Implementação de um jogo de tabuleiro em Prolog

Minefield

Trabalho Prático 2

Programação Funcional e em Lógica

Turma 16 - Minefield 6

Contribuições:

João Proença (up202207835@fe.up.pt) - 50% maior foco nas regras e funcionamento do jogo

Ricardo Parreira (up202205091@fe.up.pt) – 50% maior foco no bot greedy

Instalação e execução

Para instalar o Minefield é necessário fazer download da pasta do jogo, PFL_TP2_T16_Minefield_6.zip, e descomprimi-la. Dentro da pasta src use o comando "consult('game.pl')" através da linha de comandos do Sicstus. Para iniciar o jogo utiliza-se o predicado "play.".

Descrição do jogo

Introdução

Minefield é um jogo para dois jogadores, jogado num tabuleiro quadrado de qualquer tamanho, inicialmente vazio. As bordas superior e inferior do tabuleiro são coloridas de preto, enquanto as bordas esquerda e direita são coloridas de branco. Um jogador joga com peças pretas e outro com peças brancas.

Objetivo do Jogo

O objetivo do jogo é criar um caminho ortogonalmente interconectado (horizontal e/ou vertical) de peças da sua cor, conectando as duas bordas do tabuleiro correspondentes a essa cor. O jogador que conseguir formar esse caminho primeiro vence a partida. Por exemplo, o jogador das peças pretas deve conectar aa borda superior à barda inferior, enquanto o jogador das peças brancas deve conectar a borda esquerda à borda direita.

Regras do Jogo

O jogo é disputado por dois jogadores, Preto e Branco, que jogam alternadamente, sempre começando com o jogador Preto. Em cada turno, os jogadores devem colocar uma peça correspondente à sua cor em um ponto desocupado do tabuleiro. Não é permitido passar a vez manualmente, mas, caso não existam movimentos válidos disponíveis, o turno será automaticamente passado para o próximo jogador.

Existem restrições específicas durante o jogo: não é permitido criar *hard corners* nem *switches*. Um *hard corner* ocorre quando, em uma área 2x2, há duas peças de uma cor dispostas em diagonais opostas, uma

peça de outra cor em um dos cantos restantes, e o último ponto está vazio. Já um *switch* ocorre em áreas 2x3 ou 2x4, onde os quatro cantos estão ocupados por duas peças de cada cor, dispostas em diagonais opostas, enquanto os pontos centrais permanecem desocupados.

A condição de vitória para o jogador Preto é conectar, de forma ortogonal, as bordas preta superior e inferior. Para o jogador Branco, a vitória ocorre ao conectar, também de forma ortogonal, as bordas branca esquerda e direita. O jogo é considerado empatado se nenhum dos dois jogadores tiver movimentos válidos disponíveis.

Fontes Consultadas

- https://boardgamegeek.com/boardgame/420797/minefield
- https://marksteeregames.com/Minefield_rules.pdf

Considerações para Extensões do Jogo

Ao projetar o jogo **Minefield**, uma das principais considerações foi a adaptação a diferentes tamanhos de tabuleiro, oferecendo maior flexibilidade para atender a diversos estilos de jogo e preferências dos jogadores. Para isso, implementamos suporte a três tamanhos de tabuleiro: **10x10 , 13x13** e **16x16**.

Além disso, foram incluídas **regras opcionais** para permitir uma experiência mais acessível e adaptável, especialmente para iniciantes. Uma das principais opções é a **eliminação das restrições de hard corners e switches**, facilitando o aprendizado ao permitir que os jogadores se concentrem exclusivamente no objetivo principal do jogo: conectar as bordas de sua cor. Essa simplificação torna as primeiras partidas mais dinâmicas e acessíveis. A flexibilidade é garantida pela opção de escolher entre as regras padrão ou simplificadas no momento da configuração do jogos.

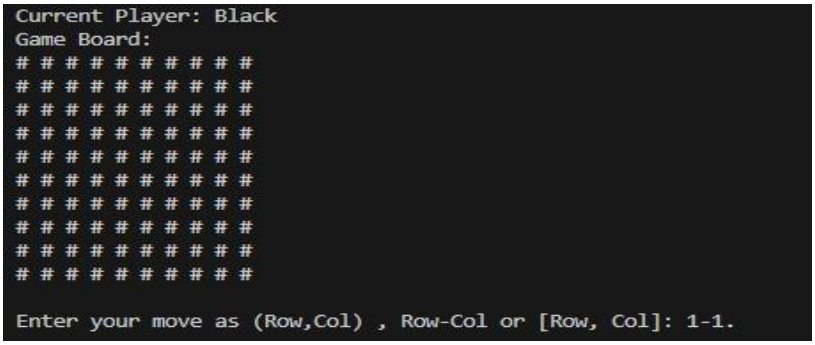
Lógica do Jogo

Representação da Configuração do Jogo

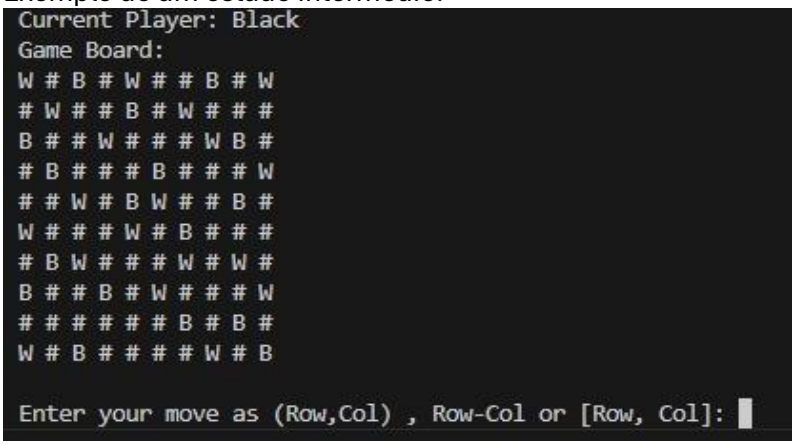
Na nossa implementação, a configuração do jogo é representada como uma lista contendo os seguintes elementos: **Tipo de Jogo** (definindo se é entre jogadores humanos ou com computador), **Tamanho do Tabuleiro** (escolha entre 10x10, 13x13 ou 16x16), **Dificuldade** (apenas para jogos com computador, podendo ser 1 para movimentos aleatórios ou 2 para estratégia baseada em avaliação de jogadas), e **Modo de Jogo** (onde 1 representa regras simplificadas e 2, regras padrão).

Representação Interna do Estado do Jogo

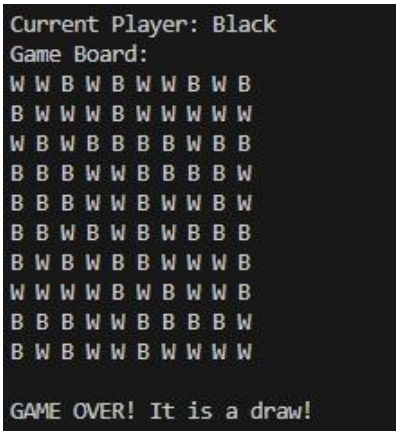
A representação interna do estado do jogo é essencial para gerenciar o progresso e o resultado da partida. O estado do jogo inclui o tabuleiro, o jogador atual, a lista dos jogadores e o modo de jogo. O tabuleiro é uma lista de listas, onde cada célula pode ser `empty` (vazia), `'b'` (peça preta) ou `'w'` (peça branca), e seu tamanho varia conforme a configuração (10x10, 13x13 ou 16x16). O jogador atual é representado por um átomo que pode ser `'Black'` ou `'White'`. A lista de jogadores armazena o nome e o nível de dificuldade, por exemplo um player dificuldade 0, um bot random dificuldade 1 e um bot greedy dificuldade 2. O modo de jogo é um valor indicando se as regras são simplificadas (1) ou padrão (2). Essas informações são usadas para atualizar e controlar o jogo conforme ele avança.
Exemplo do estado inicial:



Exemplo de um estado intermédio:



Exemplo do estado final:



Representação de Movimentos

Informação Necessária para Representar um Movimento

Para representar um movimento no jogo, utilizamos as **coordenadas do tabuleiro**, expressas como [Row, Col], onde Row e Col indicam a linha e coluna em que o jogador deseja posicionar sua peça. O **estado do jogo atual** inclui informações como o tabuleiro, o jogador da vez, a lista de jogadores (com suas características) e o modo de jogo selecionado.

Representação Interna

Um movimento é representado internamente como uma lista contendo duas partes: as **coordenadas [Row, Col]**, que indicam a posição no tabuleiro onde o jogador deseja realizar sua jogada, e o **estado do jogo**, que é atualizado ao aplicar o movimento válido ao estado anterior.

Uso no Predicado `move/3`

O predicado **move/3** valida e aplica o movimento ao estado atual do jogo em quatro etapas principais: primeiro, a função **valid_moves/2** obtém os movimentos válidos e verifica se o movimento solicitado pertence à lista. Em seguida, o predicado **color/2** associa o jogador atual ao símbolo correspondente no tabuleiro (por exemplo, "b" para "Black" e "w" para "White"). O tabuleiro é então atualizado com o predicado **set_cell/5**, que preenche a célula selecionada com o valor do jogador atual. Por fim, o predicado **switch_player/2** alterna o turno para o próximo jogador.

User Interaction

Game Menu System

O sistema de menu do jogo oferece uma interface intuitiva para os usuários configurarem e interagirem com o jogo. As opções incluem: escolha do tipo de jogo (H/H, H/PC, PC/H, PC/PC), definição do tamanho do tabuleiro (10x10, 13x13 ou 16x16), seleção do modo de jogo (Beginner ou

Normal) e configuração do nível de dificuldade do computador (Random ou Greedy). O menu é baseado em entradas sequenciais, com validação para garantir que apenas escolhas válidas sejam aceitas. Caso uma entrada inválida seja detectada, o sistema exibe uma mensagem de erro e solicita ao usuário que tente novamente.

Validação e Processamento de Entradas

A validação das entradas no jogo ocorre em dois níveis. Primeiro, para a leitura de números, o predicado **read_number/2** lê os caracteres digitados no teclado e os converte em números inteiros, acumulando os valores e verificando se a entrada é finalizada corretamente (por exemplo, ao pressionar Enter).

Em seguida, o predicado **parse_input/2** lida com a entrada de coordenadas, aceitando formatos como (Row, Col), Row-Col ou [Row, Col]. Ele valida se os valores são inteiros, retornando uma mensagem de erro caso contrário e solicitando uma nova entrada. Além disso, o predicado **translate_input/3** ajusta as coordenadas fornecidas pelo jogador para a convenção interna do tabuleiro, onde a origem (1,1) está no canto inferior esquerdo. Assim, coordenadas como (1,5) em um tabuleiro 10x10 são traduzidas para [10,5] na representação interna.

Conclusão

O projeto Minefield foi desenvolvido com sucesso em Prolog, permitindo partidas entre dois jogadores, bem como modos de jogo com I.A. em dois níveis de dificuldade. A implementação seguiu rigorosamente as especificações, incluindo regras de validação de movimentos (hard corner e switch), detecção de vitória por conexão, e a manipulação de tabuleiros de tamanhos variáveis. O uso de algoritmos de avaliação, como a função de potencial de peças e a detecção de conexões, demonstrou ser eficaz na simulação de estratégias básicas e avançadas para a I.A.

Entre as limitações, destaca-se o grande desgosto de não ter conseguido usar a ideia original de connection_score que calcularia a pontuação de cada conexão usando um dfs e um sistema inteligente de cáculo de pontos. Também a ausência de optimizações para grandes tabuleiros, que poderiam causar lentidão em cálculos complexos, como a análise de todas as conexões. Além disso, a I.A.poderia usufruir de algoritmos mais sofisticados, como o minimax com pruning, o que seria uma excelente adição para futuros desenvolvimentos e não seria muito difícil de implementar visto que o predicado value/3 já está definida e serviviria de apoio para o minimax. Melhorias futuras podem incluir a integração de visualizações gráficas, melhoria de optimização do programa e um melhor equilibro no sistema de pontuação do value/3.

Bibliografia

O recurso que nos foi mais útil e utilizado durante o trabalho foi sem dúvida a documentação oficial do SICStus Prolog, lá encontrámos todas as funções tal como o seu funcionamento e argumentos, que nos permitiram completar este trabalho. Recorremos também a alguns fóruns de discussão informática onde encontrámos soluções para problemas que estávamos a ter tais como Reddit e StackOverflow. Recorremos também a modelos como o ChatGPT principalmente para analisar código feito por nós. Tanto para analisar outras opções mais optimais, como também para analisar a lógica e a concordância entre funções e se haveria algum caso específico que podesse falhar e nos tivesse escapado nos testes que conduzimos.