PFL - 2ºTrabalho Prático

Implementação de um Jogo de Tabuleiro em Prolog

Identificação do tópico e do grupo

Neste projeto foi desenvolvido o jogo **STAQS** em Prolog, no contexto da unidade curricular de Programação Funcional e em Lógica (PFL).
Foi realizado pelos elementos do grupo *STAQS_7* da turma 10:

- João Francisco Costa Cordeiro, up202205682
- Luana Filipa de Matos Lima, up202206845

Ambos os membros contribuíram de igual forma (50% / 50%) para este trabalho.

Instalação e Execução

Para instalar este jogo é necessário fazer o download da pasta ***PFL_TP2_T10_STAQS_7.zip*** e descompactar. Dentro do diretório ***scr*** deve-se consultar o ficeiro ***game.pl*** através da linha de comandos ou pela própria UI do Sicstus Prolog 4.9. O jogo está disponível para ambientes Windows e Linux.
O jogo inicia-se com o predicado play/0:

```
?- play.
```

Descrição do Jogo

STAQS é um jogo de tabuleiro estratégico para dois jogadores. É jogado num tabuleiro 5 por 5 em que cada jogador tem 4 peças empilháveis. O jogo começa com 25 peças neutras, uma por cada casa do tabuleiro. Numa fase inicial (fase de posicionamento), cada jogador deve posicionar, à vez, cada uma das suas peças em cima de uma das peças neutras até as 8 peças estarem colocadas. Já na fase de movimento, à vez, cada jogador deve mover uma das suas pilhas (com a cor correspondente no topo) uma casa, para cima de outra peça neutra.

Principais regras:

- Uma pilha só se pode mover para uma das casas que a contornam no momento, ou seja, pode mover-se na vertical, horizontal ou diagonal;
- Uma pilha não pode saltar para cima de outra pilha ou mover-se para espaços vazios, ou seja, só se pode mover para uma casa com exatamente uma peça neutra;
- Se um jogador não puder mover nenhuma das suas pilhas, deve passar;
- Se ambos os jogadores passarem consecutivamente, o jogo termina.

Critérios para determinar o vencedor:

1. O jogador com a pilha mais alta vence;
2. Se ambos os jogadores tiverem pilhas com a maior altura, vence o que tiver mais pilhas com esse tamanho;
3. Se ainda assim houver empate, comparam-se as seguintes pilhas mais altas ou a maioria das pilhas de altura igual até que o vencedor seja determinado.

Considerações para Extensões do Jogo

Neste jogo é permitido escolher tabuleiros de diferentes tamanhos, sendo que o número de peças de cada jogador depende desse tamanho (o número de peças por jogador é igual ao tamanho do tabuleiro menos 1).

Além disso, o utilizador pode também escolher entre 4 modos de jogo diferentes:

- **Humano vs. Humano -** O jogo espera a interação de dois jogadores;
- **Humano vs. PC -** O jogo espera interação para o primeiro jogador enquanto o segundo jogador é um bot;
- **PC vs. Humano -** Semelhante à opção anterior, mas a ordem de jogadores é trocada;
- **PC vs. PC -** O jogo é jogado por dois bots.

Para casos em que o PC é escolhido para jogar, o utilizador pode também escolher o seu nível de dificuldade entre duas opções:

1 - O bot faz jogadas aleatórias tendo em conta todas as opções disponíveis e válidas;
2 - O bot utiliza um algoritmo greedy para determinar qual a melhor jogada tendo em conta as opções disponíveis e válidas.

Já para casos com um ou dois jogadores humanos, cada um destes pode inserir um nome para ser apresentado nas suas jogadas.

Lógica do Jogo

Representação da Configuração do Jogo

A configuração do jogo encapsula os detalhes essenciais, como o tipo de jogadores (humano ou computador), tamanho do tabuleiro, nomes dos jogadores e níveis de dificuldade dos bots. Isto é representado como um termo Prolog game_config/3:

```
game_config(Player1, Player2, BoardSize)
```

- **Player1** e **Player2** representam os dois jogadores. Para jogadores humanos, inclui o nome do jogador e a cor. Para jogadores controlados pelo computador, inclui o nível de dificuldade, a cor e um identificador predefinido;
- **BoardSize** especifica as dimensões do tabuleiro NxN.

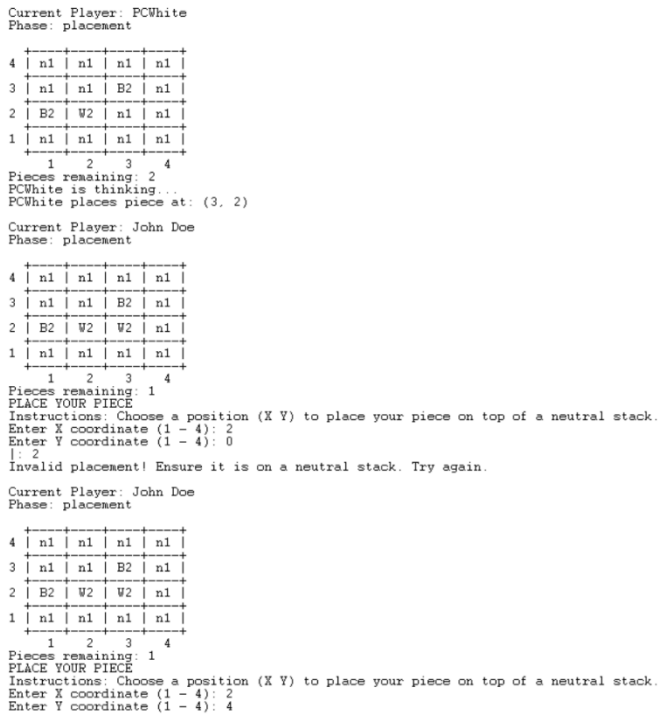
Esta configuração é utilizada no predicado initial_state/2 para inicializar o estado inicial do jogo.

Representação Interna do Estado do Jogo

O estado do jogo captura toda a informação sobre o jogo num dado momento. É representado como:

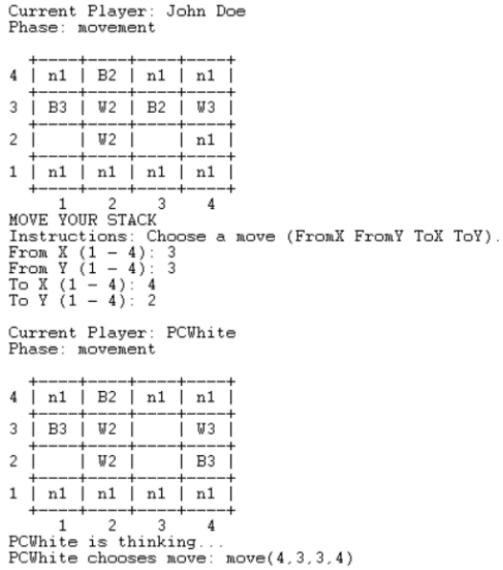
```
game(Board, CurrentPlayer, Phase, BoardSize, GameConfig)
```

- **Board:** Uma lista de listas, onde cada elemento representa uma célula do tabuleiro. As células contêm pilhas (listas) de peças (neutral, blue ou white);
- **CurrentPlayer:** O jogador que está atualmente a jogar;
- **Phase:** Indica se o jogo está na fase de "posicionamento" ou "movimento";
- **BoardSize:** Dimensão do tabuleiro (ex.: 5 para um tabuleiro 5x5);
- **GameConfig:** Contém os detalhes dos dois jogadores e o tamanho do tabuleiro.



Fase de posicionamento

Final do Jogo



Fase de movimento

Representação das jogadas

Os movimentos são representados como um termo Prolog:

```
move(FromX, FromY, ToX, ToY)
```

- **FromX** e **FromY:** Coordenadas da pilha que será movida;
- **ToX** e **ToY:** Coordenadas de destino. Esta representação é utilizada no predicado move/3 para validação e execução dos movimentos.

Interação do utilizador

O sistema de interação do jogo começa no menu principal, onde os jogadores inserem o tamanho do tabuleiro (entre 3 e 9) e escolhem o modo de jogo: H/H, H/PC, PC/H ou PC/PC. Além disso, dependendo do modo selecionado, é solicitado o nome dos jogadores humanos e o nível de dificuldade do computador (1 ou 2). Todas essas entradas são validadas para garantir que estão dentro dos limites definidos.

Durante o jogo, os inputs mudam conforme a fase. Na fase de colocação, os jogadores inserem coordenadas (X, Y) para posicionar peças em pilhas neutras. Na fase de movimento, os jogadores inserem coordenadas (FromX, FromY,ToX, ToY) para mover pilhas para células adjacentes válidas. O sistema verifica automaticamente se os inputs estão dentro dos limites do tabuleiro, se a jogada segue as regras (ex.: destino válido) e, em caso de erro, exibe mensagens explicativas e solicita nova entrada.

```
| ?- play.

Welcome to STAQS!

Instructions:
1. During the PLACEMENT phase, type "X Y" to place your piece on top of a neutral stack.
   Example: "1 1" places your piece at the lower-left corner of the board.
2. During the MOVEMENT phase, type "FromX FromY ToX ToY" to move a stack to an adjacent
   cell (vertical, horizontal, diagonal).
   Example: "1 1 2 2" moves the stack from (1, 1) to (2, 2).
3. Stacks can only move onto cells containing exactly one neutral piece.
4. The game ends if there is no valid moves for both players.
5. Wins the player with the tallest stack or with the most stacks of the tallest height.
6. Follow the prompts and enjoy the game!

Choose board size (N for NxN grid) (3 - 9): 4

Game Modes:
1. Human vs Human
2. Human vs PC
3. PC vs Human
4. PC vs PC
Choose an option (1 - 4): 2

Enter name for Player 1 (Blue): John Doe
Choose difficulty level for PC (1 - 2): 2
```

Menu principal do jogo

Demonstrações

Foram incluídos estados de jogo intermédios para uma demonstração mais rápida. Estes estados de jogo podem ser acedidos através dos seguintes comandos:

- Fase de posicionamento num tabuleiro 5*5 com dois jogadores humanos (Player1 e Player2):

```
? - intermediate_state_placement(GameState), game_loop(GameState, 0).
```

- Fase de movimento num tabuleiro 5*5 com um jogador humano (Player1) e um bot (PCBlue) com apenas alguns movimentos já feitos:

```
?- intermediate_state_movement(GameState), game_loop(GameState, 0).
```

- Fase de movimento num tabuleiro 4*4 com dois jogadores humanos (Player1 e Player2) em que aplicando o movimento (1,4,1,3) o jogo termina em empate, mostrando que os jogadores têm exatamente o mesmo número de stacks com iguais alturas:

```
?- near_final_state_draw(GameState), game_loop(GameState, 0).
```

- Fase de movimento num tabuleiro 4*4 com dois jogadores humanos (Player1 e Player2) em que aplicando o movimento (1,1,1,2) o jogo termina em vitória para o Player 1, mostrando que ganhou o jogador com a stack mais alta:

```
?- near_final_state_blue(GameState), game_loop(GameState, 0).
```

- Fase de movimento num tabuleiro 4*4 com dois jogadores humanos (Player1 e Player2) em que aplicando o movimento (4,4,3,4) o jogo termina em vitória para o Player 2, mostrando que ambos tinham duas stacks com a maior altura e o fator de desempate foi a terceira stack de cada um:

```
?- near_final_state_white(GameState), game_loop(GameState, 0).
```

Conclusões

O jogo foi implementado com sucesso, sendo que foram desenvolvidos 4 modos de jogo (Humano/Humano, Humano/PC, PC/Humano e PC/PC) e para cada jogador PC dois níveis de IA diferentes foram elaborados, um com jogadas aleatórias e outro com jogadas ponderadas visando a vitória.Todo o código foi implementado visando ao máximo uma abordagem declarativa em Prolog e foi todo comentado com a declaração do predicado e de pequenas notas sobre o funcionamento de cada um.

Bibliografia

- Descrição do Jogo - https://boardgamegeek.com/boardgame/425529/staqs

- Exemplos de queries realizadas ao ChatGPT:
  - → Enviou-se o código e uma imagem da consulta do ficheiro no SICStus. -> Explica os problemas que são mostrados na imagem tendo em conta o código disponibilizado.
  - → Como uso o sleep entre as jogadas do computador para demorar um pouco antes da próxima jogada?
  - → Enviou-se as guidelines para o código presentes no guião -> Look at these requirements and tell me if my current src code is following all the requirements or not and what I can still improve, I've noticed for example the use of coordinates starting the (1,1) at lower left corner, which I'm not doing, help me with that.
  - → Ajuda-me a construir alguns game states intermédios e perto do final de jogo para poder demonstrar mais rapidamente algumas regras do meu jogo (these game states can be hard-coded directly into the code file, using predicates similar to the initial_state/2 predicate).
  - → Something in execute_move predicate is not working as expected crashing my sicstus, can you help me finding the problem?