GREEN SKULL

Identificação do Jogo: Green Skull
Identificação do Grupo:
- Diogo Santos (up201806878)
- Tomás Gonçalves(up201806763)

Instalação

Para poder instalar o jogo, basta fazer o download dos ficheiros, e, através do sicstus(ou outro ambiente de prolog) fazer consult do ficheiro *green_skull.pl*. De seguida, basta escrever o predicado *play* para o jogo começar.

Descrição

Introdução

Green Skull é um jogo de tabuleiro disputado por dois jogadores. O material necessário para jogar é: um tabuleiro, peças redondas (8 verdes, 10 roxas e 10 brancas) e uma peça não redonda (geralmente em formato de crânio).

tabuleiro_real

O tabuleiro é de formato triangular sendo que cada lado contém 10 células, e cada um destes lados apresenta bordas de cores correspondentes aos três tipos de peças.

Os três tipos de peças representam diferentes criaturas mitológicas (embora a representação seja apenas abstrata). As peças verdes são chamadas de Zombies, as brancas de Orcs e as roxas de Goblins.

Instruções de jogo

Ao iniciar o jogo, um jogador toma posse dos Goblins enquanto que o adversário controla os Orcs. O controlo das peças Zombie vai alternando entre os dois jogadores à medida que vão jogando (embora seja primeiro atribuído ao jogador que possui os goblins).

Ambos os jogadores podem efetuar um de dois movimentos possíveis: -O primeiro é deslocar uma das suas peças para uma casa adjacente vazia.

movimento1

-Pode também efetuar um ou mais saltos em linha reta sobre outra peça (incluindo peças do próprio jogador) caindo numa casa vazia. Estas peças pelas quais a peça vai passando por cima vão sendo removidas do tabuleiro. Se o jogador que possui o crânio optar por este segundo movimento, deve ceder o crânio ao seu adversário tendo agora ele posse dos Zombies.

movimento2

O jogador que possui o crânio pode ainda mover uma das peças zombies após cada jogada com as suas respetivas peças.

O jogo termina quando todas as peças de um tipo forem comidas ou estiverem em contacto com a borda da mesma cor.

Pontuação

Vence a espécie que obtiver mais pontos de acordo com a seguinte contagem:

-cada espécie recebe 2 pontos por cada peça que toque a borda da sua cor -cada espécie recebe 1 ponto por cada peça capturada que não seja da sua cor

Página do Jogo: https://nestorgames.com/rulebooks/GREENSKULL_EN.pdf

Lógica do Jogo

Representação interna do estado do jogo

O estado do jogo é representado através da estrutura *state(Board, Player, ZPlayer, XEliminated, OEliminated, ZEliminated)*.

state

Nesta estrutura Board representa-se o tabuleiro de jogo, em Player representa o jogador atual(*X*, *O* ou *Z*), ZPlayer representa o jogador que detém posso sobre a *green skull* (*X* ou *O*), ou seja, sobre os zombies, XEliminated representa o número de peças X eliminadas do tabuleiro, OEliminated representa o número de peças O eliminadas do tabuleiro e finalmente ZEliminated representa o número de peças Z eliminadas do tabuleiro.

O tabuleiro triangular é representado através de uma lista de listas com a seguinte disposição:

Estado inicial do jogo: tabela_inicial

O tabuleiro é criado com as peças nas suas posições iniciais. As peças Zombie são representadas visualmente pela letra "Z", Goblins por "O" e Orcs por "X".

Os três tabuleiros seguintes representam estados de jogo intermédios. Podemos observar algumas peças movidas e outras retiradas do tabuleiro.

Estados intermédios do jogo: tabela_intermedia

Esta última tabela mostra-nos o estado final do jogo. O jogo terminou pois todas as peças Zombie ("Z") estão em contacto com a face do tabuleiro correspondente à sua cor (neste caso a base do triângulo).

Estado final do jogo: tabela_final

Guardamos a informação de qual o jogador que deve jogar através do facto que vamos alterando dinamicamente chamado player_turn e guarda uma string que indica qual é o jogador a jogar. Da mesma forma, representamos o detentor do crânio verde com o facto z_belongs_to que também contém uma string que nos diz qual dos dois jogadores controla os zombies nesse instante. Em relação a peças eliminadas, temos também três factos que alternam dinamicamente: o_eliminated, z_eliminated e x_eliminated; estes factos contêm o número de peças eliminadas da sua espécie. Sempre que alguma destas peças é eliminada no respetivo facto, incrementamos o valor de peças eliminadas. Em relação à representação das peças, são utilizadas strings. Como descrito anteriormente, cada uma situada na sua posição específica do tabuleiro.

Visualização do Estado de jogo

Para se poder visualizar o estado atual do jogo temos de recorrer ao predicado display_game/2 que recebe como primeiro parâmetro o estado atual do jogo e como segundo o jogador atual.

display_game

Este predicado recorre a outros três predicados que fazem o seguinte:

- display_board/1: recebe como parâmetro o tabuleiro atual de jogo e apresenta também as linhas e colunas de cada elemento para facilitar a escolha da peça a mover, tal como a casa para onde a mesma deve ser movida. Este predicado recorre a outro predicado "print_row1/1" que imprime a linha 1. Este, por sua vez chama o "print_row2/1" que imprime a linha 2 e assim sucessivamente até à linha 10. Na imagem que se segue, é possivel observar a representação do tabuleiro inicial do jogo.

tabuleiro_jogo

- display_number_eliminated/4: recebe nos três primeiros parâmetros o número de peças eliminadas de cada tipo e, no quarto argumento, recebe o jogador que detém posse sobre os zombies. Este predicado permite apresentar uma tabela indicativa do número de peças eliminadas de cada espécie e indicar também o jogador que possui a caveira.
- display_player_turn/1: recebe como primeiro parâmetro o jogador atual e apresenta no ecrã uma mensagem que indica quem é o jogador que deve fazer a próxima jogada. A imagem que se segue mostra o resultado destes dois últimos predicados.

tabuleiro_jogo

Ao correr inicialmente o jogo, é nos apresentado o menu principal. Neste, é pedido para escolher um dos três estilos de jogo: humano contra humano, humano contra computador ou computador contra computador.

menu_principal

Para isto recorremos ao predicado main_menu/0, que por sua vez chama o display_main_menu/0. Este limita-se a mostrar o menu. Recorremos também ao ask_menu_option/2 que pede uma opção ao utilizador, validando-a de seguida. Recorremos também ao predicado next_state/2 que com base no estado atual e na opção do utilizador decide qual o estado de jogo seguinte.

menu_principal_predicado

Existe também um outro menu no qual o utilizador pode selecionar qual a dificuldade do jogo (este menu só é apresentado no caso do utilizador ter selecionado uma das opções de jogar com computador). Aqui, o utilizador pode selecionar um de três níveis de dificuldade: fácil, médio ou difícil.

menu_de_dificuldade

Para isto, recorre-se ao predicado menu_select_difficulty/1 que por sua vez utiliza o predicado display_menu_select_difficulty/0. Este simplesmente mostra o menu e ask_menu_option/2 que vai pedir uma opção ao utilizador e ao mesmo tempo vai também validá-la. Este predicado retorna a dificuldade selecionada pelo utilizador.

menu_de_dificuldade_predicado

Por último temos também outro menu que só aparece caso o utilizador tenha selecionado a opção humano contra computador no qual é possível escolher qual das peças o computador será ou Os ou Xs.

menu_de_seleção_de_peça

Para isso recorremos ao predicado menu_select_piece/1 que é idêntico ao predicado anterior. Simplesmente recorre ao predicado display_menu_select_piece/0 que mostra o menu assim como ask_menu_option/2 pede uma opção ao utilizador e ao mesmo tempo valida-a. Este predicado retorna a peça selecionada pelo utilizador para o computador.

menu_de_seleção_de_peça_pred

Lista de jogadas válidas

É possível obter um conjunto de jogadas válidas através no predicado valid_moves/3. Este predicado recebe o estado de jogo Board e o Player, devolvendo uma lista ListOfMoves com todas as jogadas possíveis para esse jogador. A lista ListOfMoves é representada através de uma lista de listas. Cada uma das sub-listas contém por sua vez uma ou várias listas internas que representam uma jogada (várias no caso de serem comidas várias peças num turno). Uma jogada representa-se por [Row,Col,NextRow,NextCol]. ListOfMoves apresenta a estrutura final seguinte: [[[Row,Col,NextRow,NextCol],[*idem se for possível comer outra peça*]],[*outras jogadas*.]]

valid_moves

- O predicado valid_moves/3 começa por chamar get_row_num/5 de forma a obter uma lista contendo a posição de cada elemento do jogador Player no tabuleiro.
- De seguida, é utilizado o predicado get_valid_adj_pos/4 de forma a obter as deslocações (para casas adjacentes) possíveis para cada peça do jogador. Estas jogadas, no entanto ainda não incluem movimentos para comer outra peça. O predicado confirma que na linha seguinte, as colunas adjacentes se encontram livres para a peça em questão poder ser movimentada. Caso estejam, as mesmas são adicionadas à lista de posições adjacentes disponíveis.
- Depois, através de get_valid_eat_pos/3 é possível obter as jogadas que permitem comer outras peças. O predicado utiliza um outro: eat_all_dir/5 que obtém todas as casas a uma distância de 2 para onde será possível movimentar a peça que vai comer outra, no caso da futura casa estar livre. Por fim, get_valid_eat_pos/3 chama-se recursivamente de modo a percorrer todos os elementos.

Execução de jogadas

Para a a execução de jogadas utiliza-se o seguinte predicado move/3.

`move_pred`

Este predicado recebe como primeiro parâmetro o estado de jogo atual, como segundo a jogada a efetuar e retorna no terceiro o novo estado de jogo que irá existir após a jogada. Como é possível observar, este predicado recorre a outros predicados entre os quais estão:

- verifyEatMove/3: este predicado recebe como primeiro parâmetro a jogada a efetuar, como segundo parâmetro o tabuleiro atual e como terceiro retorna uma variável que nos diz se a jogada a efetuar é uma jogada de movimento normal ou se alguma peça é "comida". Sucintamente, o predicado permite diferenciar se o jogador efetua um movimento para uma casa adjacente ou se "come" uma outra peça.
- execute_moves/11: este predicado recebe no primeiro parâmetro o tabuleiro atual, no segundo o jogador atual, no terceiro, quarto e quintos o número de cada peças comidas de cada tipo, sexto a jogada atual, sétimo o tipo de jogada, oitavo o novo tabuleiro depois da execução da jogada, nono, décimo e décimo-primeiros os novos valores de peças comidas de cada tipo depois da execução da jogada. Em suma, este predicado executa a jogada, removendo a peça selecionada da posição atual e colocando-a na nova posição. Eliminando também eventuais peças comidas.
- change_Player_Has_Z/4: este predicado recebe como primeiro parâmetro o tipo de jogada, em segundo o jogador atual, em terceiro o jogador que detém posse sobre os zombies e como quarto e valor de retorno, o jogador que detém posse sobre os zombies depois da jogada ser executada. Este predicado muda (ou não) o jogador que detém controlo sobre os zombies tendo em conta a jogada atual.
- next_player/4: este predicado recebe como primeiro parâmetro o jogador atual, segundo parâmetro o jogador que detinha a posse dos zombies antes da jogada, em terceiro o jogador que detém posse sobre os zombies após a jogada e como quarto e valor de retorno, o jogador que deve executar a próxima jogada. O predicado indica o jogador seguinte em função da jogada atual.

Final de jogo

Para avaliarmos se o jogo chegou ao fim recorremos ao seguinte predicado game_over/2. Este predicado recebe como 1º parâmetro o estado atual do jogo e como 2º e valor a retornar o vencedor, caso o jogo ainda não tenha chegado a um fim o predicado falha.

Se o jogo chegou a um fim significa uma das seis seguintes possibilidades foi atingida:

- No caso de todas as peças X serem eliminadas:

`game_over_x_elim_pred`

- No caso de todas as peças O serem eliminadas:

`game_over_o_elim_pred`

- No caso de todas as peças Z serem eliminadas:

`game_over_z_elim_pred`

- Todas as peças X estão em contacto com a borda do tabuleiro correspondente:

`game_over_x_bord_pred`

- Todas as peças O estão em contacto com a borda do tabuleiro correspondente:

`game_over_o_bord_pred`

- Todas as peças Z estão em contacto com a borda do tabuleiro correspondente:

`game_over_z_bord_pred`

Todos estes predicados têm os dois últimos predicados comuns:

- calcElemPontuation/7: este predicado recebe como primeiro parâmetro o tabuleiro atual, segundo, terceiro e quartos o número de peças de cada tipo eliminadas e quinto, sexto e sétimos os valores a retornar com a pontução de cada tipo de peça tendo em conta os parâmetros anteriores. O predicado permite-nos obter a pontuação de cada tipo de peça.
- calcWinner/4: este predicado recebe como primeiro, segundo e terceiros parâmetros as pontuações de cada peça e como quarto e valor de retorno o vencedor. Em caso de empate, este parâmetro fica em branco. O predicado permite-nos obter a pontuação de cada tipo de peça.

Avaliação do tabuleiro

A avaliação do tabuleiro é feita através do predicado value/3. Este recebe o estado de jogo Board e o Player e devolve o valor da pontuação do jogador Player em Value. A pontuação de um jogador depende de dois parâmetros, as peças adversárias comidas (1 ponto cada) e o número de peças próprias que se encontram posicionadas na parede do tabuleiro com a cor correspondente (2 pontos cada). O predicado value utiliza duas outras funções para este cálculo:

`value`

- calc_value_on_board/4 que recebe o jogador Player e tabuleiro Board e a linha RowNum e devolve em Value o valor da pontuação do número de peças do jogador que se encontram em contacto com as casas da cor correspondente. Todas as linhas do tabuleiro são percorridas recursivamente e o valor vai sendo adicionado a Value.
- calc_value_elim/5 que recebe o jogador Player ('X','Z','O') assim como as peças comidas de cada tipo XElim, OElim, ZElim, e devolve em Value a soma do número de peças comidas de tipos diferentes das do Player.
- O resultado Value resulta da soma das pontuações resultantes de ambas as funções anteriores.

Jogada do computador

As jogadas do computador são realizadas através do predicado choose_move/4 que recebe o estado de jogo GameState, o jogador Player e o nível de dificuldade Level e devolve a jogada em Move. O predicado tem também uma segunda implementação choose_move/3 onde é passado apenas o nível e uma lista de jogadas ordenada por dificuldades e retorna a jogada adequada em Move.

`choose_move`

- No primeiro caso é criada uma lista com as jogadas válidas para o jogador em questão, através do predicado valid_moves. De seguida são calculados os valores de pontuação associados a cada jogada e adicionados a uma outra lista, que será posteriormente ordenada por dificuldade crescente. De seguida é chamado o choose_moves/3, onde é passada a dificuldade ('Easy','Medium','Hard') e a lista ordenada das jogadas com os valores associados
- Quando o nível de dificuldade passado é 'Easy', o predicado devolve a primeira jogada da lista. Quando esta é 'Hard' devolve a última jogada. Se for 'Medium', é devolvida a jogada presente no centro da lista.

Conclusões

Ao longo da realização do projeto, foi possível uma familiarização do grupo com a linguagem de Prolog. Linguagem esta que requereu um pensamento diferente das linguagens já estudadas previamente. Desta forma, foi possível cumprir os objetivos impostos ao grupo e desenvolver um jogo de tabuleiro funcional e completo.

Bibliografia

Manual do Sicstus: https://sicstus.sics.se/sicstus/docs/latest4/pdf/sicstus.pdf

Página do Jogo: https://nestorgames.com/rulebooks/GREENSKULL_EN.pdf