '*start/1'* predicate that receives a player id, initializes the board and displays the game corresponding to that board and player-turn.

The final result of displaying the game looks like this:



5. Attachment: display code

5.1 At squex.pl

```
:- [display].

start(Player) :- init_board(Board), display_game(Board, Player).

init_board([[
             [0, 0, 0, 1, 2, 2, 2, 2],
             [0, 0, 0, 1, 1, 1, 1, 1],
             [0, 0, 2, 2, 1, 1, 2, 1],
             [0, 2, 2, 1, 2, 1, 2, 1],
             [0, 0, 1, 2, 1, 2, 2, 1],
             [0, 0, 0, 0, 2, 1, 0, 1],
             [0, 0, 0, 0, 1, 2, 1, 0],
             [0, 0, 0, 0, 1, 1, 2, 2]
            ],
            [
             [0, 1, 1, 1, 1, 1, 1, 1, 0],
             [2, 0, 0, 0, 1, 0, 0, 0, 2],
             [2, 0, 0, 0, 1, 1, 1, 1, 2],
             [2, 0, 0, 0, 0, 1, 0, 0, 2],
             [2, 0, 0, 1, 1, 0, 0, 0, 2],
             [2, 0, 0, 0, 0, 1, 0, 0, 2],
             [2, 0, 0, 0, 0, 1, 1, 1, 2],
             [2, 0, 0, 0, 0, 1, 0, 0, 2],
             [0, 1, 1, 1, 1, 1, 1, 1, 0]
            ]]
).
```

5.2 At display.pl

```
% display board

display_game([OctagonBoard, SquareBoard |[]], Player) :-
    display_horizontal_coordinates(a, 8), nl,
    display_board(OctagonBoard, SquareBoard, 0), nl, nl,
    write("Player "), write_player(Player), write(" s turn").

write_player(1) :- ansi_format([bold, fg(blue)], 1, [world]).
write_player(2) :- ansi_format([bold, fg(red)], 2, [world]).

display_board([], [], _).
display_board([], [SquareRow | SquareBoard], Y) :-
    display_square_row_borders(SquareRow, Y),
    YNext is Y + 1,
    display_board([], SquareBoard, YNext).
display_board([OctagonRow | OctagonBoard], [SquareRow | SquareBoard], Y) :-
```

```
    display_square_row_borders(SquareRow, Y), nl,
    ansi_format(bold, Y, [world]), write(' '),
    display_octagon_hor_separator(),
    display_octagon_row(OctagonRow), nl,
    YNext is Y + 1,
    display_board(OctagonBoard, SquareBoard, YNext).


% display pieces

display_square_piece(1) :- ansi_format(fg(blue), '\u25C6', [world]).
display_square_piece(2) :- ansi_format(fg(red), '\u25C6', [world]).
display_square_piece(_) :- ansi_format([], ' ', [world]).

display_octagon_piece(1) :- ansi_format(fg(blue), '\u2BC3', [world]).
display_octagon_piece(2) :- ansi_format(fg(red), '\u2BC3', [world]).
display_octagon_piece(_) :- ansi_format([], ' ', [world]).


% display rows

display_octagon_row([]).
display_octagon_row([H | T]) :- display_octagon_piece(H),
display_octagon_hor_separator(), display_octagon_row(T).

display_square_row_borders(SquareRow, Y) :-
    write('    '), display_lower_octagon_cell(Y), nl,
    write('     '), display_square_row(SquareRow), nl,
    write('    '), display_upper_octagon_cell(Y).

display_square_row([]).
display_square_row([H | []]) :- display_square_piece(H).
display_square_row([H | T]) :- display_square_piece(H), display_octagon_ver_separator(),
display_square_row(T).


% auxiliary functions to display borders of the cells

display_upper_octagon_cell(0) :- ansi_format(bold, '  \u2571     \u2572 \u2571
\u2572 \u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571
\u2572 \u2571     \u2572 ', [world]).
display_upper_octagon_cell(8) :- ansi_format(bold, '        \u2572 \u2571     \u2572
\u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571
\u2572 \u2571 ', [world]).
display_upper_octagon_cell(_) :- ansi_format(bold, '\u2572 \u2571     \u2572 \u2571
\u2572 \u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571     \u2572 \u2571
\u2572 \u2571     \u2572 \u2571', [world]).

display_lower_octagon_cell(0) :- ansi_format(bold, '        \u2571 \u2572     \u2571
\u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572
\u2571 \u2572', [world]).
display_lower_octagon_cell(8) :- ansi_format(bold, '  \u2572     \u2571 \u2572
\u2571 \u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572
```

```
\u2571 \u2572     \u2571 ', [world]).
display_lower_octagon_cell(_) :- ansi_format(bold, '\u2571 \u2572     \u2571 \u2572
\u2571 \u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572     \u2571 \u2572
\u2571 \u2572     \u2571 \u2572', [world]).

display_octagon_ver_separator() :- write('  \u2501\u2501\u2501  ').
display_octagon_hor_separator() :- write('   \u2503   ').


% display coordinates

display_horizontal_coordinates(LastChar, 1) :- write('   '), ansi_format(bold, LastChar,
[world]), nl.
display_horizontal_coordinates(a, Num) :-
    Num > 1,
    write('         '),
    ansi_format(bold, a, [world]),
    write('    '),
    char_code(a, Code),
    NewCode is Code + 1,
    NewNum is Num - 1,
    char_code(NewChar, NewCode),
    display_horizontal_coordinates(NewChar, NewNum).
display_horizontal_coordinates(Char, Num) :-
    Num > 1,
    write('   '),
    ansi_format(bold, Char, [world]),
    write('    '),
    char_code(Char, Code),
    NewCode is Code + 1,
    NewNum is Num - 1,
    char_code(NewChar, NewCode),
    display_horizontal_coordinates(NewChar, NewNum).
```