Trabalho 3: Jogos de Dois Jogadores

> O `nim` é um jogo de dois jogadores bem conhecido, jogado com peças pequenas, todas iguais. Inicialmente as peças são arrumadas por linhas: A primeira linha tem uma peça, a segunda linha duas peças, *etc.*

```
Estado inicial do nim, com quatro linhas.

(linha 1) |
(linha 2) ||
(linha 3) |||
(linha 4) ||||
```

> Em cada turno, cada jogador **retira um certo número de peças de uma única linha** (não é obrigatório retirar todas as peças da linha, mas tem de ser retirada pelo menos uma peça).
>
> O primeiro jogador que ficar sem peças para retirar **perde** o jogo. Isto é, quando restar apenas uma linha, o jogador *ativo* tem uma jogada que garante a vitória: retirar todas as peças dessa linha.

```
Neste estado o jogador ativo garante a vitória se deixar apenas uma peça na segunda linha.

(linha 1) |
(linha 2) |||||
```

Exercícios

Grupo 1 (nim)

Considere o `nim` e:

1. Escolha uma **estrutura de dados** adequada para representar os estados do jogo.
2. Defina o predicado `estado_terminal(Estado)` que sucede apenas para os **estados terminais**.
3. Defina uma função `utilidade(EstadoTerminal, Utilidade)` que, para cada estado terminal, define a **utilidade** desse estado (por exemplo, `-1` : derrota, `0` : empate, `+1` : vitória).
4. Use a implementação do algoritmo `minimax` (no ficheiro `minmax.pl`) para testar a sua representação (*sugestão: teste com estados próximos dum estado terminal*).
5. Implemente a pesquisa `alfa-beta` e compare os resultados (tempo e espaço) com a pesquisa `minimax`.
6. Defina uma função que lhe permita calcular o **valor** de qualquer estado do jogo.
   - Use os dois algoritmos anteriores.
   - Modifique o algoritmo `minimax` para cortar (*cutoff*) à profundidade 3.
   - Avalie o desempenho (tempo e espaço) da sua função de avaliação.
7. Implemente um agente inteligente que jogue o `nim`.
8. Apresente uma tabela com o número de nós expandidos para vários estados do jogo (pelo menos, dez estados), com os vários algoritmos.

Grupo 2 (galo)

Considere o `jogo do galo` e:

1. Escolha uma **estrutura de dados** adequada para representar os estados do jogo.
2. Defina o predicado `estado_terminal(Estado)` que sucede apenas para os **estados terminais**.
3. Defina uma função `utilidade(EstadoTerminal, Utilidade)` que, para cada estado terminal, define a **utilidade** desse estado (por exemplo, `-1` : derrota, `0` : empate, `+1` : vitória).
4. Implemente um agente inteligente que jogue o `jogo do galo` usando várias estratégias (`minimax` , `alfa-beta` e `corte em profundidade`).
5. Apresente uma tabela com o número de nós expandidos para vários estados do jogo (pelo menos, dez estados), com os vários algoritmos.