Projeto de Programação Lógica

Projeto desenvolvido no ambito da unidade curricular de Programação Funcional e em Lógica.

22 de Janeiro de 2022

Trabalho Prático 2: JOSTLE

O nosso trabalho consistiu na implementação do jogo "Jostle", criado por Mark Steere, recorrendo para isso à linguagem Prolog com execução em terminal.

Grupo T2_Jostle_3:

Nome do Estudante	Número mecanográfico	Contribuição
Francisco Renato Barbosa Pires	up201908044	50%
Alberto José Ribeiro da Cunha	up201906325	50%

Instalação e Execução

Esta implementação assume apenas a instalação base de SICStus Prolog.

Para iniciar o jogo deve mudar o "Working Directory" para a pasta /src, consultar o ficheiro main.pl e executar o predicado play().

Descrição do jogo

As regras detalhadas do jogo podem ser consultadas na página oficial do seu criador em http://www.marksteeregames.com/Jostle_Go_rules.pdf

Trata-se de um jogo para duas pessoas, jogado num tabuleiro 10 por 10 na qual cada jogador tem 16 peças semelhantes a damas. Em turnos, os jogadores devem mover uma das suas peças para um espaço adjacente livre (apenas nas direções ortogonais: cima; baixo; esquerda e direita) que dê á sua peça um valor de **ligações adjacentes** mais elevado do que o que tinha anteriormente, se o valor for igual ou inferior a jogada não é permitida.

O valor de **ligações adjacentes** define-se pelo número de peças adjacentes pertencentes ao mesmo jogador menos o número de peças adjacentes adversárias.

Ganha o último jogador a efetuar uma jogada, i.e. o primeiro jogador a não poder mover nenhuma peça perde.

Lógica do Jogo

Nas secções seguintes apresentaremos as partes relevantes da lógica de funcionamento do nosso jogo.

Para o inicio de jogo existe o predicado **play()** que, utilizando predicados para apresentação de diversos menus, pergunta ao utilizador qual os modos de jogo desejados e inicia o ciclo de jogo de acordo com essas escolhas.

Representação interna do estado do jogo

Neste jogo todas as peças encontram-se no tabuleiro desde o inicio e não há captura de peças, tendo isto em conta o nosso estado de jogo consiste numa lista de 32 elementos contendo informação sobre a posição de cada peça, visto cada jogador ter 16 peças.

Para facilitar a consulta de peças pela sua posição e vice-versa, bem como facilitar a inserção de novos elementos com novos dados, optámos por guardar cada elemento na lista com o formato Cor-Indice-Posição.

Cor - Representa a cor do jogador, vermelho ou azul, representado pelos átomos 'r' ou 'b'.
Indice - Para distinguir entre as 16 peças de um jogador usamos um indice de 0 a 15.

Posição - Um valor inteiro que representa a posição no tabuleiro, começando em 0 e seguindo a convenção semelhante à de leitura de um texto, da esquerda para a direita e depois em linhas de cima pra baixo. Isto tem o efeito de no tabuleiro 10 por 10 o primeiro algarismo representar o indice das linhas e o segundo as colunas.

Um exemplo de um elemento poderia ser r-5-34, que representa a peça vermelha de indice 5 (i.e. a sexta peça do jogador 1) na posição 34, que no nosso tabuleiro fica na quarta linha e quinta coluna.

Isto permite um fácil acesso aos dados do tabuleiro e não necessita que se mantenha uma ordem dos elementos na lista.

O predicado **initial_state(-GameState)** devolve o estado inicial do jogo numa lista tal como indicado nas regras do jogo.
Outro predicado **board(+BoardIndex, -GameState)** devolve ainda outros estados de jogo com o propósito de servirem para demonstração de estados mais avançados da partida. Em alternativa ao predicado **play/0** existe o **play_test_custom_board(+X)** com a qual se podem utilizar estes tabuleiros alternativos usando para isso o mesmo indice do predicado **board/2**.

Visualização do estado de jogo

Para a visualização do estado do tabuleiro implementámos o predicado **display_game(+Size, +GameState, +Player)** que mostra ao utilizador uma representação do mesmo usando caracteres ASCII.

Este predicado constrói uma representação do tabuleiro com número Size de quadrados de lado e coloca as peças no tabuleiro de forma a refletir o estado de jogo passado em GameState. Ainda mostra ao utilizador de qual jogador é a vez de jogar.

Existem também outros predicados tais como **display_start_screen/2** e **display_start_menu/2** que apresentam menus simples ao utilizador de modo a que este possa escolher os modos de jogo e dificuldade desejados.

Execução de Jogadas

Para a execução de jogadas recorremos ao predicado **move(+GameState, +From-To, -NewGameState)** que para um determinado movimento, definido pela posição da peça e a nova posição indicada no formato From-To, e um estado de jogo GameState retorna um novo estado em NewGameState. A validação da jogada é feita no momento da recolha do input do utilizador por comparação com a lista de jogadas possíveis criada por **valid_moves/2**.

Final do Jogo

O predicado **game_over(+GameState, +Player)** recorre a **valid_moves(+GameState, -ListOfMoves, +Player)** para gerar a lista de possíveis jogadas, se esta for vazia retorna verdade uma vez que o fim do jogo se define pelo momento em que um utilizador fica sem jogadas.

Lista de Jogadas Possíveis

Este predicado é usado na deteção do fim do jogo e para apresentar ao jogador as suas opções de movimento e validar a legalidade da escolha do utilizador.

O **valid_moves(+GameState, -ListOfMoves, +Player)** gera uma lista de todos os possíveis movimentos do jogador especificado.

Para isso, consulta a posição de cada peça do jogador em GameState e compara o valor de **ligações adjacentes** nessa posição e nas 4 direções possíveis em que se pode mover a peça. Com base nestes valores acrecenta ou não a nova jogada à lista de possíveis jogadas.

Cada elemento da lista segue o mesmo formato de movimentos aceite pelo predicado **moves/3** (From-To), em que:

From - é o indice da posição da peça que se pretende mover.

To- posição para onde se pretende movimentar a peça.

Avaliação do Estado do Jogo

Através do predicado **value(+GameState, +Player, -Value)** é possível obter a soma do valor de **ligações adjacentes** de todas as peças de um jogador.
Este valor permite ter uma ideia de qual dos jogadores está a ganhar, visto que aquele que tiver o valor mais elevado num determinado estado de jogo deverá se encontrar mais perto de ficar sem jogadas restantes.

Jogada do Computador

O predicado **choose_move(+GameState, +Level, -Move, +Player)** escolhe um movimento com base no estado do jogo e o indice de jogador atribuido.

Esta escolha varia consoante o nível escolhido (1 ou 2):

- No nível 1 é simplesmente escolhido um movimento aleatoriamente de entre os possíveis indicados pelo predicado **valid_moves/3**.
- No nível 2 a escolha é feita com base no predicado **best_move(-Move, +GameState, +Moves, +Player)** que faz uso do predicado **evaluate_move/3** para identificar qual das jogadas possíveis tem o menor aumento do valor de **ligações adjacentes**. Visto que no cenário de fim de jogo o valor de cada peça será o mais elevado possível, levará mais tempo pra chegar a essa situação quanto menor for o aumento em cada jogada.

Conclusões

Através da realização deste projeto ficámos a conhecer melhor a forma correta de programar em prolog.

Após terminar o projeto apercebemo-nos de que alguns dos predicados que decidimos alterar os argumentos podiam, com mais algum esforço e planeamento, ter sido implementados exatamente como estavam descritos no enunciado.

O nosso projeto acabou por ficar um pouco limitado na construção do estado inicial do tabuleiro em que são geradas sempre as posições das peças assumindo um tabuleiro de 10 por 10. Apesar do **display_game/1** suportar tabuleiros de tamanho arbitrário acabámos por não poder fazer uso do mesmo por não termos a restante lógica de funcionamento generalizada para tabuleiros de diferentes tamanhos.

Numa continuação deste projeto um ponto a melhorar seria a generalização destes predicados.

Bibliografia

Na implementação deste projeto, além de consultarmos os contéudos teóricos de programação lógica disponibilizados através da plataforma moodle da unidade curricular em https://moodle.up.pt/course/view.php?id=4031#section-1, fizemos também uso dos seguintes recursos:

Moura, Paulo. "SWI-prolog - how to clear terminal screen with a keyboard shortcut or global predicate?" Em StackOverflow a 12 de Novembro de 2018, https://stackoverflow.com/questions/53262099/swi-prolog-how-to-clear-terminal-screen-with-a-keyboard-shortcut-or-global-pre

SICStus4. "The Prolog Library" em https://sicstus.sics.se/sicstus/docs/4.3.0/html/sicstus/The-Prolog-Library.html#The-Prolog-Library

Todos estes endereços estavam acessíveis á data de escrita deste documento, 22 de Janeiro de 2022.