Trabalho prático 2 - 4Mation

FEUP - PFL 2021/2022

Grupo 4Mation_5:

Joel Alexandre Vieira Fernandes `up201904977@edu.fe.up.pt` (45%)
Pedro Gonçalo de Castro Correia `up201905348@edu.fe.up.pt` (55%)

23 de janeiro de 2022

Conteúdo

1 Instalação e Execução

Para executar o jogo deve-se executar o SICStus Prolog 4.7 no diretório *src/* e usar o comando [*load*]. . O menu de jogo poderá ser aberto com o predicado *play*/0.

2 Descrição do jogo

Em 4Mation, dois jogadores colocam à vez uma peça no tabuleiro de maneira a criar uma linha, coluna ou diagonal de 4 peças jogadas por si. Nesta implementação do jogo, as peças do jogador 1 serão representadas por 'X' e as do jogador 2 por 'O'. Para além disso, o número necessário de peças consecutivas para ganhar o jogo é configurável, embora seja por padrão 4.

O tamanho do tabuleiro é definido antes de iniciar o jogo. O primeiro jogador pode colocar uma peça em qualquer posição do tabuleiro. Em todas as jogadas seguintes, a peça do jogador que está a jogar tem de ser colocada numa posição adjacente (ortogonalmente ou diagonalmente) à peça colocada na jogada do jogador anterior.

O jogo termina quando um dos jogadores consegue formar uma linha, coluna ou diagonal de 4 peças suas, sendo considerado o vencedor, ou quando não há nenhuma jogada possível, sendo considerado empate.

Mais informações, bem como a descrição do jogo feita pela editora, podem ser encontradas na página do jogo na BoardGameGeek.

3 Lógica do Jogo

3.1 Representação interna do estado do jogo

O estado de jogo é representado pela estrutura de dados:

```
game_state(
    Board,
    current_player(Player),
    last_move(Move),
    players(Px, Po),
    win_target(Goal)
)
```

Board é uma lista de listas que representa o tabuleiro, *Player* é o jogador a realizar a próxima jogada (*player_x* ou *player_o*), *Move* é a posição da última jogada (*none*, quando ainda não houve jogadas; ou $position(X, Y)$ sendo X e Y os índices começados em 0 da coluna e linha no tabuleiro, respetivamente), Px e Po indicam se o jogador respetivo é humano ou computador (*human*, $bot(1)$ ou $bot(2)$) e *Goal* é o número de peças consecutivas necessárias para ganhar o jogo (um número inteiro positivo).

Cada célula do tabuleiro pode ter como valores *empty*, *player_x* ou *player_o*, respetivamente para se estiver vazio, com uma peça 'X' ou uma peça 'O'.

Exemplo de estado inicial de jogo:

```
game_state(
    [
        [empty, empty, empty, empty],
        [empty, empty, empty, empty],
        [empty, empty, empty, empty],
        [empty, empty, empty, empty],
    ],
    current_player(player_x),
    last_move(none),
    players(human, bot(2)),
    win_target(3)
)
```

Exemplo de estado intermédio de jogo:

```
game_state(
    [
        [empty, empty, empty, empty],
        [empty, player_x, player_o, empty],
        [empty, player_x, empty, empty],
        [empty, empty, player_o, empty],
    ],
    current_player(player_x),
    last_move(position(3, 2)),
    players(human, bot(2)),
    win_target(3)
)
```

Exemplo de estado final de jogo (como há uma linha vertical com 3 'X' e *win_taget*(3), o predicado *game_over*/2 irá detetar o jogador *player_x* como vencedor):

```
game_state(
    [
        [empty, empty, empty, empty],
        [empty, player_x, player_o, empty],
        [empty, player_x, empty, empty],
        [empty, player_x, player_o, empty],
    ],
    current_player(player_o),
    last_move(position(3, 1)),
    players(human, bot(2)),
    win_target(3)
)
```

Antes do início do jogo, o estado do menu é mantido com a estrutura de dados:

```
menu_state(
    Menu,
    config(
        size(Cols, Rows),
        players(Px, Po),
        first_player(Player),
        win_target(Goal)
    )
)
```

Onde *Px*, *Po* e *Goal* têm o mesmo significado que no estado de jogo. *Player* é o jogador a realizar a primeira jogada (*player_x* ou *player_o*). *Cols* e *Rows* é o número de colunas e de linhas no tabuleiro, respetivamente. *Menu* é o menu aberto atualmente (por exemplo, *main*, *rules*, *config*, *game_over*(*Winner*) onde *Winner* é o vencedor do jogo ou *none* em caso de empate).

Exemplo de estado de menu:

```
menu_state(
    main,
    config(
        size(4, 4),
        players(human, bot(2)),
        first_player(player_x),
        win_target(3)
    )
)
```

3.2 Input e output

Foram implementados predicados auxiliares genéricos para facilitar a escrita de elementos no ecrã, nomeadamente caixas de texto retangulares com borda (*print_banner*(+*Lines*, +*WidthPadding*, +*HeightPadding*, +*Margin*), com um conjunto de linhas, *padding* e margem), um conjunto de N caracteres de espaço (*margin*(+N)) e um átomo repetido N vezes (*print*$_n$(+S, +N)).

Um menu é aberto com o predicado *open_menu*(+*MenuState*). Este predicado usa *display_menu*(+*MenuState*) para mostrar o conteúdo do menu, *read_response*(+*State*, −*Action*) para ler a resposta do utilizador e obter a ação a realizar e *do_menu_action*(+*MenuState*, +*Action*) para executar essa ação.

O jogo é iniciado com o predicado *play_game*(+*Config*), que obtém o estado inicial do jogo com *initial_state*(+*Config*, −*GameState*) e inicia o primeiro turno com *play_turn*(+*GameState*, +*Config*). Por sua vez, este predicado mostra o estado do jogo com *display_game*(+*GameState*), deteta se o jogo terminou (e nesse caso abre o menu de fim do jogo), no caso de o jogador atual ser um humano lê a resposta do utilizador com *read_response*(+*State*, −*Action*) e executa a ação obtida com *do_game_action*(+*GameState*, +*Action*), e no caso de o jogador atual ser o computador, aguarda um segundo, determina a jogada a fazer e executa-a, passando ao turno seguinte.

A função *read_response*(+*State*, −*Action*) lê e valida a input do utilizador, apresentando uma mensagem de erro apropriada e lendo nova input se for introduzida uma input inválida. As respostas válidas possíveis e respetivas ações num dado estado do menu ou de jogo são definidas pelo predicado *response*(+*State*, +*Response*, −*Action*). As ações são

executadas nos predicados *do_menu_action*(+*MenuState*, +*Action*) e *do_game_action*(+*GameState*, +*Action*).

3.3 Visualização do estado de jogo

O predicado *display_game*(+*GameState*) recebe o estado atual do jogo, mostra o tabuleiro conforme esse estado e indica qual o jogador que deve efetuar a próxima jogada. A visualização do tabuleiro é implementada com o predicado *display_board*(+*Board*, +*ValidMoves*) e respetivas funções auxiliares (por exemplo, *display_board_row*(+*Row*, +*RowNum*, +*ValidMoves*) para mostrar uma linha do tabuleiro, *display_board_bar*(+*Cols*) para mostrar uma barra de separação entre cada linha, *display_board_header*(+*Cols*) para mostrar o cabeçalho e *display_board_body*(+*Rows*, +*ValidMoves*) para mostrar as várias linhas do tabuleiro separadas por barras). Para cada linha, as jogadas válidas são marcadas com *legal_move* através do predicado *mark_valid_moves*(+*Row*, +*RowNum*, +*ValidMoves*, −*RowWithLegalMoves*) e depois, para cada posição, é mostrado um ' ' se for *empty*, um 'X' se for *player_x*, um 'O' se for *player_o* ou um '.' se for *legal_move*.

3.4 Execução de Jogadas

O predicado que efetua uma jogada chama-se *move*(+*GameState*, +*Move*, −*NewGameState*), onde *GameState* é o estado de jogo atual, *Move* é a jogada a efetuar (*position*(*X*, *Y*) sendo *X* e *Y* o índice começado em 0 da linha e da coluna no tabuleiro, respetivamente) e *NewGameState* o estado do jogo após a jogada.

Este predicado recorre a três predicados auxiliares internamente: *next_player*(+*Player*, −*PlayerNext*) retorna o próximo jogador a jogar (o oponente de *Player*); *valid_move*(+*Board*, +*Last*, +*Move*) que indica se *Move* é uma jogada válida tendo em conta o tabuleiro *Board* e a última jogada *Last* (certifica-se que a posição está vazia e que, se houve uma jogada *Last*, a nova jogada é adjacente a esta); *move_board*(+*Board*, +*Move*, +*Player*, −*BoardNext*) coloca a peça *Player* na posição *Move* do *Board*, retornando o novo tabuleiro em *BoardNext*.

3.5 Final do Jogo

O predicado que verifica o fim do jogo e identifica o vencedor é *game_over*(+*GameState*, −*Winner*). *Winner* pode ser *player_x*, *player_o* ou *none* para caso de empate. O empate ocorre quando a lista de jogadas válidas é vazia. A verificação de se um jogador venceu é feita no predicado auxiliar *wins_game*(+*Board*, +*Winner*, +*Goal*), com respetivamente o tabuleiro, o jogador e o número de peças consecutivas para vencer. Por sua vez, este recorre a predicados auxiliares para testar se há *Goal* − 1 peças à direita de alguma peça de *Winner*, em baixo, na diagonal superior direita ou na diagonal inferior direita. Apenas é necessário testar nestas direções porque, se houvesse uma peça anterior (por exemplo, à esquerda), o vencedor já teria sido detetado no teste (nesse exemplo, à direira) a essa peça.

3.6 Lista de Jogadas Válidas

A lista de jogadas possíveis é obtida pelo predicado *valid_moves*(+*GameState*, −*ListOfMoves*). Este predicado reutiliza a função auxiliar *valid_move*(+*Board*, +*Last*, +*Move*) de validação de uma jogada, usando um *findall*/3 para obter a lista de todas as jogadas válidas.

3.7 Avaliação do Estado do Jogo

A avaliação do estado de jogo no ponto de vista de um jogador é quantificada através do predicado $value(+GameState, +Player, -Value)$. Esta avaliação é usada pelo computador para decidir de entre a lista de jogadas válidas qual deve executar. Como tal, apenas é necessário que esta quantificação compare os estados de jogo que diferem apenas na última jogada efetuada, que é a jogada que pode ser decidida.

No caso geral, a avaliação da jogada é feita em cada direção (horizontal, vertical e ambas as diagonais) e é somada a avaliação de todas as direções. Para cada direção, a fórmula é $1+2+3+...+n$, sendo n o número de peças de $Player$ consecutivas nessa direção (incluindo a peça colocada). A contagem em cada direção é feita recorrendo duas vezes ao predicado auxiliar $value\_counter(+Board, +Position, +Direction, +Player, +Inc, -Result, -ResultInc)$, que percorre o tabuleiro a partir da posição dada e na direção dada até encontrar uma peça diferente da de $Player$, incrementando em cada iteração o valor de Inc em uma unidade (retornado em $ResultInc$) e somando ao resultado acumulado o valor de Inc nessa iteração. O primeiro uso do predicado é feito num sentido, começando na posição da última jogada com $Inc = 1$, e o segundo no sentido oposto começando já na posição a seguir à da última jogada (usando o valor de $IncResult$ do primeiro como valor inicial de Inc).

Uma exceção a esta regra é quando o oponente pode vencer na sua próxima jogada. Como se deve evitar a todo o custo dar a vitória ao adversário, nestes casos o valor do estado do jogo é -1. A outra exceção é quando $Player$ venceu o jogo no estado atual. Neste caso (que se sobrepõe à exceção anterior) a jogada deve sempre ser escolhida uma vez que o objetivo é vencer o jogo. Como tal, o valor é alto suficiente para ser superior ao de qualquer outra jogada, neste caso $1 + n * (n + 1) + m * (m + 1)$ onde m é o número de linhas e n o número de colunas no tabuleiro. Este número foi escolhido para garantir que é maior que o valor máximo que se pode obter num tabuleiro $n \times m$ (somando o valor máximo em cada direção):

$$
\begin{aligned}
&\sum_{i=1}^{n} i \; + \; \sum_{i=1}^{m} i \; + \; 2 \times \sum_{i=1}^{min(n,m)} i \; = \\
&\quad = \frac{n \times (n+1)}{2} + \frac{m \times (m+1)}{2} + 2 \times \frac{min(n,m) \times (min(n,m)+1)}{2} \\
&\quad \leq \frac{n \times (n+1)}{2} + \frac{m \times (m+1)}{2} + \frac{n \times (n+1) + m \times (m+1)}{2} \\
&\quad = 2 \times \frac{n \times (n+1)}{2} + 2 \times \frac{m \times (m+1)}{2} \\
&\quad = n \times (n+1) + m \times (m+1) \\
&\quad < 1 + n \times (n+1) + m \times (m+1)
\end{aligned}
\tag{1}
$$

3.8 Jogada do Computador

A escolha da jogada a efetuar pelo computador é feita através do predicado $choose\_move(+GameState, +Level, -Move)$, onde $Level$ é o nível de dificuldade que pode ser 1 ou 2.

Quando $Level = 1$, é obtida a lista de jogadas válidas com $valid\_moves(+GameState, -List)$ e selecionado um elemento aleatório da lista com $random\_member(-Member, +List)$.

Quando $Level = 2$, o computador escolhe a melhor jogada no momento, usando um algoritmo ganancioso. É calculado o valor do estado de jogo após cada jogada na lista de jogadas válidas e estas jogadas são ordenadas pelo valor respetivo, com recurso a $setof/3$. De modo a obter a jogada com valor mais elevado, é retornada a última jogada na lista retornada por $setof$. Para obter o último elemento é usado o predicado auxiliar $choose\_last\_move(+List, -Last)$.

4 Conclusão

Neste trabalho foi implementado o jogo de tabuleiro 4Mation, permitindo jogos Humano vs. Humano, Humano vs. Computador e Computador vs. Humano. Tentou-se modularizar a implementação, recorrendo a múltiplos ficheiros (ficheiro *load.pl* principal, ficheiro *board.pl* para predicados utilitários do tabuleiro, ficheiro *game.pl* com as funções que implementam a lógica do jogo e *io.pl* para input, output e o ciclo do jogo). Foi tido em atenção que a visualização e interação do jogo e dos menus fossem apelativas e intuitivas e que inputs fossem validadas, mostrando mensagens de erro se necessário. Foram escolhidas representações de estado de jogo e de menus flexíveis e configuráveis, funcionando para qualquer tamanho do tabuleiro, jogador inicial e condição para vencer. O predicado $initial\_state(+Config, -GameState)$ devolve o estado de jogo inicial para as configurações dadas. Foram implementados dois níveis de dificuldade para as jogadas do computador, um que devolve uma jogada aleatória e outro que recorre a um algoritmo ganancioso que usa o resultado de $value(+GameState, +Player, -Value)$ como heurística para escolher uma jogada.

Uma possível melhoria que poderia ser explorada no futuro seria implementar mais níveis de dificuldade do computador, que em vez de escolher a melhor jogada no momento utilizaria o algoritmo https://en.wikipedia.org/wiki/Minimax para maximizar o ganho mínimo tendo em conta as jogadas possíveis nos N turnos seguintes (e considerando o valor das jogadas do oponente negativas).

5 Bibliografia

"4Mation", BoardGameGeek, https://boardgamegeek.com/boardgame/329175/4mation. Acedido a 20/12/2022.