Faculdade de Engenharia da Universidade do Porto

Murus Gallicus

Trabalho Prático 1

Programação Funcional e em Lógica

Turma 11 – Murus Gallicus_6

Rodrigo dos Santos Arteiro Rodrigues(up202108749@fe.up.pt). Contribuição: 60%

Luiz Henrique Mamede Queiroz (up202102362@fe.up.pt). Contribuição: 40%

Porto , 6 de outubro de 2023

Índice

Instalação e Execução

Para instalar o jogo Murus Gallicus primeiro é necessário fazer o download dos ficheiros presentes em PFL_TP1_T11_MurusGallicus_6.zip e descompactá-los. Dentro do diretório src haverá o ficheiro main.pl, dentre outros necessários para o funcionamento do jogo. Será necessário consultar o ficheiro main.pl. Uma forma de consultar o ficheiro é instalar o Sicstus Prolog e utilizar o comando “consult('path_main.pl').”. Após isso, o jogo inicializará ao fazer o comando “play.”. O jogo está disponível em ambientes Windows e Linux.

Descrição do jogo

Murus Gallicus é um jogo de tabuleiros para dois jogadores, o jogador que possui peças brancas que se chama *Romano* e o jogador que possui peças pretas que se chama *Gaulês*. Na sua versão tradicional o jogo é jogado num tabuleiro retangular de 7 linhas por 8 colunas (7x8). Inicialmente, os jogadores deverão decidir quem será o jogador a guiar as peças brancas e quem guiará as peças pretas. Cada jogador tem 16 peças (brancas ou pretas) ao qual se refere como *Pedra* e a união de duas Pedras sobrepostas se chama *Torre*. Para começar o jogo, o tabuleiro é iniciado com as Torres preenchendo completamente a primeira linha e a última linha. O Romano é o jogador a abrir o jogo fazendo o primeiro movimento. Ganha o primeiro jogador a conseguir chegar com uma Pedra à linha inicial do seu adversário ou ganha aquele que deixar seu oponente sem movimentos válidos.
Contudo, existem algumas restrições de movimento:

1. Somente as peças Torres se podem mover pelo tabuleiro;
2. As peças Torres para se moverem no tabuleiro, sem envolver captura de peças adversária, devem avançar para a segunda célula, a célula destino, pulando uma célula, a célula intermédia. Esse movimento pode ser ortogonalmente ou diagonalmente;
3. Quando uma peça Torre for se mover, esta será dividida em duas peças Pedras e essas Pedras serão colocadas na célula destino e na célula intermédia;
4. Para que uma peça Torre se possa mover é preciso que tanto na célula destino como na célula intermédia só possa haver ou peças Pedras da sua cor ou então não haver peça nenhuma;
5. As peças Pedras poderão voltar a ser peças Torres quando, em um movimento da peça Torre, a célula destino ou a célula intermédia já possuírem uma peça Pedra. Nesse caso, após inserir a peça Pedra, vinda da divisão da peça Torre, por cima da já existente peça Pedra numa das células, ela tornar-se-á uma peça Torre.
6. Para um jogador poder capturar peças adversárias, uma peça Pedra do seu adversário tem que estar em uma das células adjacentes a sua peça Torre no tabuleiro. Deste modo, para se realizar a captura será removida a peça Pedra adjacente a sua peça Torre e também será removida da sua peça Torre, uma peça Pedra.

A implementação em Prolog que se segue rege-se através destas regras.
As regras e funcionamento do jogo foram consultadas em: Iggamecenter e em Wikipedia .

Lógica do Jogo

Representação interna do estado do jogo

O estado do jogo é representado da seguinte forma:

- **Board**: é usado lista de listas para representar uma matriz com número de linhas (**NumRow**) fornecido pelo jogador dentro do limite de 6 a 12 linhas. O número de colunas (**NumCol**) será o número de linhas + 1. Contém espaços vazios para representar células disponíveis, X/X ou 0/0 para representar células ocupadas por Torres e X ou O para representar células ocupadas por Pedras.
- **Player:** Indica o atual jogador da rodada e também a peça Torre desse atual jogador. Pode ser o X/X para representar um jogador Romano que controla as peças brancas ou O/O para representar um jogador Gaulês que controla as peças pretas.
- **SoloPeace:** Indica as peças Pedras do atual jogador. Pode ser X ou 0 dependendo se o jogador é X/X ou O/O.
- **Opponent:** Indica o adversário do atual jogador da rodada e também a peça Torre desse adversário. Pode ser o X/X para representar um adversário Romano que controla as peças brancas ou O/O para representar um adversário Gaulês que controla as peças pretas.
- **SoloOpponent:** Indica as peças Pedras do adversário do atual jogador da rodada. Pode ser X ou 0 dependendo se o adversário é X/X ou O/O.

Visualização do estado do jogo

Ao iniciar o jogo, é apresentado o primeiro menu. Ele está disponível com o predicado print_main_menu/0 que irá printar o menu na tela. Com esse menu, é possível escolher o tipo de jogo com opções Player vs Player, Player vs Bot e Bot vs Bot. O usuário irá escolher sua opção que será gravada em GameMode e validada pelo predicado checkInput/1. A validação do input será feita por:

```
repeat,
print_main_menu
read(GameMode),
(checkInput(GameMode) -> true ; write('Invalid option!'), nl, nl, fail).
```

Sendo que o GameMode pode assumir valores 0, 1, 2 ou 3. Valores diferentes desses, receberão mensagem de opção inválida e será necessário escolher um novo valor.

Para a opção Player vs bot, ao ser selecionada, um novo menu é printado. Sendo o predicado utilizado print_choose_level_bot/0. Esse menu serve para escolher o level do bot entre easy e hard. Essa escolha será gravada em BotLevel e a validação do input será feita por:

```
checkInput(2) :-
        print_choose_level_bot,
        read(BotLevel),
        ((BotLevel = 1) -> condição
```

```
;
(BotLevel = 2) -> condição
;
write('Invalid option!'), nl,
checkInput(2)).
```

Sendo que o BotLevel pode assumir valores 1 ou 2. Valores diferentes desses, receberão mensagem de opção inválida, o menu será printado novamente e será necessário escolher um novo valor. Ao escolher um valor de BotLevel aceito, um novo menu será printado para a escolha do tipo de jogador que o usuário quer ser, podendo ser Romano (X/X) ou Gaulês (O/O).

Para a opção Bot vs Bot, ao ser selecionada, um novo menu é printado. Sendo o predicado utilizado print_bot_vs_bot/0. Esse menu serve para o usuário escolher a combinação do tipo de bot (easy/hard) que irão se enfrentar. Essa escolha será gravada em BotsGame a validação do input será feita por:

```
repeat,
print_bot_vs_bot,
read(BotsGame),
(bot_Game(BotsGame) -> true;  write('Invalid option!'), nl, fail).
```

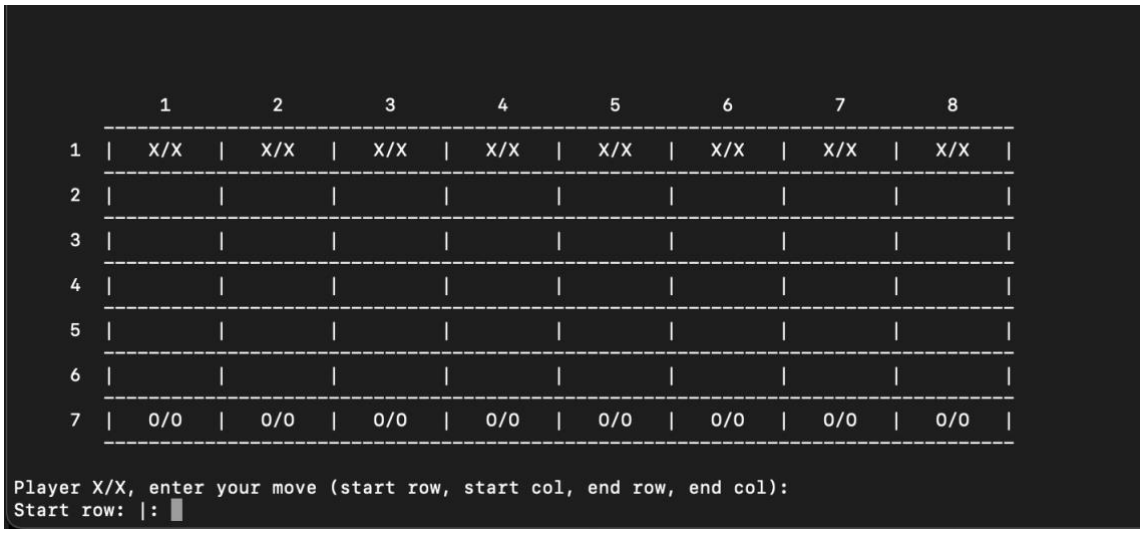
Sendo que BotsGame pode assumir valores 1, 2, 3 ou 4. Valores diferentes desses, receberão mensagem de opção inválida, o menu será printado novamente e será necessário escolher um novo valor.

Ao fim das escolhas, um último menu é printado para todas as opções. Esse menu serve para a escolha do tamanho do tabuleiro. Será escolhido o número de linhas do tabuleiro e consequentemente o número de colunas será o número de linhas + 1. O predicado para printar esse menu é print_board_row/0. O predicado para a escolha do número de linhas é o initial_board(-NumRow, -NumCol, -Board). Essa escolha é gravada em R e a validação do input é feita por:

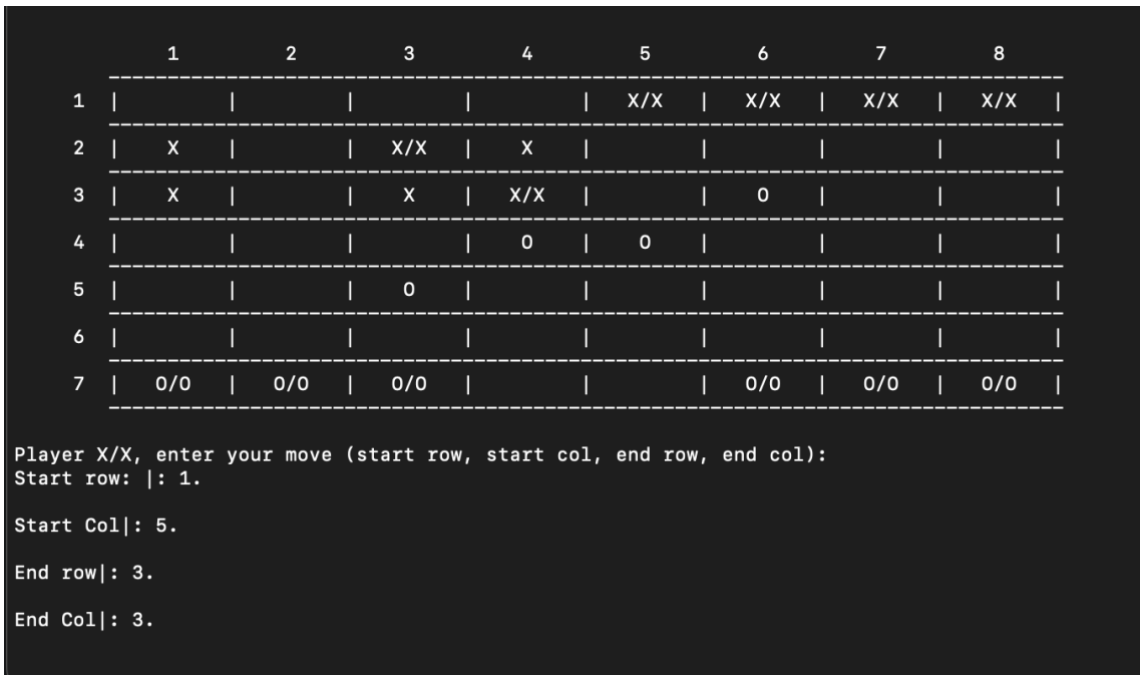
```
repeat,
print_board_row,
write('Nrows'),
read(R),
rowLimits(MinRow,MaxRow),
(((R >= MinRow),(R =< MaxRow));
format('The number of rows must be between ~w and ~w', [MinRow, MaxRow]), nl, fail),
C is R + 1,
board(R, C, Board).
```

Sendo que R pode assumir valores entre MinRow e MaxRow. Foram usados os valores 6 e 12 para MinRow e MaxRow, respectivamente. Valores fora desse intervalo, receberão mensagem informando o intervalo de valores possíveis de número de linhas, o menu será printado novamente e será necessário escolher um novo valor.
Assim, o board inicial é gerado pelo predicado board/3 e é usado o predicado display_board(+Board) para printar o board no ecrã.

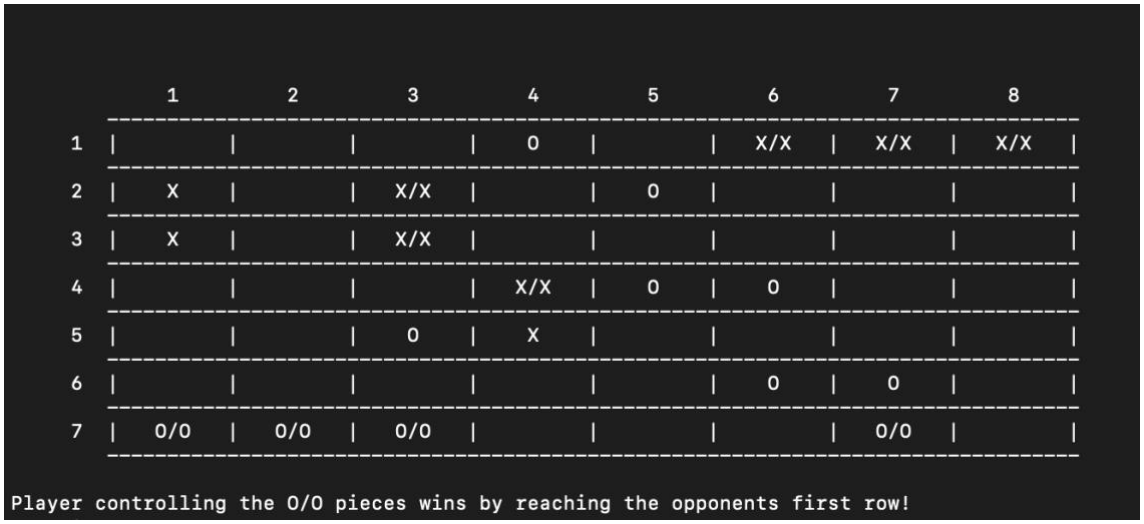
Estado do jogo inicial:



Estado do jogo intermédio:



Estado do jogo final:



Validação do movimento e Execução

A execução de uma jogada se inicia no predicado get_move/12. Esse predicado obtém o movimento que o jogador pretende realizar. Para isso, o jogador insere os seguintes inputs:

- **StartRow:** Indica a linha da posição referente a peça que o atual jogador quer mover.
- **StartCol:** Indica a coluna da posição referente a peça que o atual jogador quer mover.
- **EndRow:** Indica a linha da posição de destino refere ao movimento que o jogador quer realizar.
- **EndCol:** Indica a coluna da posição de destino refere ao movimento que o jogador quer realizar.

Esses inputs são validados pelo predicado valid_move/12 que faz a validação do movimento e verifica se é um movimento válido de acordo com as regras do jogo. Caso o movimento não seja válido, é printado no ecrã o motivo pelo qual o movimento não é válido e é requerido que o jogador insira novos inputs para validação. Os casos de movimentos inválidos são:

- Quando as coordenadas ultrapassam os limites do tabuleiro
- Quando o jogador seleciona uma peça para se mover que não é sua ou não é uma Torre
- Quando a célula de destino já possui alguma peça
- Quando a célula intermédia possui uma peça adversária bloqueando o movimento
- Quando o jogador tenta fazer uma captura, mas não há peça adversária para capturar
- Quando as coordenadas inseridas não servem nem para captura e nem para locomover a peça duas células da posição inicial

Somente quando o jogador insere um movimento válido que é verificado se é uma captura ou se é para mover a peça. Se for uma captura, é utilizado o predicado capture_piece/8 para atualizar o tabuleiro, se for para mover a peça, é utilizado o predicado move/8. Depois de atualizar o tabuleiro com o movimento válido, o novo tabuleiro é printado na tela com o predicado display_board/1 e é verificado se o jogo terminou ou se irá para a próxima rodada através do predicado game_over/11.

Lista de movimentos válidos

A lista de movimentos válidos é obtida através do predicado valid_moves/7. Ele utiliza o findall/3 com o predicado valid_move_bot/12 para encontrar os movimentos válidos que o bot pode fazer. Para um humano, por questões de eficiência, é usado diretamente o predicado valid_move/12 para verificar se o movimento é válido em vez de calcular todos os movimentos possíveis. A lista de movimentos válidos é uma lista da forma [DeltaRow, DeltaCol, StartRow, StartCol, EndRow, EndCol] que informa a posição de início e a posição de destino pelo número da linha e coluna, além de possuir informação sobre a variação (delta) dessa posição tanto para linha quanto para coluna.

Fim do jogo

A verificação do fim de jogo é feita pelo predicado game_over/11. Essa verificação é feita ao final de cada rodada para saber se o jogo será encerrado ou se começará a nova rodada. Ela verifica se a posição de destino inserida pelo atual jogador da rodada coincide com a primeira linha do adversário e se ele está fazendo um movimento ou somente uma captura, já que se ele estiver em captura, ele não se move e portanto o jogo não chegaria ao fim. Caso o jogo não tenha ainda vencedor, o predicado play_game/8 é chamado novamente e assim o loop é criado e só será encerrado quando houver um vencedor. É importante notar também que em play_game/8, uma primeira verificação já é feita com o predicado has_value_in_sublists/2 que verifica no tabuleiro se ainda existe peça Torre correspondente ao atual jogador da rodada. Se não existir, o jogo é encerrado pelo jogador não possuir movimentos válidos e seu oponente é o vitorioso.

Avaliação do estado do jogo

A avaliação do estado do jogo é feita no início de cada rodada pelo predicado has_value_in_sublists/2 onde é possível verificar se o jogador atual ainda possui peças Torres no tabuleiro para realizar movimentos. E é feita também ao final de cada rodada, para saber se o jogador atual chegou na primeira linha do seu oponente.

Jogadas do computador

O computador tem 2 modos de dificuldade. O modo aleatório e o modo estratégico. No modo aleatório é utilizado o predicado generate_random_valid_move/12. Ele gera uma lista com os movimentos válidos utilizando o predicado valid_moves/7, gera um número aleatório entre 1 e o tamanho da lista e obtém dessa lista o movimento com o índice da lista correspondente ao número aleatório gerado através do predicado get_random_move/3.

No modo estratégico, ele gera uma lista com os movimentos válidos utilizando o predicado valid_moves/7. A partir dessa lista, será escolhido, de forma mais estratégica, o movimento de

acordo com certas prioridades. Essa escolha será feita no predicado execute_valid_move/7. As prioridades do movimento são: selecionar um movimento que chegue na primeira linha do oponente, selecionar um movimento que capture uma peça do oponente e, por fim, selecionar um movimento aleatório.

Conclusão

O jogo Murus Gallicus foi implementado com sucesso em Prolog. O jogo pode ser jogado em 3 diferentes modos: Player vs Player, Player vs Bot e Bot vs Bot, com os bots tendo 2 níveis de dificuldades possíveis. O tabuleiro do jogo possui flexibilidade de tamanho, podendo ser ampliado ou diminuído dentro de um intervalo de valores para o número de linhas.

Com a implementação do jogo, foi possível colocar em prática os conhecimentos adquiridos e aprimorar a capacidade de programação no paradigma lógico.

Por fim, o jogo funciona como esperado e todos os modos propostos foram realizados.

Bibliografia

https://www.iggamecenter.com/en/rules/murusgallicus
https://en.wikipedia.org/wiki/Murus_Gallicus_(game)
https://www.educba.com/prolog-programming/
https://www.geeksforgeeks.org/lists-in-prolog/