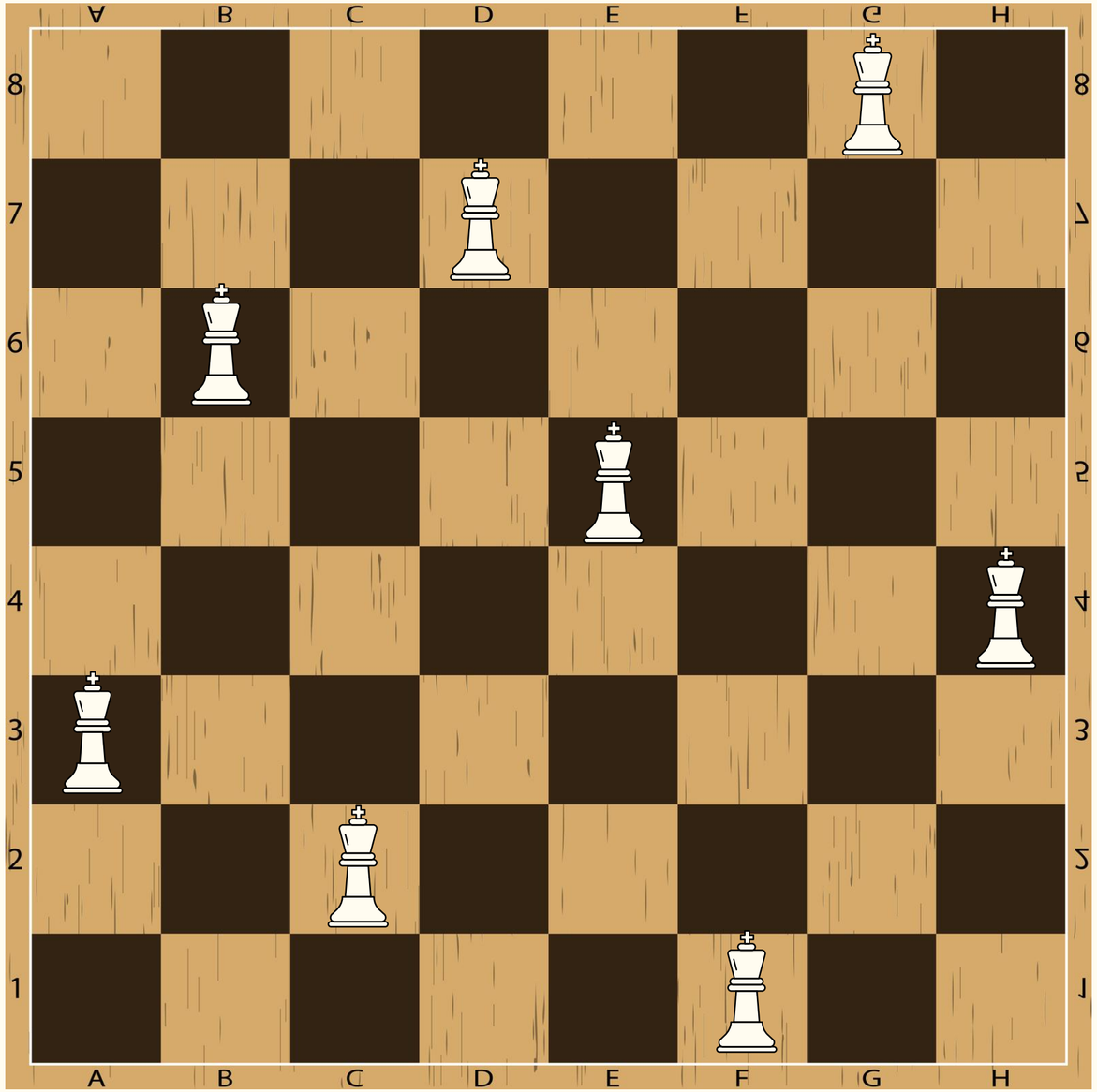
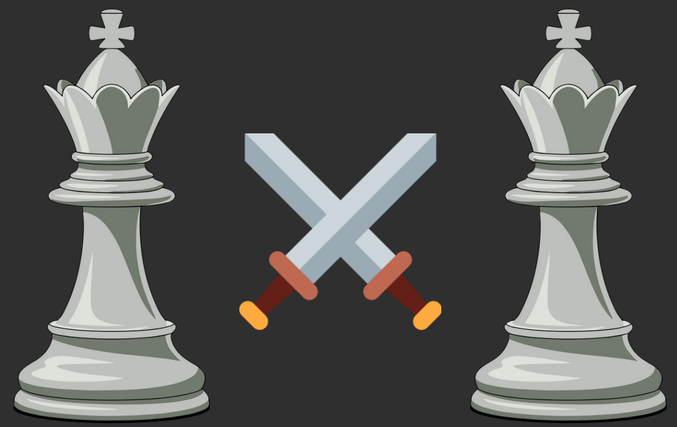
PROBLEMA DAS N RAINHAS



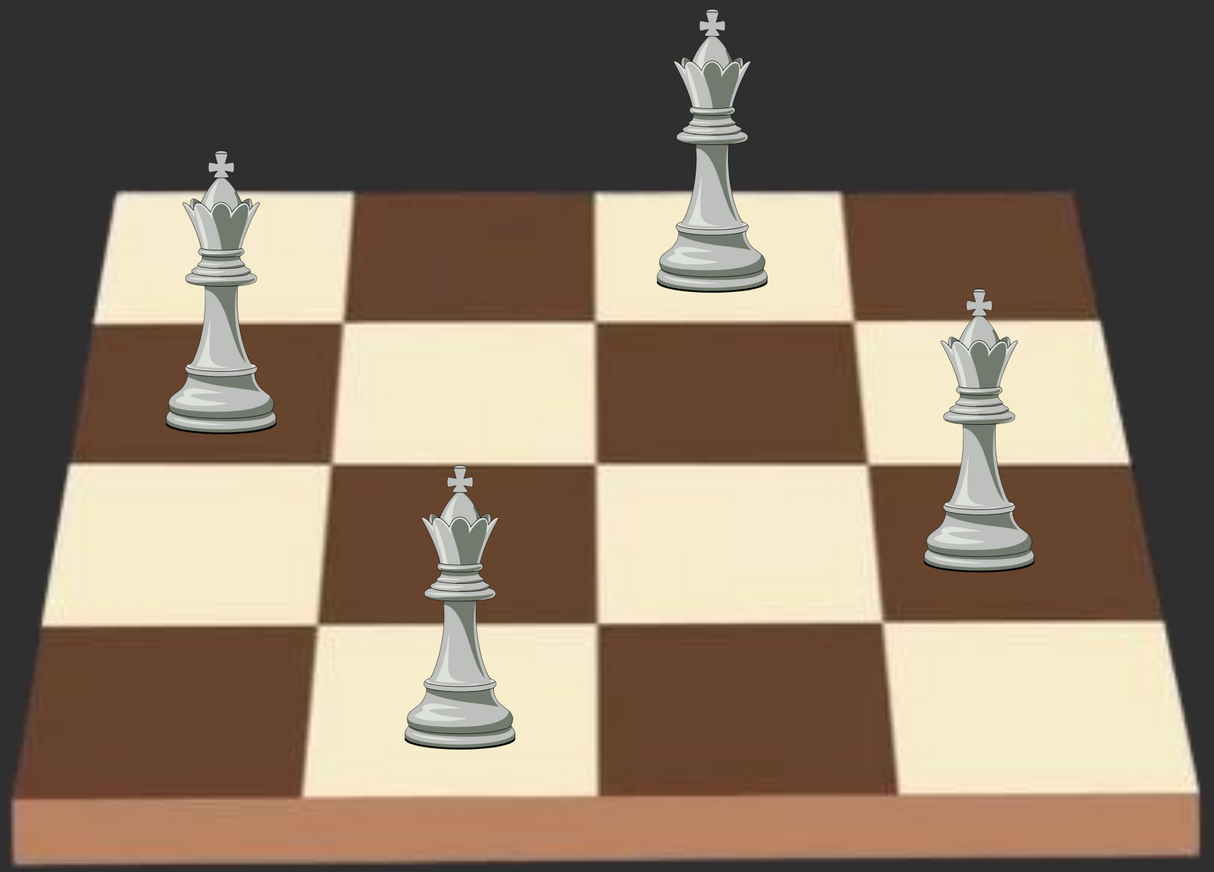
José Dhonatas Alves Sales

O problema das **N rainhas** consiste em posicionar **N** rainhas em um tabuleiro de xadrez **NxN** de forma que nenhuma rainha ataque outra.

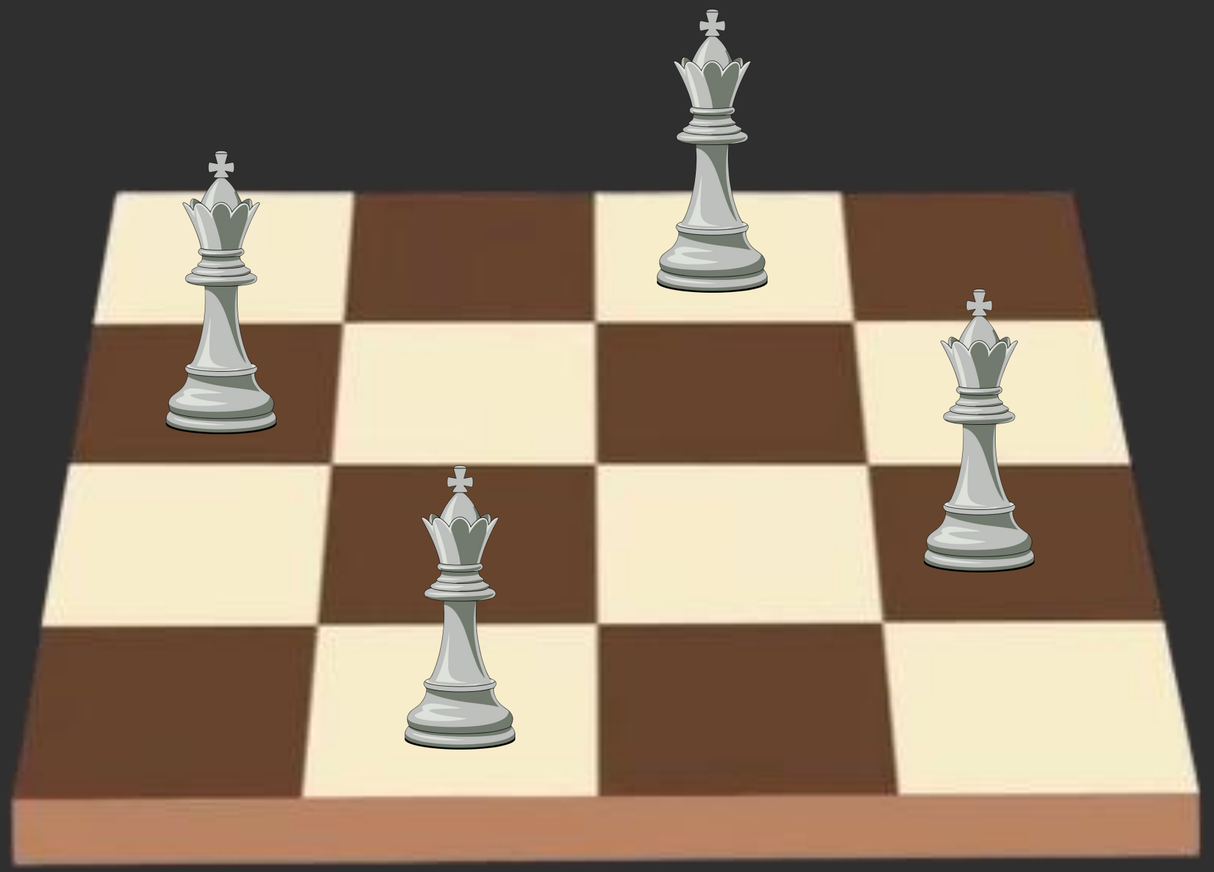
O problema das N-Rainhas, na sua versão de decisão, pertence à classe NP. Isso significa que, se alguém nos der uma configuração de rainhas pronta, conseguimos verificar rapidamente, em tempo polinomial, se nenhuma rainha ataca outra.



Por exemplo, a seguir está uma das soluções para o problema das 4 Rainhas.



Cada rainha é posicionada de forma que não ataque nenhuma outra rainha no tabuleiro.



Neste vídeo, resolveremos um problema com **4 rainhas** como exemplo
e vamos generalizar nossa solução para o problema com **N rainhas**.

Vamos começar com um tabuleiro de xadrez 4x4.

Nossa tarefa é encontrar uma configuração em que nenhuma rainha ataque outra e que haja apenas uma rainha em cada coluna.

resolveNRainhas (int tabuleiro[N][N], int col)

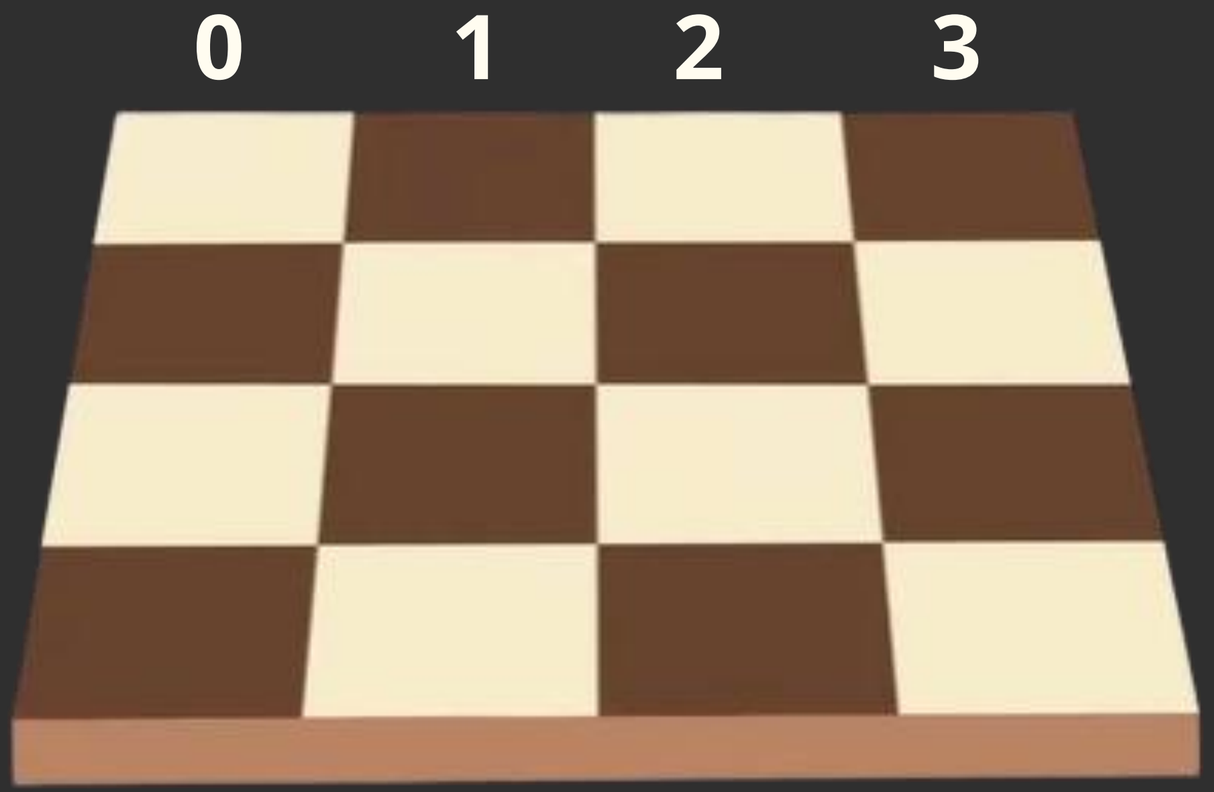
resolveNRainhas (int tabuleiro[N][N], int col)

Esta função retorna o possível arranjo das rainhas para as colunas **col até N** do **tabuleiro[N][N]**, de acordo com as rainhas já posicionadas nas colunas anteriores de **0 a col-1**.

tabuleiro[N][N]
N=4

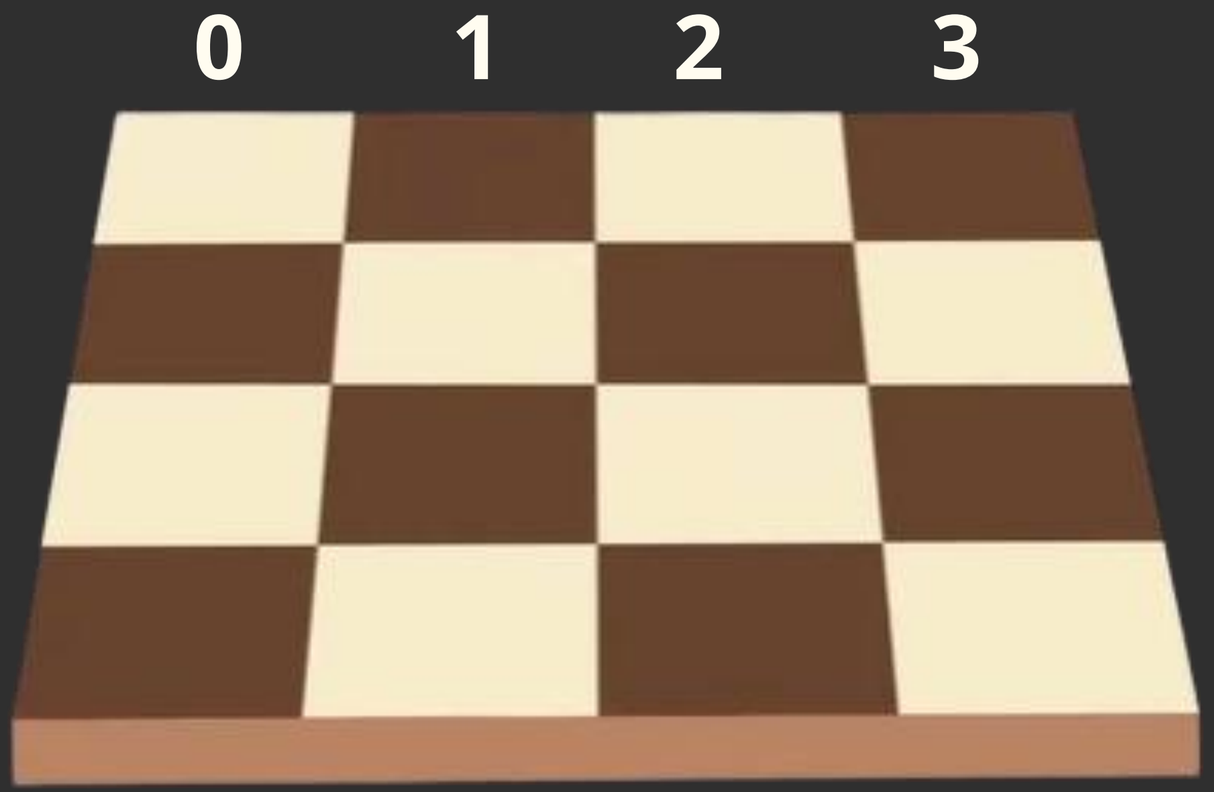
Nós podemos definir essa função como:

resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}



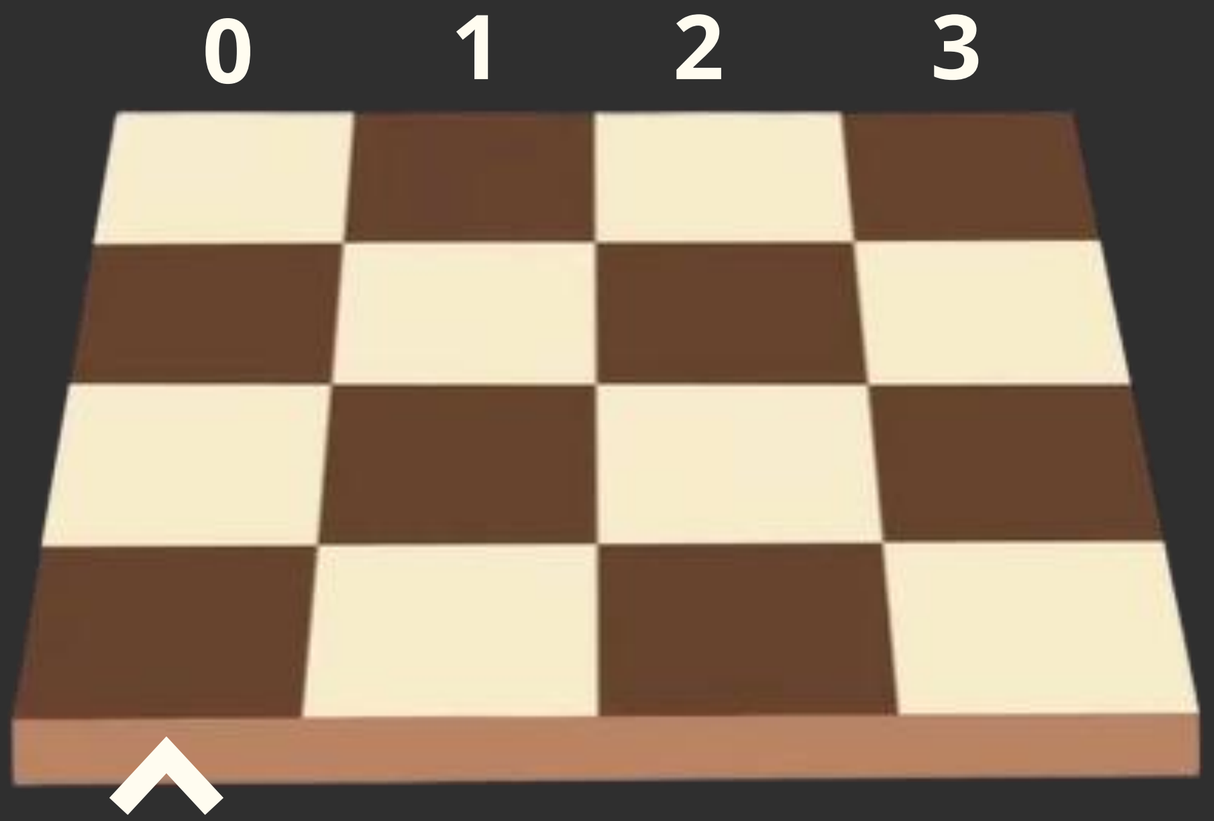
resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}

Então, começamos colocando a rainha na primeira linha, ou seja, na linha 0 da coluna 0.



resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}

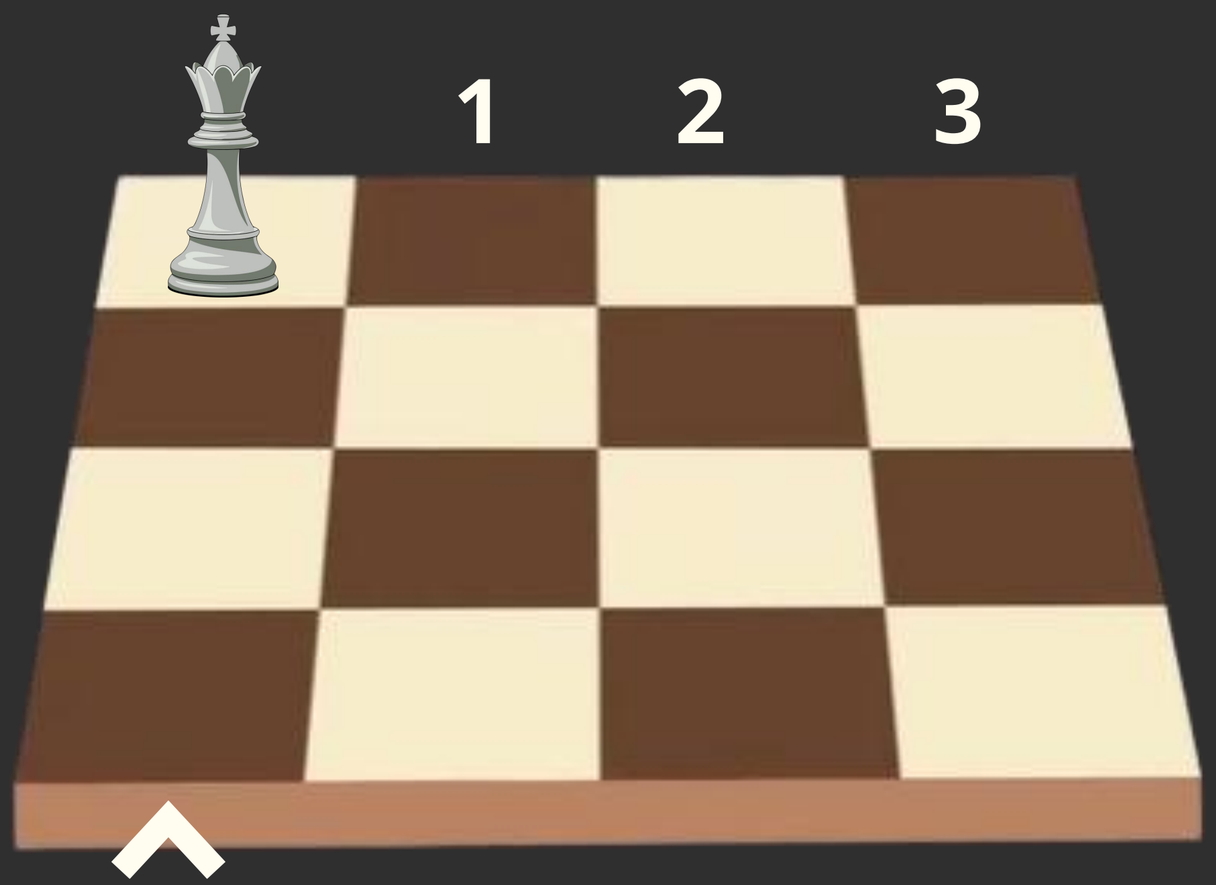
Então, começamos colocando a rainha na primeira linha, ou seja, na linha 0 da coluna 0.



resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}

Então, começamos colocando a rainha na primeira linha, ou seja, na linha 0 da coluna 0.

resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}



resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}

Em seguida, passamos para a próxima coluna, conforme definido na função recursivamente, enquanto salvamos a posição da rainha colocada anteriormente.

resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 2)}

Em seguida, passamos para a próxima coluna, conforme definido na função recursivamente, enquanto salvamos a posição da rainha colocada anteriormente.

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Vamos começar colocando a rainha na **linha 0.**

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Vamos começar colocando a rainha na **linha 0.**

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

A rainha não está segura aqui, então nós movemos para a proxima linha

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Continuamos nos movendo até alcançarmos uma posição segura.

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Esta posição é segura. Passamos para a próxima coluna 2 da mesma forma e seguimos o mesmo procedimento.

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

Como não há posição segura na coluna 2, retrocedemos e passamos para a coluna 1.

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Agora, encontramos a próxima posição segura na coluna 1, passamos para a coluna seguinte e repetimos o mesmo processo.

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

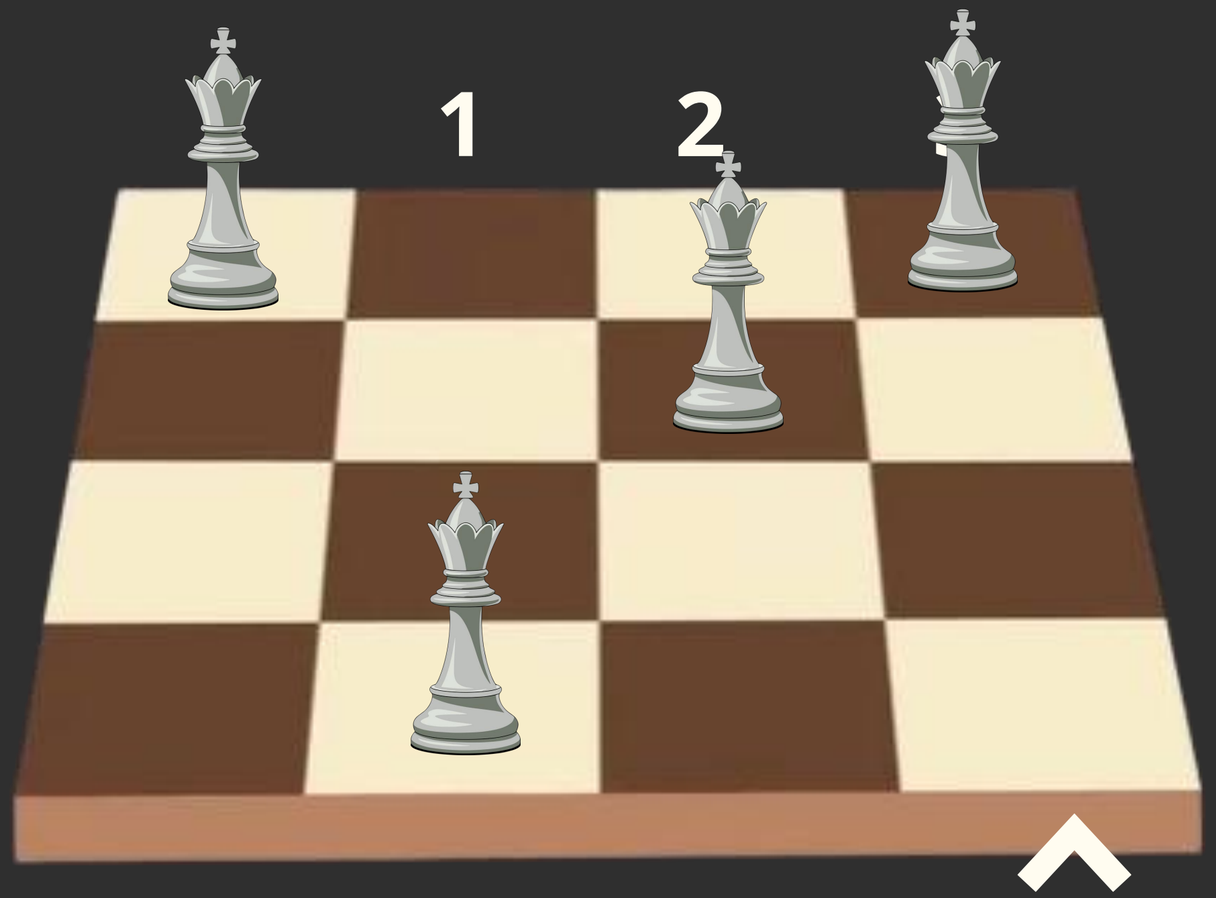
resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

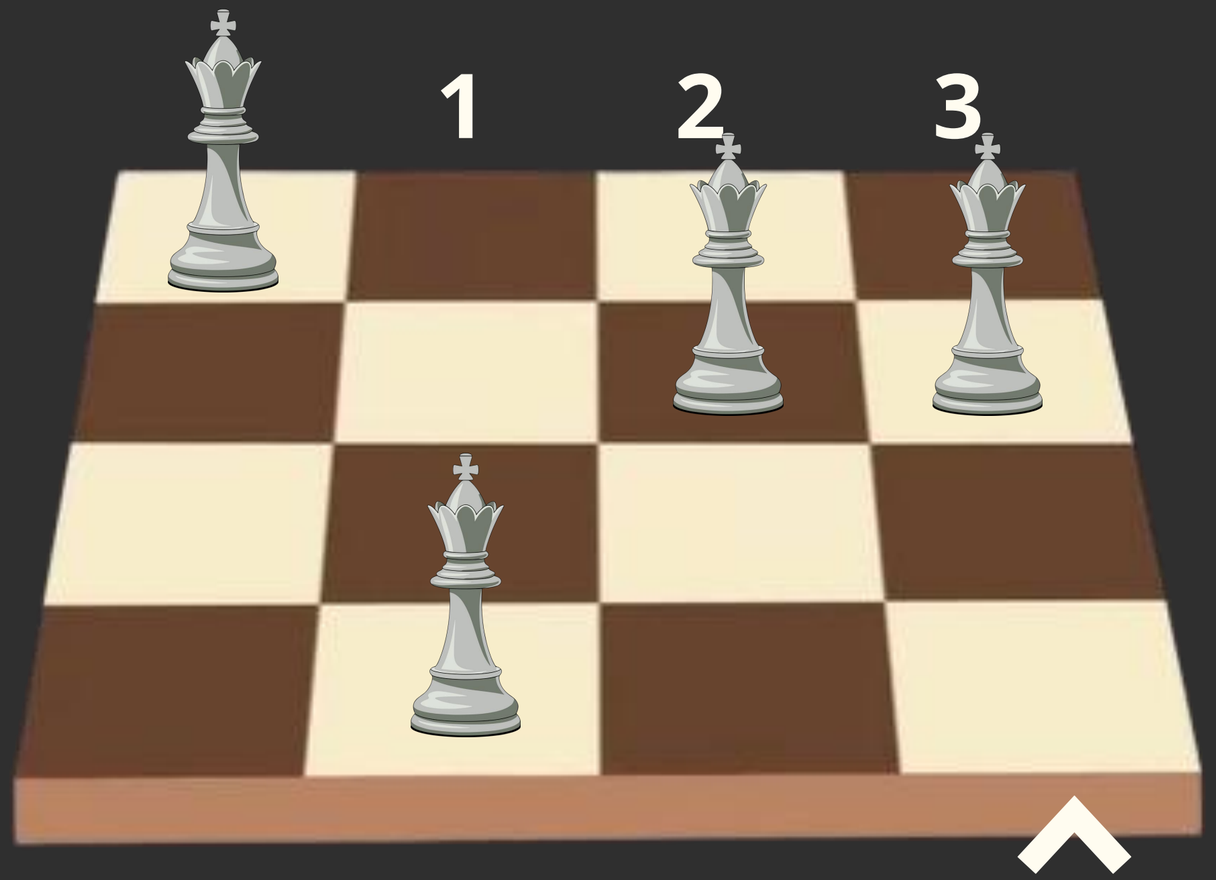
resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 3)

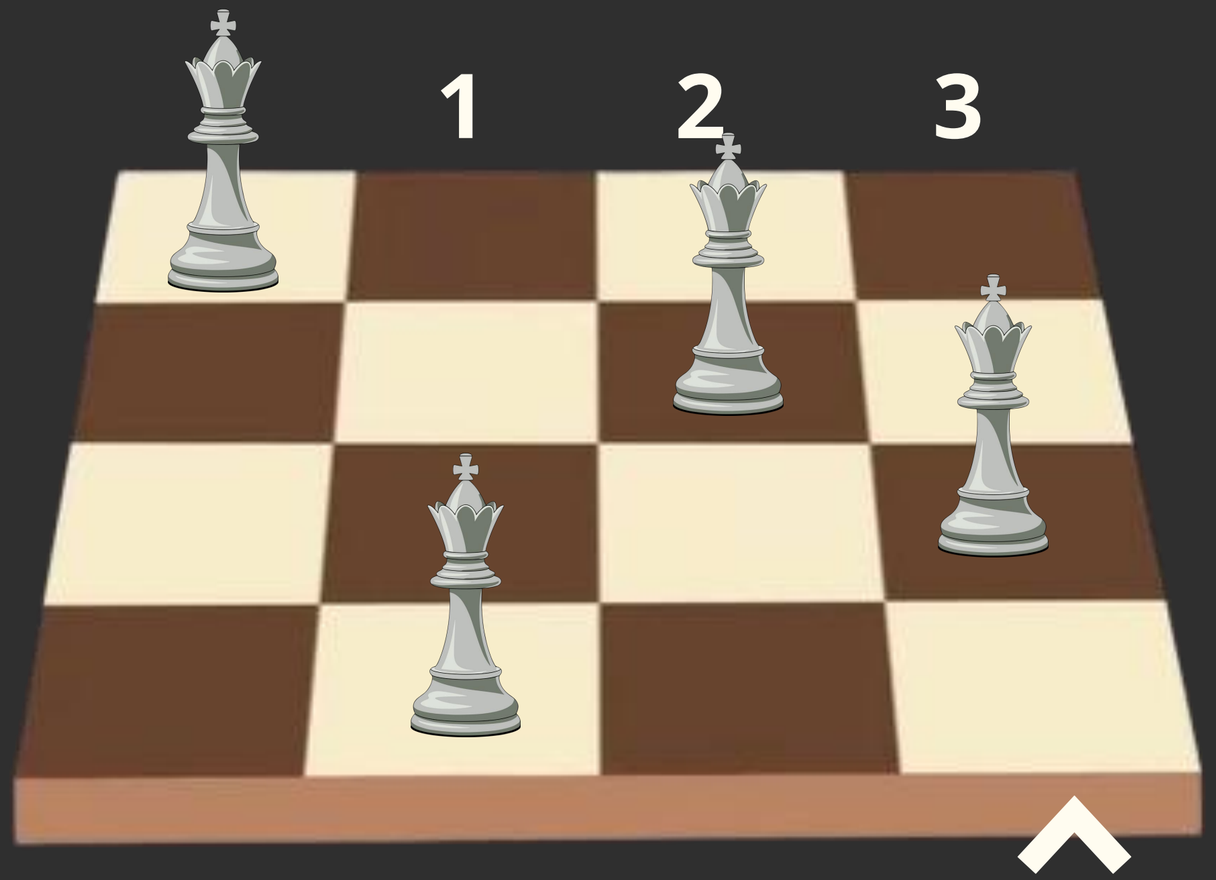
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}



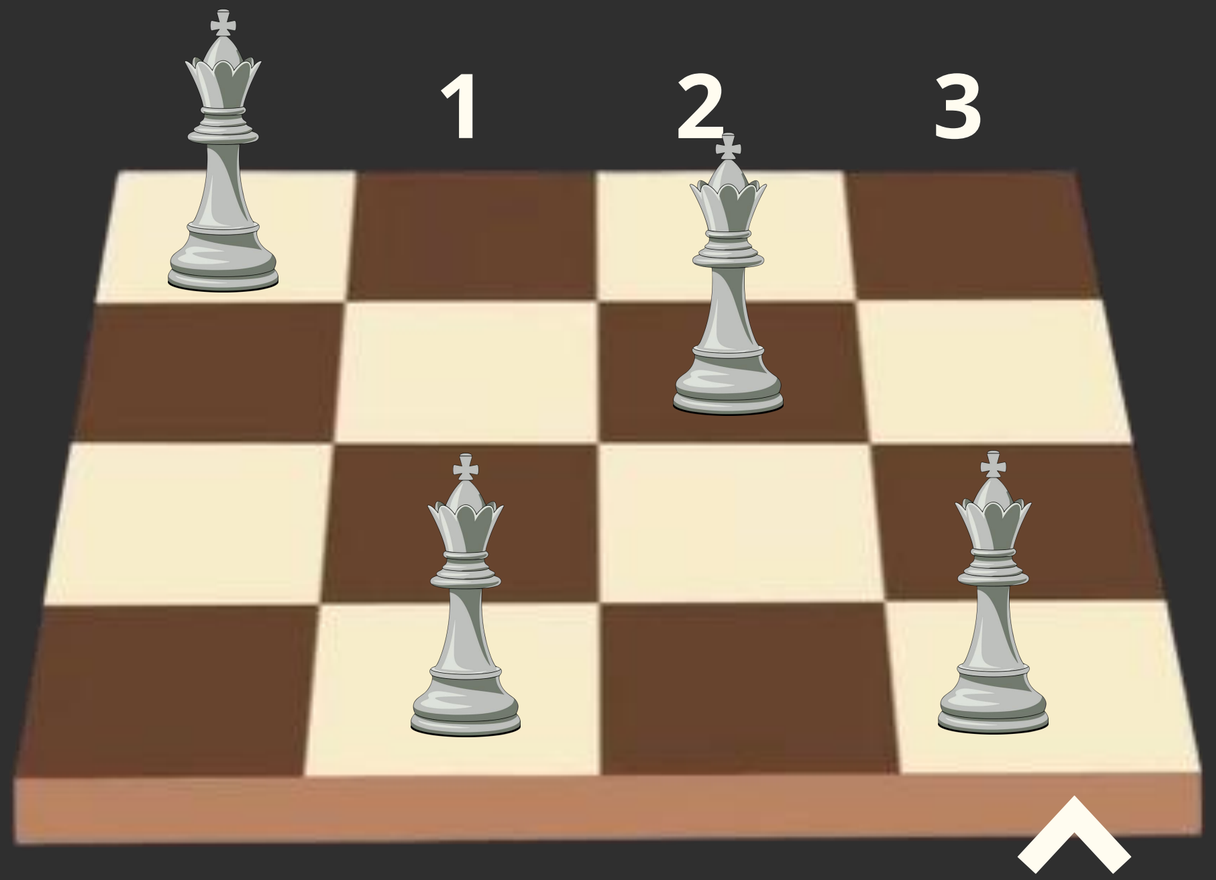
resolveNRainhas (int tabuleiro[4][4], 3)
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}



resolveNRainhas (int tabuleiro[4][4], 3)
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}

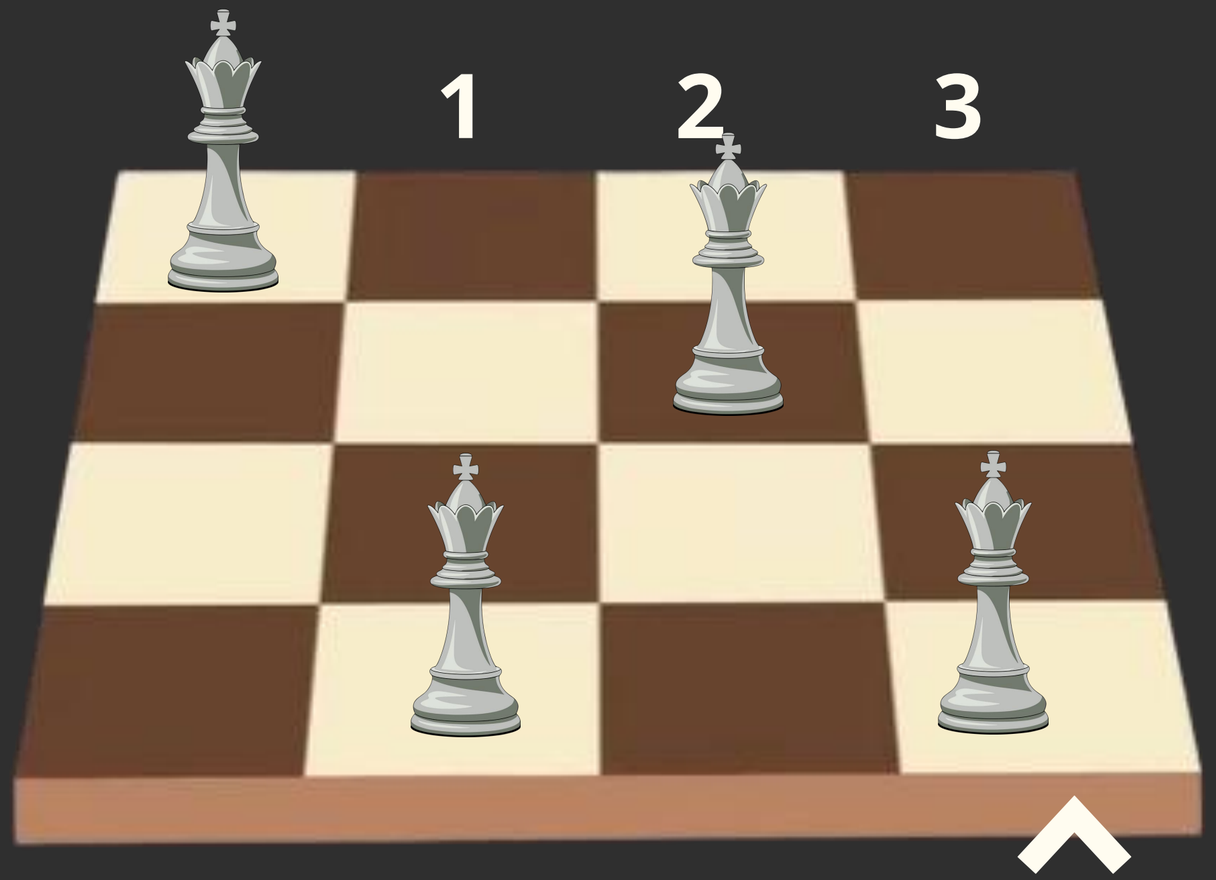


resolveNRainhas (int tabuleiro[4][4], 2)
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}

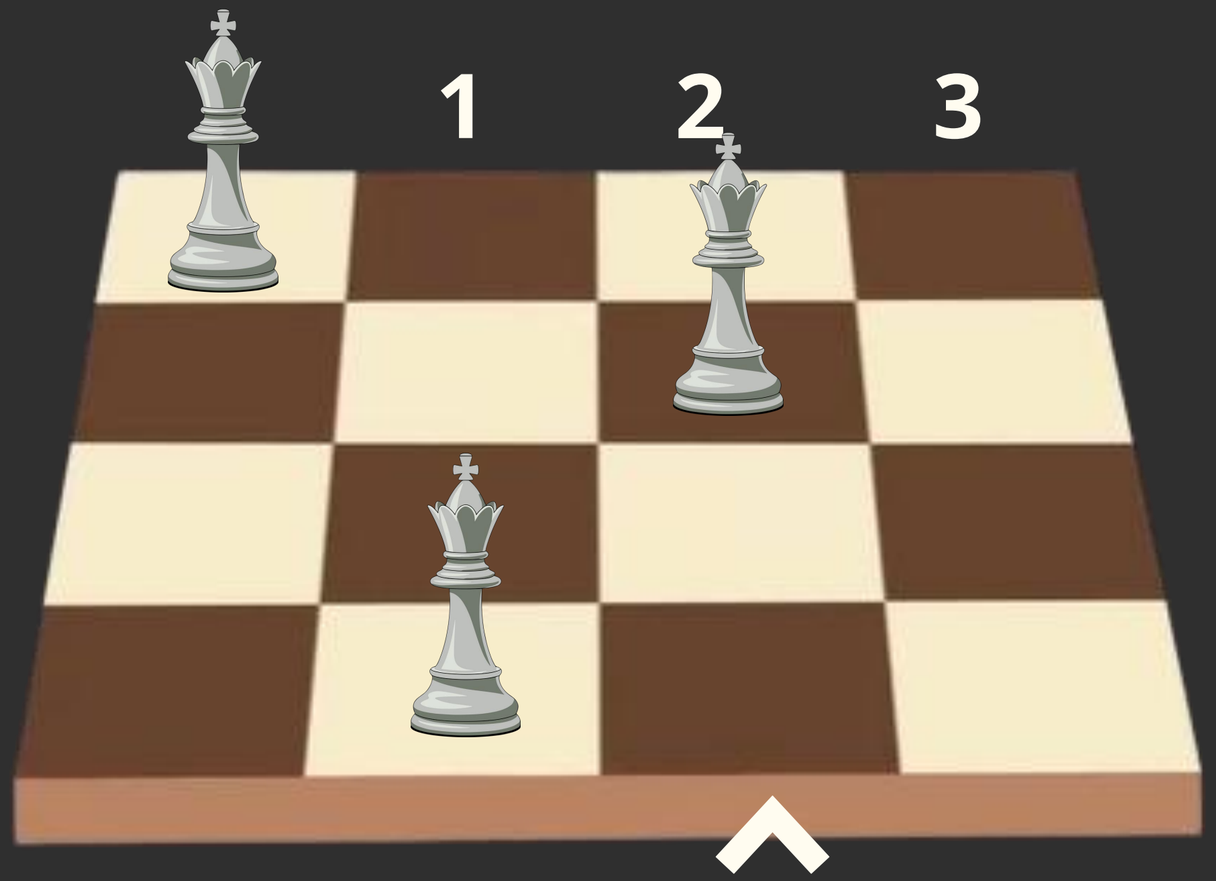


resolveNRainhas (int tabuleiro[4][4], 2)
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}

Não há posição segura aqui, então vamos retroceder.

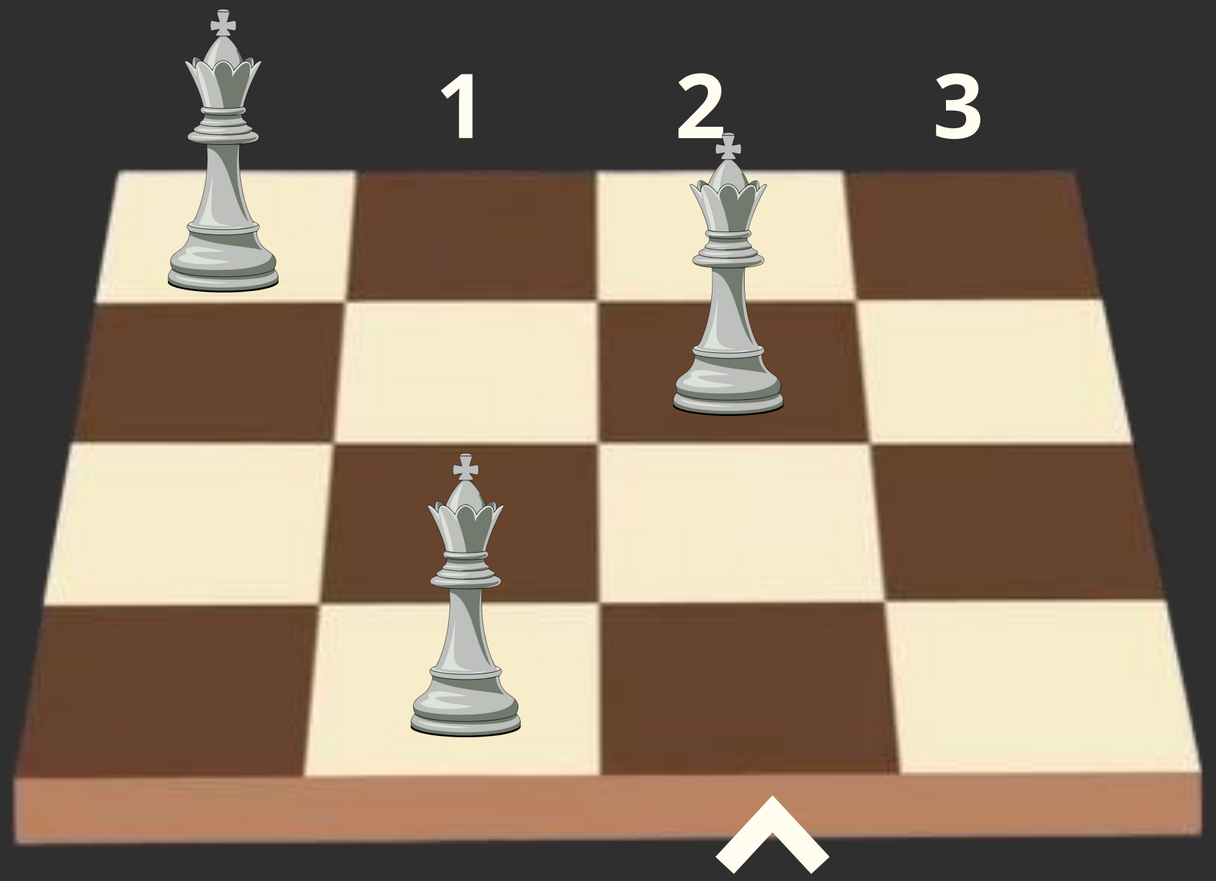


resolveNRainhas (int tabuleiro[4][4], 2)
{ Encontre a linha segura na coluna 3, na última coluna, interrompa a recursão.)}

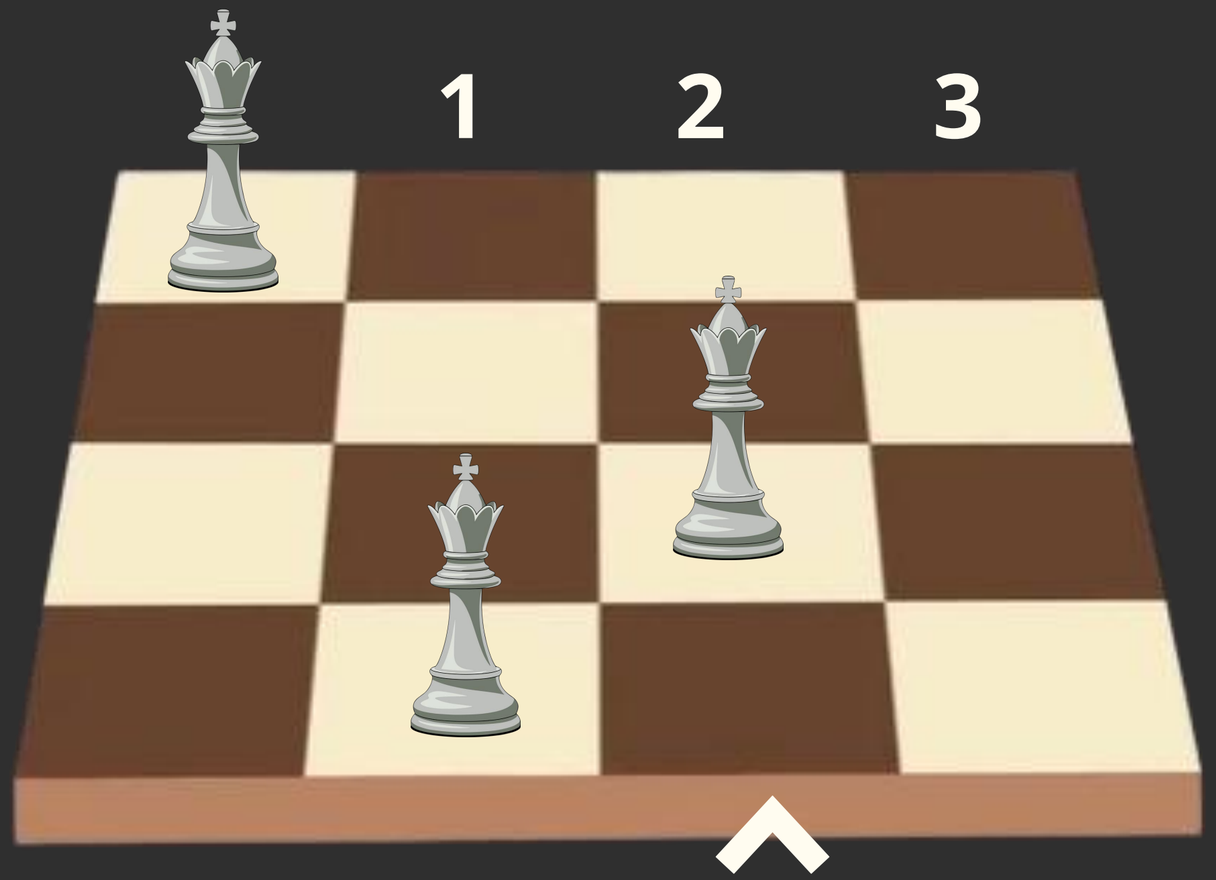


resolveNRainhas (int tabuleiro[4][4], 2)

{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}



resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}



resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

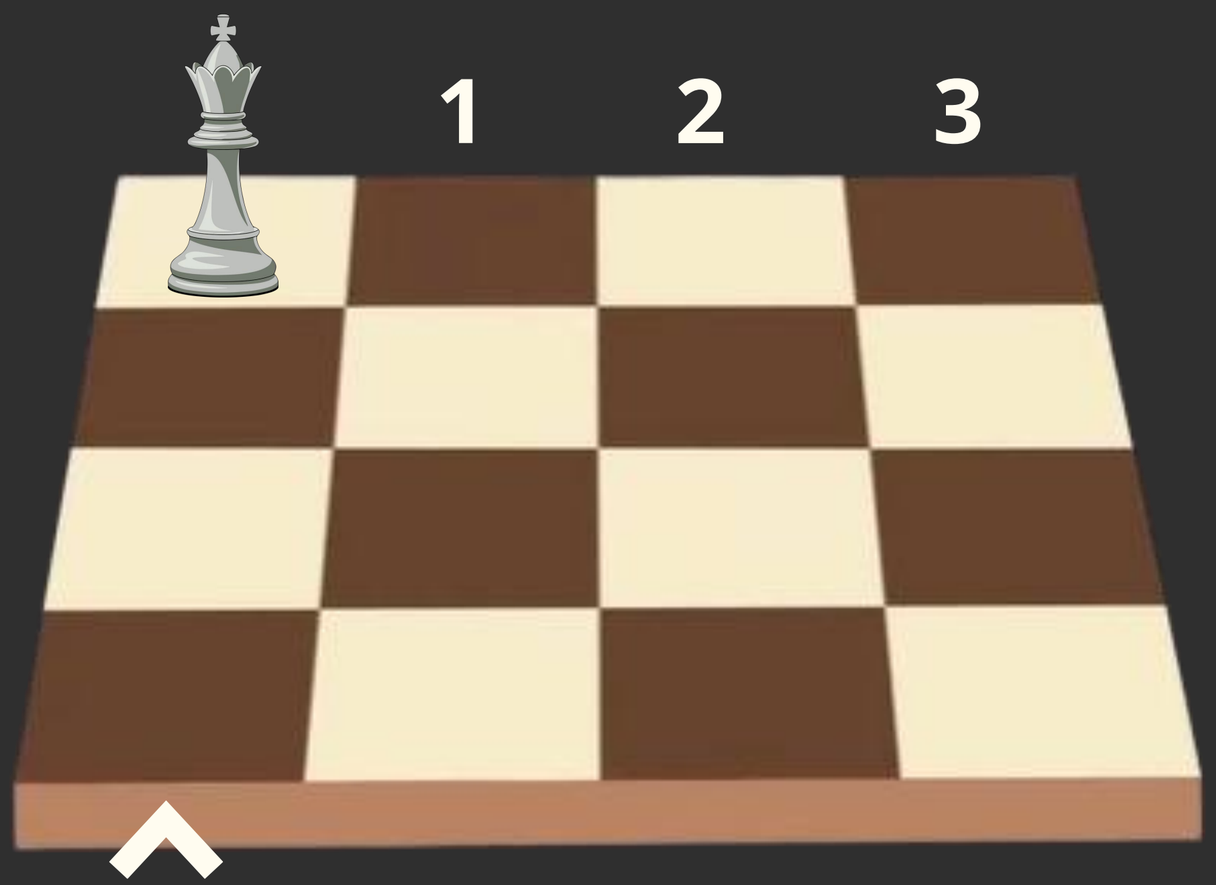
resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

Como não há uma próxima posição segura aqui, voltamos à coluna anterior.

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}



resolveNRainhas (int tabuleiro[4][4], 0)
{ Para cada linha segura na coluna 0 resolverNRainhas (tabuleiro[4][4], 1)}

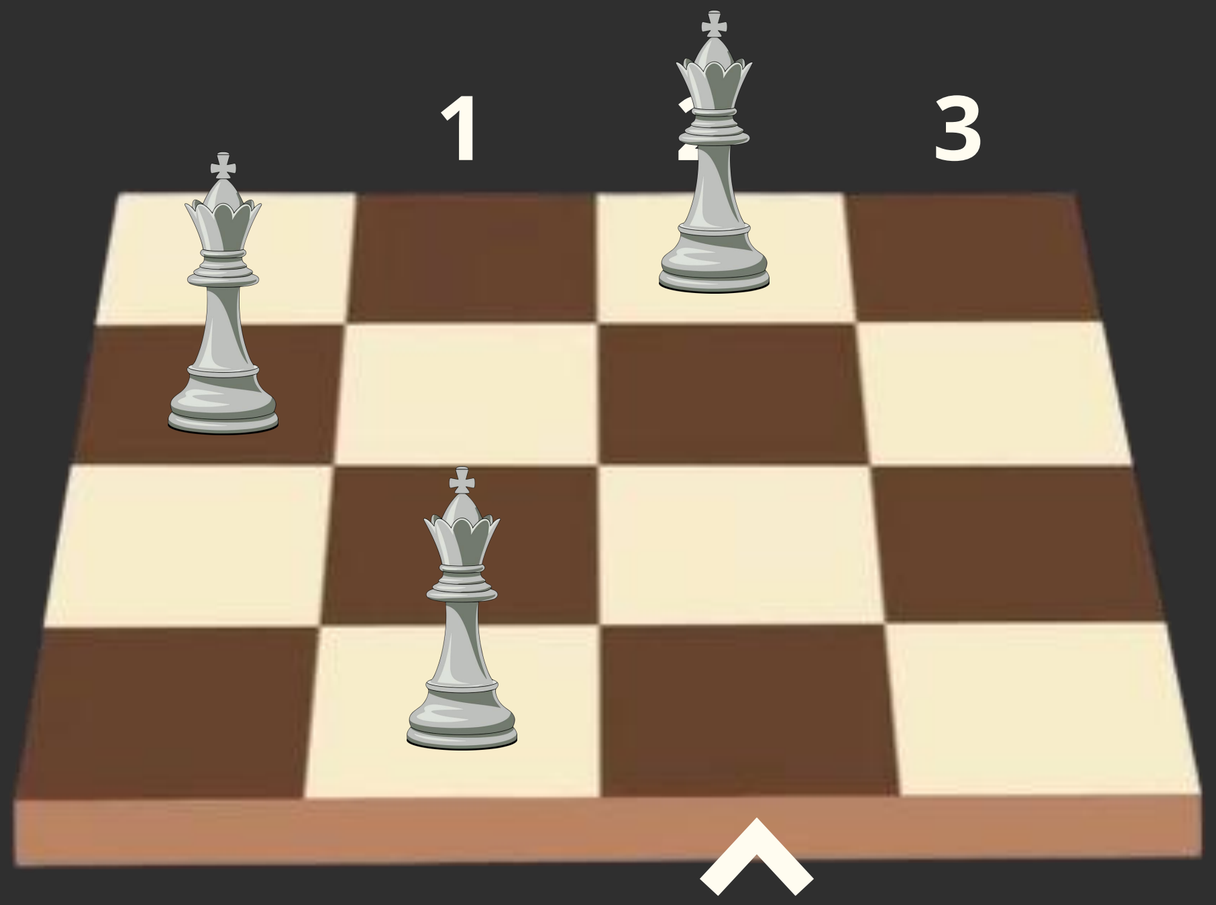
resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

resolveNRainhas (int tabuleiro[4][4], 1)
{ Para cada linha segura na coluna 1 resolverNRainhas (tabuleiro[4][4], 2)}

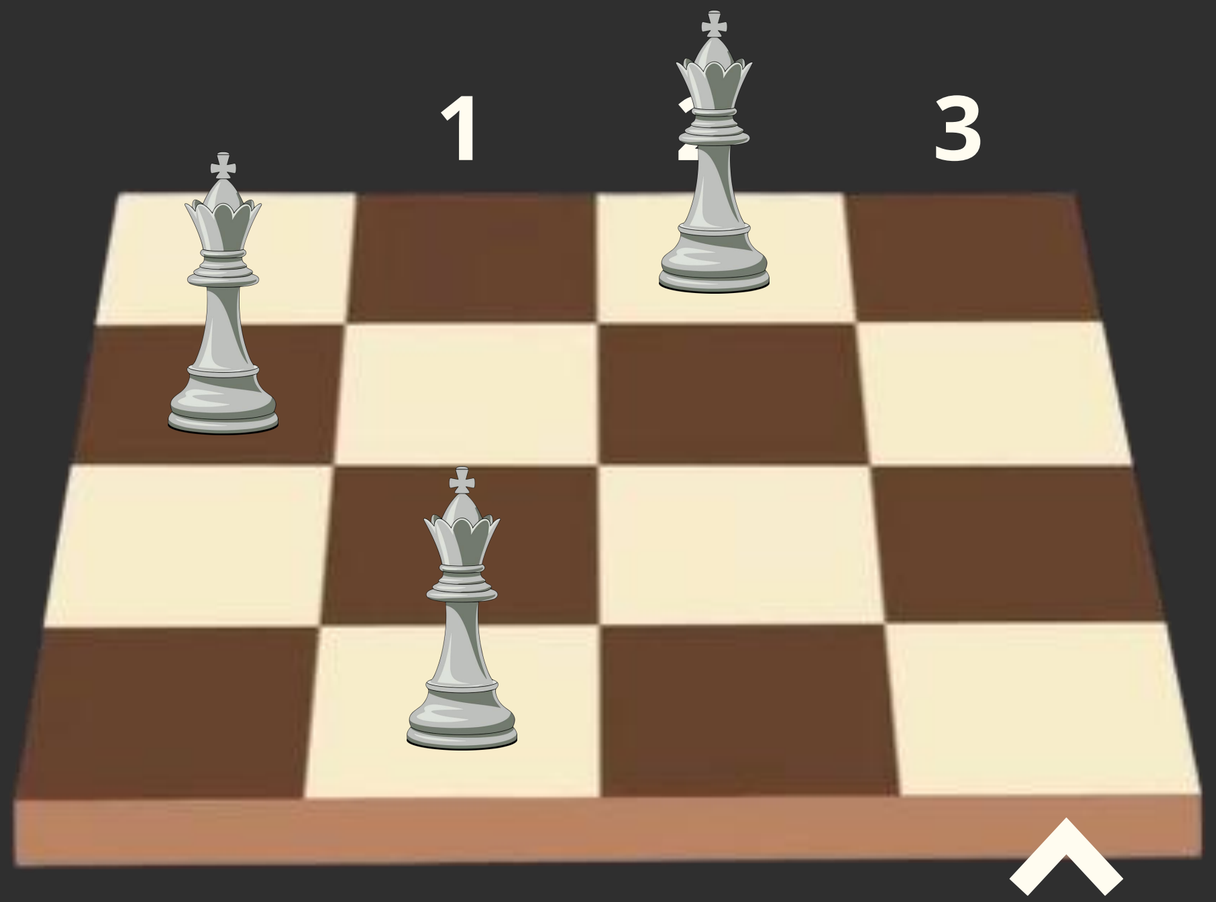
Agilizando as coisas

resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}

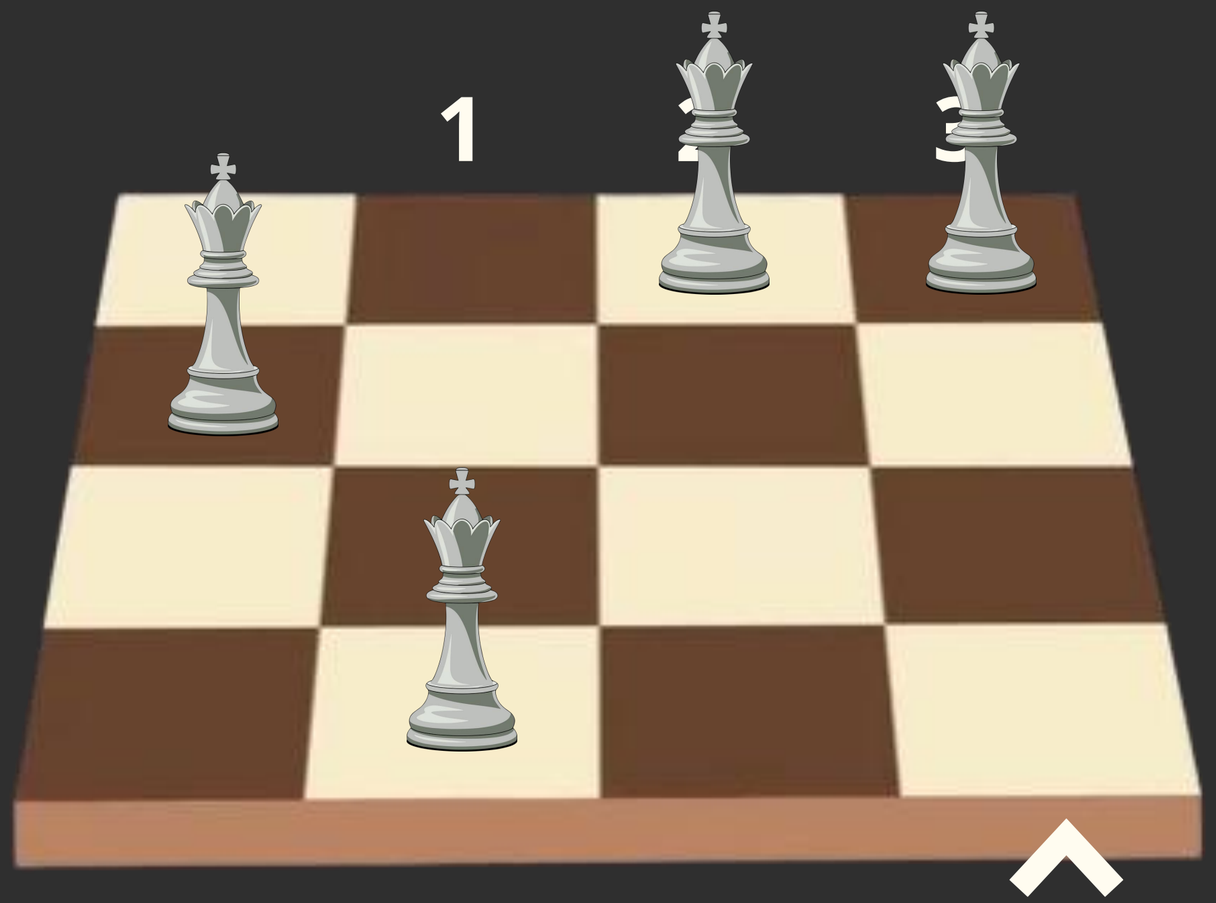
resolveNRainhas (int tabuleiro[4][4], 2)
{ Para cada linha segura na coluna 2 resolverNRainhas (tabuleiro[4][4], 3)}



resolveNRainhas (int tabuleiro[4][4], 3)
{Encontre a linha segura na coluna 3, a última coluna, para que possamos interromper a recursão.}

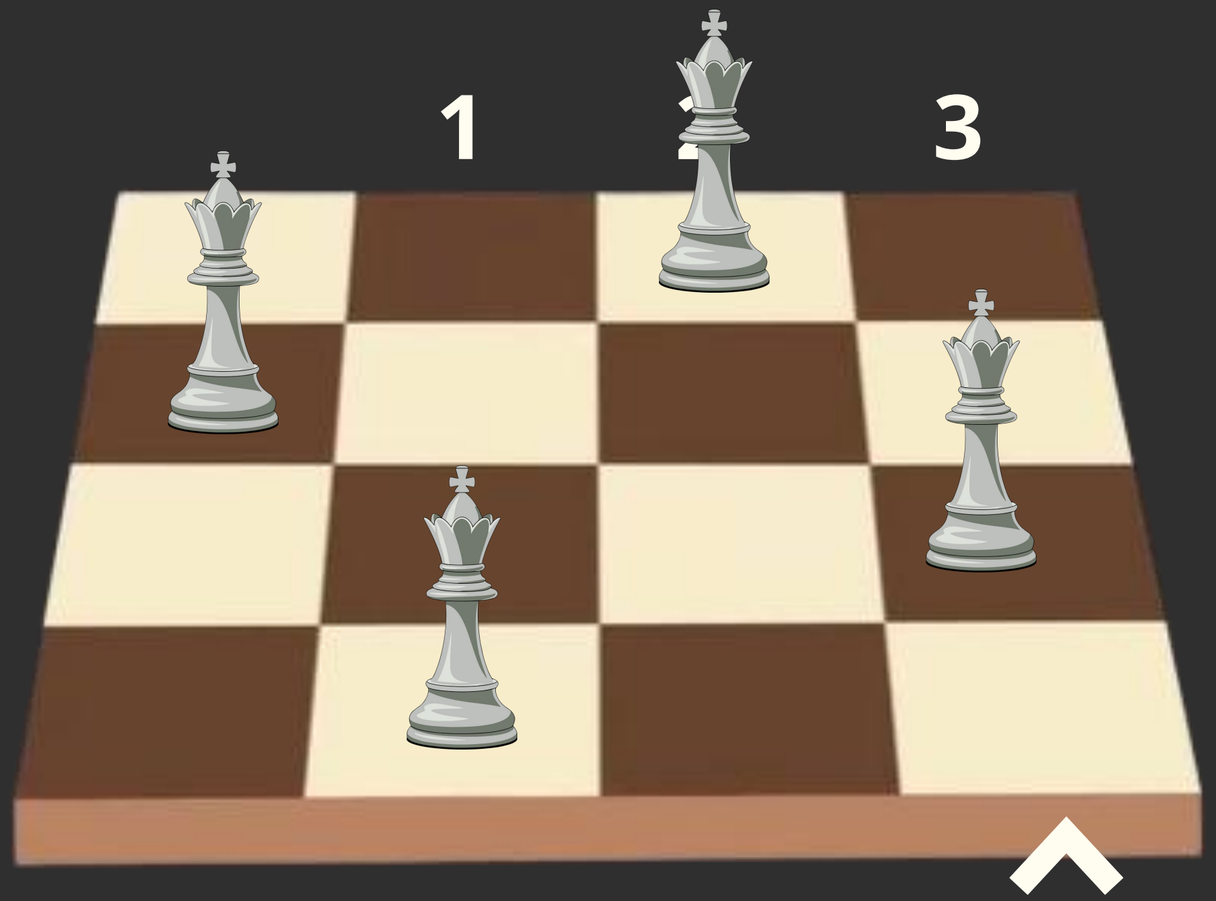


resolveNRainhas (int tabuleiro[4][4], 3)
{Encontre a linha segura na coluna 3, a última coluna, para que possamos interromper a recursão.}



resolveNRainhas (int tabuleiro[4][4], 3)
{Encontre a linha segura na coluna 3, a última coluna, para que possamos interromper a recursão.}

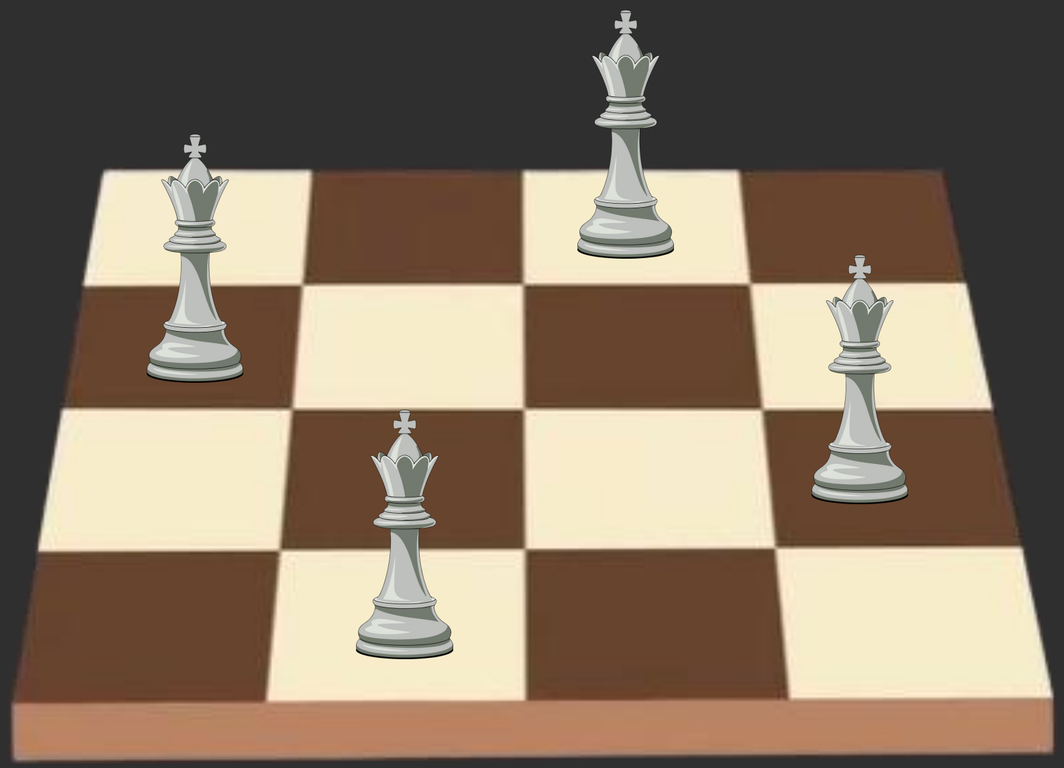
Como todas as colunas têm uma rainha posicionada, temos a nossa solução.



A posição da rainha pode ser armazenada em uma matriz onde 1 representa a rainha e 0 representa uma posição vaga.

A posição da rainha pode ser armazenada em uma matriz onde 1 representa a rainha e 0 representa uma posição vaga.

Por exemplo, para a nossa solução, a representação matricial será:



$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

```
função resolveNRainha(tabuleiro, coluna):

    se coluna >= N então
        retorne verdadeiro

    para linha de 0 até N-1 faça:

        se estaPosiçãoÉSegura(tabuleiro, linha, coluna) então

            tabuleiro[linha][coluna] ← 1

            se resolveNRainha(tabuleiro, coluna + 1) então
                retorne verdadeiro

            tabuleiro[linha][coluna] ← 0   // backtracking

    retorne falso
```