Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores
Licenciatura em Engenharia Informática e de Computadores
Licenciatura em Matemática Aplicada à Tecnologia e à Empresa
Licenciatura em Engenharia Informática, Redes e Telecomunicações

The Checkers game

First Practical Project

Artificial Intelligence Course

Summer Semester 2024/2025

22 April 2025

Docente: Nuno Leite
Daiana Lupaiescu, A48668
João Lopes, A50457

Índice:

Introdução

Este relatório descreve o desenvolvimento de um jogo de Damas (Checkers), implementado em Prolog, como parte do primeiro projeto prático da disciplina de Inteligência Artificial (AI) .
Este projeto teve como principal objetivo a criação de um jogo funcional que possibilite duas modalidades: Jogador contra Jogador (Player vs Player) e Jogador contra Computador (Player vs AI).

O projeto contempla diferentes tamanhos de tabuleiro (5x5 e 8x8), inteligência artificial com algoritmo Minimax e variação Alpha-Beta Pruning, com a implementação das regras tradicionais da modalidade de damas.

Objetivos

Os principais objetivos do projeto foram:

- Compreender e dominar a sintaxe do Prolog.
- Implementar conceitos básicos do Prolog, como átomos, números, termos compostos, listas, operadores e aritmética.
- Controlar o backtracking em Prolog.
- Utilizar predicados integrados do Prolog.
- Implementar um jogo de dois jogadores com informação perfeita.
- Compreender o princípio do minimax e a implementação eficiente do algoritmo alfa-beta.

Descrição do Jogo de Damas

O jogo de Damas é um jogo de tabuleiro entre dois jogadores, que ocorre em um tabuleiro quadrado de 8x8.
Cada jogador começa com 12 peças, posicionadas nas casas escuras das três primeiras linhas do tabuleiro. O objetivo do jogo é capturar todas as peças do oponente ou bloqueá-lo de maneira que não tenha mais movimentos possíveis.

Regras principais:

- Movimento das peças: As peças movem-se diagonalmente, uma casa por vez, em direção ao lado oposto do tabuleiro.

- Captura: Quando uma peça adversária ocupa uma casa adjacente a uma peça do jogador e a casa além dessa peça está vazia, o jogador pode capturar a peça adversária saltando sobre ela. O movimento de captura deve ser realizado obrigatoriamente quando possível.

- Captura Obrigatória: Sempre que uma peça tem a possibilidade de capturar uma peça adversária, a captura deve ser realizada.

- Promoção de peças: Quando uma peça atinge a última linha do tabuleiro, ela é promovida a uma "Dama", podendo mover-se para frente e para trás diagonalmente qualquer número de casas possíveis.

- Fim de jogo: O jogo termina quando um dos jogadores não tem mais peças ou não pode realizar movimentos válidos.

Implementação

O código é dividido logicamente em quatro módulos principais:

Game.pl - Inicio do jogo
Responsável:

- Iniciar o jogo.
- Escolher em que modo queremos jogar, Player VS Player ou Player vs AI, também como escolher o tamanho do tabuleiro(5x5 e 8x8).

GenerateBoard.pl – Geração do Tabuleiro
Responsável por:

- Criar a estrutura inicial do tabuleiro.
- Preencher as peças de cada jogador.
- Imprimir o estado atual do jogo de forma legível ao usuário.

Validmoves.pl – Lógica do Jogo
Responsável por:

- Validar todos os movimentos possíveis para peões e damas.
- Implementar as regras de captura obrigatória.
- Verificar e promover peões a damas.
- Detectar o fim do jogo.

Minimax.pl – Inteligência Artificial
Responsável por:

- Implementar o algoritmo Minimax para a tomada de decisões do computador.
- Avaliar o tabuleiro por meio de uma função heurística simples.
- Utilizar o Alpha-Beta nos jogos de 8x8 para maior eficiência.

Modo Player vs Player

No modo Player vs Player, dois jogadores inserem alternadamente os seus movimentos no terminal. As jogadas são validadas com base nas regras do jogo. Caso uma jogada seja inválida, o jogador é notificado e deve repetir.
Foi criado um ciclo *'game_loop'* que gere este modo, imprimindo o tabuleiro, solicitando a jogada, e atualizando o estado do jogo.

Modo Player vs AI

No modo Player vs Ai, o jogador humano joga com as peças brancas e a AI com as pretas. O jogador humano introduz jogadas como no modo PvP, enquanto a AI calcula automaticamente o melhor movimento usando o Minimax e o minimax com Alpha-Beta.
O ' *game_loop_vs_ai'* gere o ciclo de jogo, incluindo a verificação do fim do jogo e o retorno ao menu inicial após a conclusão.

Representação do Tabuleiro

A representação do tabuleiro foi feita utilizando listas em Prolog. Cada linha do tabuleiro é uma lista, e cada célula contém um valor que pode ser um símbolo representando uma peça de um dos jogadores, uma dama, ou uma célula vazia.
As peças são representadas por símbolos Unicode:

- '◯' para as peças do jogador 1.
- '⬤' para as peças do jogador 2.
- '♔' para as damas do jogador 1.
- '♚' para as damas do jogador 2.
- '.' para casa vazia

A estrutura dos dados facilita a manipulação das peças durante o jogo, permitindo verificar facilmente as posições, capturas e movimentos.

Regras Implementadas

Os movimentos e capturas foram implementadas de acordo com as regras clássicas do jogo.

Movimento Simples

Os movimentos simples são executados com a função *'is_simple_move'* e só são permitidos quando:

- A célula de destino estiver vazia.
- A direção for válida para o tipo de peça.
- O movimento for diagonal e apenas de uma casa.
- Para damas, a diagonal precisa estar livre até ao destino (implementado com '*check_diagonal_clear*')

Captura

A captura é obrigatória e válida se:

- Há uma peça adversária numa diagonal adjacente.
- A célula logo após essa peça está vazia.
- O movimento cobre duas casas na diagonal.

A função '*is_capture*' valida este cenário, incluindo captura por damas.

Captura Múltipla

Após uma captura, a função '*can_continue_capture*' verifica se a mesma peça pode continuar a capturar. Caso sim, o jogador é forçado a continuar a jogada com a mesma peça.

Promoção a Dama

Quando uma peça branca atinge a última linha (Row == N) ou uma peça preta atinge a primeira linha (Row == 1), é promovida a dama. Isto acontece pois implementamos a função promote_piece durante a execução do make_move.

Minimax

O algoritmo Minimax é utilizado para simular todas as jogadas possíveis até uma profundidade limite. O objetivo é encontrar o movimento que maximiza o ganho mínimo garantido, assumindo que o adversário também joga da melhor forma possível.

A função principal é:

minimax(Board, Player, Depth, BestMove, Value)

Esta utiliza :

- '*generate_moves'* para obter jogadas válidas;
- '*best'* para determinar o melhor movimento;
- '*maxtowin'* e '*mintowin'* que alternam entre MAX e MIN;
- '*evaluate'* para calcular o valor do estado.

'Maxtowin' e 'Mintowin'

- maxtowin representa a vez do jogador atual, que quer maximizar o valor da jogada.
- mintowin representa a vez do adversário, tentando minimizar o valor.

'best' e 'worst'

Estas funções percorrem a lista de movimentos possíveis:

- 'best' escolhe o movimento com o maior valor de utilidade.
- 'worst' escolhe o menor valor (usado para simular o adversário).

A lógica compara o valor atual com o anterior e guarda o melhor ou pior de acordo com o contexto (max ou min).

'generate_moves' e 'evaluate'

- '*generate_moves*' gera todas as jogadas válidas para o jogador atual, considerando as regras do jogo.
- '*evaluate'* avalia o estado do tabuleiro, atribuindo uma pontuação baseada nas peças e damas de cada jogador.

Minimax com Alpha - Beta

A variação do algoritmo Minimax com Alpha-Beta representa uma optimização significativa no processo de tomada de decisão da Inteligência Artificial. Enquanto o algoritmo Minimax tradicional avalia todos os ramos possíveis da árvore de jogo até uma determinada profundidade, a versão com Alpha-Beta consegue eliminar ramos que não influenciam o resultado final, reduzindo substancialmente o número de estados analisados.

Implementação no Código

A função principal é:

alpha_beta(Board, Player, Depth, Alpha, Beta, BestMove, Value)

Esta inicia o processo de decisão para o Player no estado de jogo Board, com uma profundidade Depth e os limites Alpha e Beta. A função '*generate_moves*' gera todos os movimentos possíveis e, posteriormente, '*alpha_beta_best*' analisa-os recursivamente.

A lógica é dividida em três fases principais:

- '*alpha_beta_best*' – Analisa os movimentos disponíveis para o jogador atual (MAX), escolhendo o que maximiza o valor.
    - Se o valor de um movimento for maior que o Alpha atual, este é atualizado.
    - Se esse novo Alpha for maior ou igual a Beta, os restantes movimentos não são analisados.

- '*alpha_beta_min*' – Depois do jogador atual simular uma jogada, o adversário responde. Nesta fase procura-se o menor valor possível (MIN), chamando *'alpha_beta_worst'.*

- '*alpha_beta_max*' – Quando o controlo volta ao jogador original, repete-se o processo de maximização, utilizando novamente *'alpha_beta_best'.*

Conclusão

O desenvolvimento deste projeto representou uma valiosa oportunidade para aplicar os conhecimentos adquiridos durante as aulas da unidade curricular de Inteligência Artificial, integrando teoria e prática de forma eficaz em um desafio concreto.

A construção de um jogo de damas completo em Prolog exigiu a compreensão profunda de vários conceitos essenciais, como a representação de estados, manipulação de listas, raciocínio baseado em regras, e a implementação de algoritmos de tomada de decisão. Durante o processo, explorámos de forma prática técnicas avançadas como o algoritmo Minimax e Minimax com Alfa-Beta, aplicadas à criação de uma inteligência artificial capaz de competir com um jogador humano. A complexidade das regras do jogo, como capturas obrigatórias, múltiplas capturas sequenciais, e promoção de peças, reforçou a importância de uma modelação rigorosa e de uma lógica de jogo bem estruturada, elementos que foram desenvolvidos com base nos conteúdos lecionados.

Este projeto foi uma experiência muito importante, permitindo-nos aprender imenso não só com a execução prática, mas também com as aulas lecionadas. Através deste desafio, conseguimos aplicar de forma concreta os conhecimentos adquiridos, o que tornou o processo de desenvolvimento ainda mais interessante.