Practical Assignment 2

Solving Uluru Puzzles in Prolog

1. Context: The Uluru Board Game

Uluru: Neuer Tumult am Ayers Rock is a logic puzzle game played on a small board with 6 positions and 6 different colored tokens (green, yellow, blue, orange, white, black). This is a smaller travel-sized variant of a board game played with a larger board and more colors. You can see a YouTube video explaining the rules of the bigger game here: https://youtu.be/_SpAYu4_Db0?si=0nPUNUBbJucD8O49.

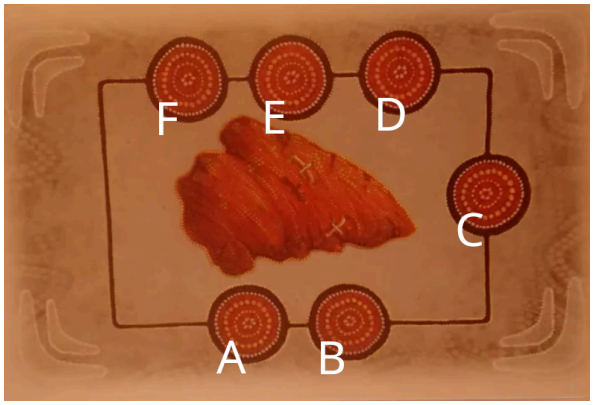
The goal is to place each colored token on the board in such a way that the positioning maximizes the number of constraints satisfied (the constraints are explained below).

In each round:

- The player gets a 6-slot board and 6 colored tokens. Note there is only one token per color.
- Six random "constraint cards" are drawn from a deck, one for each color. Each constraint card imposes a constraint on the positioning of each color in the board (e.g., a possible constraint is that the token of color green needs to be next to the token of color red in the board).
- The player places each of the 6 tokens so that as many constraints as possible are satisfied.
- It is not always possible to satisfy all constraints simultaneously.
- For each violated constraint, the player receives -1 point (a negative point).

For simplicity, we are going to focus only on the game logic from the point-of-view of a single player, ignoring the user interface that keeps track of rounds, player scores, etc.

The board has the following layout (where A,B,C,D,E,F are slots to place your colored tokens):



The possible constraints are (X and Y represent colored tokens):

- X can go into any of the positions (A through F) on the board (trivially satisfied).

- X must be next to Y (according to the map). In this board, A is next to B, B is next to C, C is next to D, D is next to E, and E is next to F, and vice versa (i.e., the relation "next to" is symmetrical). Note that A and F are not considered next to each other.
- X must be exactly one space apart from Y (e.g., positions A and C, B and D, etc.).
- X must be across from Y. A and B are across D, E, and F, and vice versa.
- X and Y must be on the same edge. There are two edges: {A, B} and {D, E, F}.
- X must be in one of the positions given in a list, where positions use the numbering A=1, B=2, C=3, D=4, E=5, F=6.
- Constraints between identical colors (e.g., "black next to black") should be ignored (trivially satisfied).

In this assignment, you will implement the reasoning engine for this puzzle in Prolog.

2. Part 1

Write a Prolog predicate `solve(+Constraints, -Board)` that receives a list of constraints and returns a valid board that satisfies all constraints, or fails if that is not possible.

You must consider that the board is represented by a list

`[A,B,C,D,E,F]`

where each of `A, B, C, D, E, F` is one of the colors green, yellow, blue, orange, white, and black.

You must implement each constraint as a predicate as follows (X and Y are colors):

- `anywhere(X, Board)`: X can go anywhere
- `next_to(X, Y, Board)`: X must be next to Y
- `one_space(X, Y, Board)`: X must be one space apart from Y
- `across(X, Y, Board)`: X must be across from Y
- `same_edge(X, Y, Board)`: X must be on the same edge as Y
- `position(X, L, Board)`: X must be in one of the positions given in the list L

Each constraint must succeed if it is possible to place X and Y in the given board as requested by the constraint. It fails otherwise. As an example, we provide the code for the constraint `next_to`:

```
next_to(X,X,_).
next_to(X,Y,[A,B,C,D,E,F]) :-
  consecutive(X,Y,[A,B,C,D,E,F]).
next_to(X,Y,[A,B,C,D,E,F]) :-
  consecutive(Y,X,[A,B,C,D,E,F]).
```

where `consecutive` verifies if the first two parameters occur consecutively in the list provided in the third parameter. We can implement this using append:

```
consecutive(X,Y,Board) :-
  append(Prefix,[X,Y|Suffix], Board).
```

Note that the constraints in the list do not have the board as the last parameter. However, the predicates implementing the constraints always receive the board. You can use the predicate `call`

to turn each constraint in the given list of constraints into an executable goal to which the board is passed as a parameter. For instance

```
?- call(anywhere(green), [A,B,C,D,E,F]). % is equivalent to execute anywhere(green, [A,B,C,D,E,F]).
yes
```

should succeed.

3. Example of List of Constraints

Use the following example puzzles to test your program (do not change them, but you can always create new ones):

```
%% 12 solutions
example(1, [ next_to(white,orange),
        next_to(black,black),
        across(yellow,orange),
        next_to(green,yellow),
        position(blue,[1,2,6]),
        across(yellow,blue) ]).

%% 1 solution
example(2, [ across(white,yellow),
        position(black,[1,4]),
        position(yellow,[1,5]),
        next_to(green, blue),
        same_edge(blue,yellow),
        one_space(orange,black) ]).

%% no solutions (5 constraints are satisfiable)
example(3, [ across(white,yellow),
        position(black,[1,4]),
        position(yellow,[1,5]),
        same_edge(green, black),
        same_edge(blue,yellow),
        one_space(orange,black) ]).

%% same as above, different order of constraints
example(4, [ position(yellow,[1,5]),
        one_space(orange,black),
        same_edge(green, black),
        same_edge(blue,yellow),
        position(black,[1,4]),
        across(white,yellow) ]).
```

An example of the execution of your program is as follows:

```
?- example(1, _E), solve(_E, Solutions).
Solutions = [yellow,green,white,orange,black,blue] ? n
Solutions = [green,yellow,white,orange,black,blue] ? n
```

```
Solutions = [yellow,green,black,white,orange,blue] ? n
…
```

2. Part 2

Write a Prolog predicate `best_score(+Constraints, -Score)` that computes the best score for a list of constraints. A score of 0 corresponds to satisfying all constraints, -1 to satisfying all but one constraint, etc.

An example of execution follows:

```
?- example(3, _E), best_score(_E, S).
S = -1 ?
yes
```