Tafl

Relatório Intercalar

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

Cláudia Margarida da Rocha Marinho – up201404493 – up2014044093@fe.up.pt

Diogo Amorim Cepa - 201403367 - up201403367@fe.up.pt

Tiago Rafael Ferreira da Silva – up201402841 – up201402841@fe.up.pt

29 de março de 2017

Objetivo

O estado final deste projeto deverá ter implementado o jogo *Tafl* para dois jogadores que permita a um humano jogar contra um computador. Deverão ter sido desenvolvidos vários modos de jogo, deverá ser permitido jogar em dificuldades diferentes e deverá ter sido implementada uma interface gráfica simples que permita visualizar os estados do jogo facilmente e intuitivamente.

Tafl é um jogo de tabuleiro e as jogadas do computador deverão ser implementadas de forma inteligente, ou seja, o computador deverá de ser capaz de prever até um número definido de jogadas no futuro e escolher as melhores jogadas tendo em conta essas previsões. Para este fim, vamos usar o algoritmo Minimax com cortes Alfa-Beta na implementação do jogo referido.

Também é importante referir que usamos Prolog para implementar o jogo.

Especificação

O objetivo deste projeto é a implementação do jogo de tabuleiro *Tafl* para dois jogadores. Para este fim, vão ser desenvolvidos três modos de jogo diferentes: humano contra humano, humano contra computador e computador contra computador. Deve ser possível definir várias dificuldade para os jogos que envolvem o computador (o que por sua vez define a "inteligência" do computador). Além disso, uma interface gráfica para este jogo vai ser implementada em Prolog, de forma a permitir uma visualização fácil dos estados do jogo.

A seguir, vamos apresentar as regras do jogo e várias outras informações relativamente à implementação deste problema como um problema de pesquisa adversarial.

Regras do jogo

Tafl é o nome genérico de uma família de jogos de tabuleiro antigos de origem germânica e céltica, jogados num tabuleiro quadrado de 9 por 9, usado nesta implementação, mas também existem as possibilidades de 11 por 11 ou 13 por 13.

A versão do jogo que vamos implementar chama-se Tablut e o estado inicial do tabuleiro nesta versão é a seguinte:

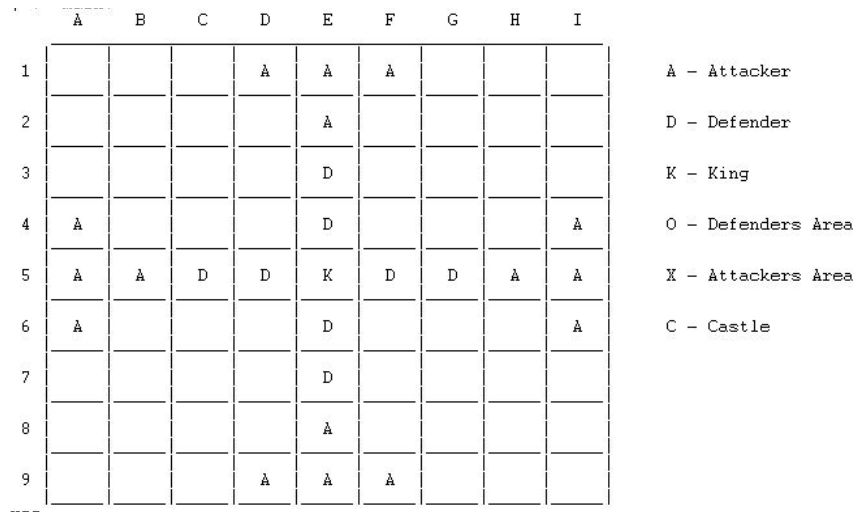

Figura 1: Configuração inicial do tabuleiro.

Como se pode observar na figura 1, o rei (representado por K) começa na casa central do tabuleiro, o castelo, e essa casa nunca pode ser ocupada por qualquer uma das outras peças. Os oitos defensores (representadas pela letra D na figura mostrada anteriormente) são posicionados em forma de uma cruz em volta do castelo inicialmente, enquanto que os dezasseis atacantes (representados pela letra A na figura 1) são posicionados em grupos de quatro no centro de cada lado do tabuleiro. Todas as casas do tabuleiro exceto o castelo, podem ser ocupadas pelos atacantes ou pelos defensores.

Todos os outros estados do jogo vão ser apresentados através de uma lista de listas e a sua impressão na consola do Sicstus é como a apresentada na figura 1.

Qualquer uma das peças mencionadas pode mover-se horizontalmente (esquerda e direita) e verticalmente (para cima e para baixo) qualquer número de cases, desde que não ultrapassem outras peças do jogo no movimento. Além disso, em cada turno um jogador só pode mover uma peça. O movimento das peças corresponde à função de transição na implementação do nosso problema de pesquisa adversarial, pois o movimento das peças vai ser o que via permitir a transição entre estados.

Qualquer peça (exceto o rei) pode ser capturada e, portanto, removida do tabuleiro se estiver cercada por duas peças adversárias em lados opostos.

O objetivo do jogo no lado dos defensores é assegurar que o seu rei consiga atingir um lado do tabuleiro que não pertença à área dos defensores, enquanto que no lado dos atacantes, o objetivo é encurralar o rei pelos quatros lados com peças adversárias. É importante mencionar que quando o rei sai do "castelo", este não pode voltar lá e, por isso, se o rei estiver rodeado por três peças adversários e no último lado estiver o castelo, o lado das peças defensoras perde. Estes estados em que o rei se encontra numa das casas na fronteira do tabuleiro ou os estados em que o rei se encontra encurralado correspondem aos estados finais.

Foi definida uma heurística para a função de avaliação do nosso trabalho que vai permitir ajudar na determinação da melhor jogada em cada turno. Como é possível comer peças, achamos que seria vantajoso ter mais peças que o adversário. Também achamos relevante a distância a que o rei está de sair do mapa um ponto relevante, tendo assim o atacante menos casas para impedir a sua saída. Com isto, chegamos a uma heurística simples, em que calculamos o melhor para o atacante: Valor(tabuleiro) = (PeçasAtacante - PeçasDefesa) + 0.1 * DistanciaRei. O 0.1 simboliza que será mais importante ter mais peças do que o rei estar perto de uma das saídas possíveis. Nesta fase, achamos que a heurística aplica-se bem ao problema ainda que tenha margem para melhorias.

O algoritmo que usamos para determinar as jogadas em cada turno é o algoritmo Minimax com cortes Alfa-Beta. Para as peças defensoras, este algoritmo vai tentar determinar a melhor jogada (ou melhor, a jogada menos "arriscada"), de tal forma que o rei consiga chegar às casas que se encontram na fronteira do tabuleiro. Para as peças atacantes, este algoritmo vai tentar determinar o melhor percurso de forma que consiga encurralar o rei nos quatros lados.

No entanto, note-se que o nível de profundidade é limitado, pois a memória do computador é limitada e seria impossível verificar todas as jogadas possíveis. Por isso pode não ser possível "chegar" aos estados finais com o nível de profundidade estabelecido não for suficiente. Se este for o caso, então o algoritmo vai tentar determinar as jogadas que permitam

eliminar peças adversárias do tabuleiro, sem por as próprias peças em risco (note-se que o algoritmo vai tentar fazer isto tanto para as peças atacantes como para as defensores).

Trabalho Efetuado

Até ao momento, conseguimos representar os vários estados do tabuleiro usando o Prolog e implementar um predicado que permita mover as peças do tabuleiro. A representação do tabuleiro é igual à representada na figura 1, em que o rei, as peças atacantes e as peças defensoras encontram-se posicionadas na sua configuração inicial. Quando se move uma das peças, também é representada qual a área dos defensores, atacantes e do castelo, como se observa na figura abaixo:

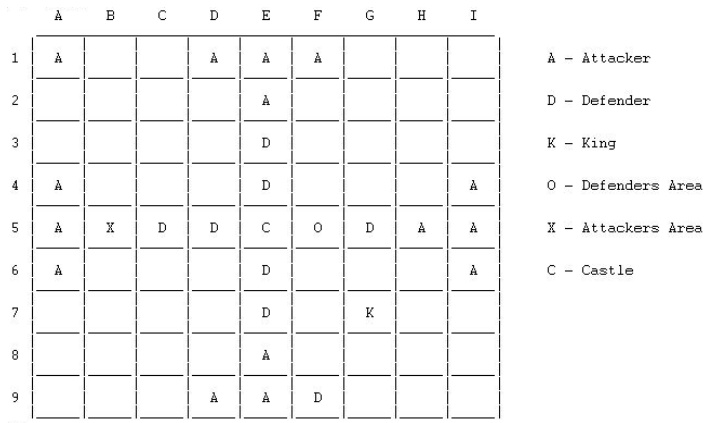

Figura 2: Representação de um estado diferente do inicial

A configuração apresentada não corresponde à inicial: o rei encontra-se fora do castelo (em G7), uma peça dos atacantes encontra-se numa zona neutra (em A1) e uma peça dos defensores encontra-se na zona dos atacantes (em F9). Como se pode verificar, nas casas em que as peças se encontravam inicialmente, agora estão letras a representar as áreas das peças (X para a área dos atacantes, O para a área dos defensores e C para a casa correspondente ao castelo).

Note-se que para mover uma peça foi implementada o predicado move(+Board, +OCol, +ORow, +NCol, +NRow, -NBoard), em que Board é o tabuleiro inicial, OCol e ORow representam a coluna e linha em que a peça que se pretende mover se encontrava inicialmente, NCol e NRow representam a nova coluna e linha da peça e NBoard é o novo tabuleiro após o movimento da peça. Outra função relevante implementada foi printBoard(+Board) que permite imprimir o tabuleiro Board.

Resultados Esperados e forma de avaliação

Quando o projeto estiver concluído, espera-se que o jogo esteja devidamente implementado, de tal forma que os modos humano contra humano, humano contra computador e computador contra computador estejam devidamente implementados (especialmente estes últimos que envolvem o computador), com pelo menos dois níveis de dificuldades distintos. Também se espera que a implementação seja eficiente o suficiente, de forma que as jogadas do computador (que envolvem o algoritmo Minimax) sejam relativamente rápidas.

Para testar o nosso projeto, deverão ser efetuados jogos em modos diferentes e com dificuldades diferentes, de forma a que seja possível testar se o algoritmo Minimax se encontra bem implementado.

Conclusões

Este projeto parece ser bastante interessante de implementar e estamos ansiosos para trabalhar nele. Também esperamos que seja possível alcançar todos os objetivos propostos para este projeto.