Universidade do Porto
Faculdade de Engenharia
FEUP

Jogo de Tabuleiro - Pentago

Relatório Intercalar

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Grupo constituído por:

Ana Filipa Barroso Pinto – 201307852 – up201307852@fe.up.pt

Pedro Miguel Vieira da Silva – 201306032 - up201306032@fe.up.pt

Pedro Vieira Lamares Martins – 201005350 – up201005350@fe.up.pt

14 de Abril de 2016

Universidade do Porto
FEUP Faculdade de Engenharia

Objetivo

A unidade curricular de Inteligência Artificial tem como principal objetivo dotar os seus estudantes de conhecimento relativo a sistemas inteligentes inteiramente criados, representados e manipulados por um computador. De todas as opções disponíveis para o projeto proposto, o grupo decidiu implementar um jogador inteligente para o jogo de tabuleiro *Pentago*, utilizando, para esse fim, algoritmos de pesquisa adversarial, tais como o *Minimax* com cortes Alfa-Beta e seus variantes. A aplicação final esperada irá colocar à disposição do utilizador uma interface gráfica com três modos de jogo, diversos níveis de dificuldade para o jogador automático e várias formas de testar a eficiência do algoritmo.

Descrição

Especificação

Pretende-se, com este trabalho, implementar uma versão do jogo de tabuleiro *Pentago* com vários modos de jogo e vários níveis de dificuldade, sendo necessária a criação de um jogador automático controlado pelo computador (utilizando, para isso, os algoritmos de pesquisa adversarial lecionadas na unidade curricular).

O *Pentago* é um jogo de estratégia do tipo soma-zero para dois jogadores concebido em 2005 por um *designer* sueco chamado *Tomas Flodén*. O jogo foi publicado no famoso *website* de jogos de tabuleiro *BoardGameGeek* e desde então, a partir de um *crowdfunding*, já foram vendidas mais de um milhão de cópias.

O jogo é realizado num conjunto de quatro tabuleiros quadrados de 3x3 cada, dispostos de forma a construir um tabuleiro maior de 6x6, totalizando 36 células onde cada um dos jogadores pode inserir peças (ou berlindes). Inicialmente, os tabuleiros encontram-se vazios.

Uma jogada consiste em dois movimentos sequenciais:

- O jogador em questão escolhe uma posição não ocupada num dos tabuleiros e coloca lá uma das suas peças.
- O jogador seleciona um dos tabuleiros para rodar 90 graus no sentido à sua escolha (no sentido dos ponteiros do relógio ou no sentido contrário).

O tabuleiro selecionado para rodar não tem de ser necessariamente o tabuleiro onde foi colocada a peça. O jogo desenrola-se ao longo de turnos consecutivos, trocando o jogador em cada turno. O jogo termina assim que, no final de cada turno, um dos jogadores conseguir pelo menos cinco peças em linha (no conjunto dos quatro tabuleiros).

Esta linha pode ser horizontal, vertical ou diagonal, e pode percorrer dois ou mesmo três tabuleiros. Se a última peça colocada pelo jogador resultou numa linha de cinco peças da sua cor, o jogo é finalizado sem a necessidade de se rodar um tabuleiro. O jogo termina num empate se não houver uma linha de pelo menos cinco peças no fim do jogo (quando todas as células do tabuleiro estiverem preenchidas com peças).

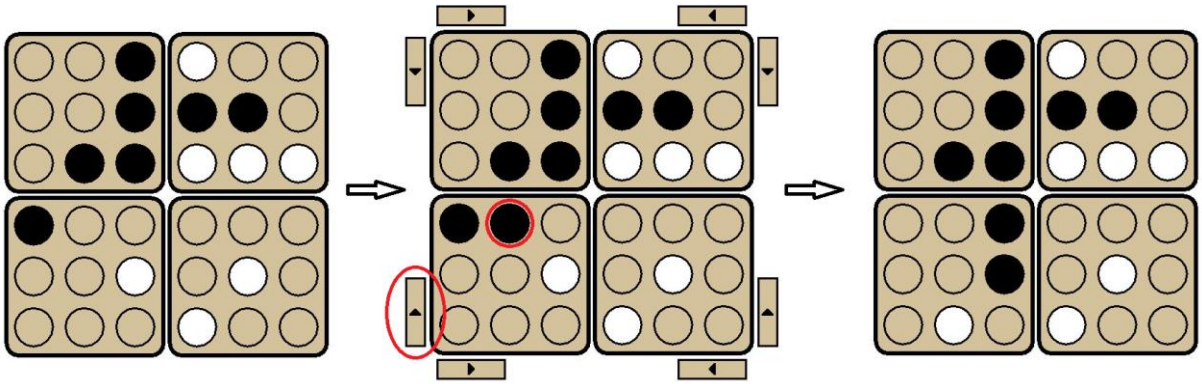

Figura 1 – Transição entre dois estados de jogo. É a vez do jogador representado pelas peças pretas. O jogador coloca a peça na posição destacada a vermelho e roda o tabuleiro no sentido dos ponteiros do relógio. Como existe uma sequência de pelo menos cinco peças da mesma cor, o jogador sagra-se vitorioso e é atingido um estado final.

O principal objetivo da realização deste projeto é a implementação de um jogador automático controlado pelo computador que possa jogar contra um utilizador humano. Este jogador inteligente deve utilizar algoritmos de pesquisa adversarial com heurísticas apropriadas para adquirir conhecimento sobre o estado de jogo, decidindo, com base nisso, qual a sua melhor jogada possível para esse estado. Propõe-se também, para além disso, a implementação de uma interface gráfica de fácil interação com o utilizador, podendo este escolher entre três modos de jogo (humano contra humano, humano contra computador e computador contra computador) e vários níveis de dificuldade.

Para concretizar estes objetivos, foi feita a seguinte abordagem:

- Construção da mecânica de jogo e de todas as suas regras utilizando a linguagem de programação *Java*.
- O desenvolvimento de uma interface gráfica para interação com o utilizador utilizando o conjunto de ferramentas *Swing*, uma biblioteca de *Java*.
- A implementação de algoritmos de pesquisa adversarial e a sua aplicação ao jogo (através da criação da noção de "estado de jogo" e de heurísticas válidas).

Relativamente à mecânica de jogo, a estrutura dos vários tabuleiros foi idealizada como um *array* de *arrays* de inteiros (ou seja, uma matriz de inteiros). Cada espaço na matriz pode ser ocupado com três números diferentes (sem peça, peça preta ou peça branca).

A junção desses tabuleiros e as operações relacionadas (tais como, por exemplo, colocar uma peça, rodar um tabuleiro no sentido dos ponteiros do relógio ou verificar se foi encontrado um estado final para terminar o jogo) são feitas como funções simples nas classes correspondentes.

A interface gráfica deve servir como a ponte de ligação entre o utilizador e o jogo, e por esse motivo deve ser simples e intuitiva. A partir do conjunto de classes da biblioteca *Swing* do *Java*, bem como da utilização de interfaces que implementam o uso do rato e o funcionamento de *threads* (as interfaces *MouseListener*, *MouseMotionListener* e *Runnable*), foi possível criar uma interface que permite ao utilizador selecionar e decidir entre os vários parâmetros disponíveis na aplicação, tais como:

- Escolher o tipo de jogo (um humano contra outro humano, um humano contra o computador ou o computador contra o computador).
- Escolher o nível de dificuldade para o computador (de três possíveis).
- Escolher o algoritmo de pesquisa a utilizar (*minimax* sem cortes alfa-beta, *minimax* com cortes alfa-beta e o algoritmo *negamax*).

Durante o jogo, o utilizador interage com a aplicação unicamente pelo uso do rato (clicando na célula e no botão para onde deseja colocar a peça e rodar o tabuleiro, respetivamente). Uma zona branca, por baixo do tabuleiro de jogo, clarifica ao utilizador humano aquilo que deve fazer ou explica a forma como o jogador automático procura/encontra a sua jogada. O jogador pode voltar ao menu inicial a qualquer momento.

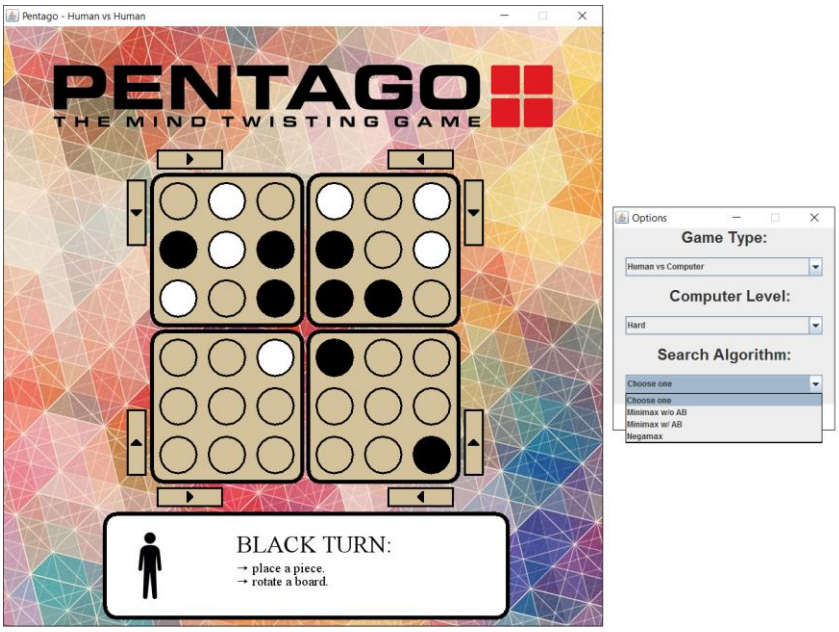

Figura 2 – Dois exemplos da interface gráfica de interação com o utilizador. A primeira imagem diz respeito à área de jogo. O utilizador clica nas células sem peça e nos botões de rotação, sendo a aplicação responsável por trocar de turnos. A segunda imagem demonstra as opções existentes no programa antes do início de um jogo.

Em relação ao módulo de representação de conhecimento, distinguem-se dois objetivos principais:

- Implementação dos vários algoritmos de pesquisa.
- Criação de heurísticas precisas e admissíveis.

O grupo tem em mente implementar dois algoritmos de pesquisa distintos (ainda que, a nível de código, bastante semelhantes) para o utilizador escolher, ambos lecionados na cadeira: o algoritimo *minimax*, que poderá ser escolhido com ou sem cortes alfa-beta (de modo a que o utilizador consiga perceber a diferença de eficiência entre os dois) e o algoritmo *negamax*. O pseudocódigo do *minimax* está a ser pensado da seguinte forma:

```
minimax(profundidade, jogador) // jogador definido pela sua cor
se (fim do jogo || profundidade == 0)
  devolve avaliacaoDoEstado
descendentes = movimentos possíveis a partir deste estado
se (jogador == PECAS_PRETAS) // caso para o jogador a maximizar
  // encontra o máximo
  melhorValor = -INFINITO
  para cada descendente: {
    valor = minimax(profundidade - 1, PECAS_BRANCAS)
    se  (valor > melhorValor)
        melhorValor = valor
  }
  devolve melhorValor
caso contrário (jogador == PECAS_BRANCAS) // caso minimizar
  // encontra o mínimo
  melhorValor = +INFINITO
  para cada descendente: {
    valor = minimax(profundidade - 1, PECAS_PRETAS)
    se (valor < melhorValor)
        melhorValor = valor
  }
  devolve melhorValor

// chamada inicial para profundidade 3
minimax(3, PECAS_PRETAS)
```

A versão com cortes alfa-beta precisa de mais dois parâmetros no início do algoritmo: um valor para o *alpha* (que guarda o maior valor encontrado nos níveis de maximização) e outro valor para o *beta* (que guarda o menor valor encontrado nos níveis de minimização). O algoritmo irá parar o ciclo de análise de cada descendente sempre que *alpha* >= *beta*.

Por sua vez, o algoritmo *negamax* irá modelar o seu comportamento na premissa de que *max(a,b) = -min(-a, -b)*. Assim, para cada chamada recursiva ao algoritmo na análise a cada descendente, será feita uma chamada a *–negamax* (ao invés de apenas *negamax*) e o valor passado para essa chamada será sempre o negativo do valor atual. Ao utilizador será permitido escolher a profundidade máxima a usar no algoritmo.

Há dois aspetos a destacar nos algoritmos que serão alvo fundamental na implementação dos mesmos: a geração dos estados possíveis seguintes a partir do estado atual, bem como a função de avaliação de um estado a partir das heurísticas apropriadas.

Em primeiro lugar, o algoritmo só vai gerar estados possíveis seguintes que sejam diferentes uns dos outros. Por exemplo, para o estado inicial (no início do jogo), existem 288 estados sucessores possíveis (36 posições possíveis para a colocação da peça e, para cada uma destas posições, 8 rotações possíveis sobre os tabuleiros de jogo). No entanto, o algoritmo deve ser capaz de descartar estados iguais (rotações sobre tabuleiros sem nenhuma peça, por exemplo, não resultam em estados diferentes). Isto faz com que, no estado inicial, sejam admitidos apenas 36 estados sucessores a percorrer (nenhuma rotação irá criar um estado diferente de todos os estados que existem apenas colocando a peça).

Em segundo lugar, a função de avaliação deve ser capaz de atribuir um valor a cada estado que seja admissível perante a noção de heurística. Isto significa que um estado final deve ter uma avaliação que seja sempre maior do que um estado não-final, ainda que este estado possa ser muito promissor. Da mesma forma, uma vez que não é possível remover peças e as linhas podem ser formadas em qualquer direção e sentido, um estado que contenha quatro peças da mesma cor juntas em linha nunca poderá ter uma avaliação inferior a um estado que contenha apenas três peças da mesma cor. Um estado com três possibilidades diferentes de vitória também não pode ser avaliado com pior nota do que um estado com menos possibilidades de vitória. A heurística, ainda que numa fase inicial de planeamento, deverá seguir a seguinte lista de prioridades:

- Movimentos que levem o jogador à vitória.
- Movimentos que impeçam o oponente de chegar à vitória.
- Movimentos que me aproximem da vitória (fazer dois, três ou quatro em linha).
- Movimentos que impeçam o oponente de se aproximar da vitória (bloquear ambas as extremidas de duas, três ou quatro peças em linha).
- Movimento aleatório.

Se a função de avaliação encontrar mais do que um estado sucessor com o mesmo valor, deve escolher aleatoriamente um desses estados.

Trabalho efetuado

Até à data de entrega deste relatório, das três fases da abordagem descritas anteriormente, duas encontram-se praticamente terminadas. A interface gráfica não está só concluída como já autoriza o utilizador da aplicação a configurar tudo da forma que bem entender.

A mecânica de jogo encontra-se completamente implementada, significando que já é possível iniciar um jogo, ter os turnos alternados entre os dois jogadores, haver validação de jogadas, assim como a verificação de fim de jogo. Já é também possível iniciar qualquer um dos três tipos de jogo (ainda que as jogadas do computador, neste momento, sejam apenas aleatórias).

O projeto encontra-se na 3ª fase de desenvolvimento. O algoritmo *minimax* sem cortes alfa-beta já se encontra implementado e a funcionar, faltando adicionar as heurísticas de jogo e adaptá-lo para aceitar cortes alfa-beta. Além disto, falta também implementar e testar o algoritmo *negamax* (as heurísticas serão as mesmas). Espera-se ainda desenvolver uma forma de ver o algoritmo a funcionar, ou pelo menos guardar e apresentar uma série de estatísticas que permitam avaliar a eficiência dos algoritmos já referidos.

Perante todo o trabalho feito até à data desta entrega, é estimado que foi concluído cerca de 80% do projeto na totalidade.

Resultados esperados e forma de avaliação

O principal resultado esperado é os vários algoritmos serem precisos e eficientes. Infelizmente, devido às limitações físicas de computação, não é aguardado que o algoritmo percorra e avalie todos os estados sucessores possíveis em qualquer altura do jogo (será estipulado um limite máximo de profundidade na pesquisa que irá devolver o valor do estado onde se encontra sempre que esse limite for alcançado). No entanto, são esperadas as seguintes condições na versão final do projeto:

- O jogador automático deverá ter uma inteligência que permita uma partida desafiadora para o jogador humano.
- No modo de jogo “Computador contra Computador”, o jogador com o nível de dificuldade mais alto deverá sempre ou, pelo menos, a maior parte das vezes, vencer o jogador com o nível de dificuldade mais baixo.
- O algoritmo nunca deverá atingir ciclos infinitos, estados não existentes, jogadas impossíveis ou demorar mais do que uma certa quantidade de tempo a devolver um resultado (para profundidades admissíveis).
- A versão do algoritmo *minimax* com cortes alfa-beta deverá demorar tanto ou preferencialmente menos tempo do que a versão sem cortes.

Existirão várias formas de avaliar o resultado final. Poderão ser feitos testes a casos pontuais, onde interessa averiguar se o algoritmo escolhe a jogada perfeita para o estado de jogo apresentado. Serão ainda realizados ciclos de jogo a partir de um estado qualquer com diferentes parâmetros, para assegurar que é atingido o estado final, e quem é o vencedor.

O utilizador terá a oportunidade de ver os resultados do algoritmo em tempo real (na zona branca em baixo do jogo), e visualizar estatísticas pertinentes tais como o número de estados percorridos, o tempo que demorou a percorrer esses estados, bem como a avaliação máxima encontrada. No relatório final serão depois documentados todos estes resultados.

Conclusões

O grupo é da opinião de que o desenvolvimento do projeto está a corresponder às expectativas. Poder responder a duas das três fases estipuladas nesta entrega intercalar é bastante positivo. Os próximos pontos de desenvolvimento passam por criar e testar as heurísticas de avaliação de estados de jogo e aplicá-las aos algoritmos de pesquisa, bem como fazer vários testes para assegurar que essas heurísticas são precisas e admissíveis. Perspetiva-se ainda uma fase final para aprimorar o código existente de modo a seguir as boas práticas de programação.

Recursos

Bibliografia:

1. Caso de estudo para a implementação do algoritmo *MiniMax*, “Tic-tac-toe AI”, http://www3.ntu.edu.sg/home/ehchua/programming/java/javagame_tictactoe_ai.html
2. Regras e estratégia de jogo, “Pentago Game Rules”, https://webdav.info.ucl.ac.be/webdav/ingi2261/ProblemSet3/PentagoRulesStrategy.pdf
3. Tutoriais de Swing, The Java ™ Tutorials, “Trail: Creating a GUI With JFC/Swing”, http://docs.oracle.com/javase/tutorial/uiswing/ (vários links).
4. GitHub – repositório com o código desenvolvido para o projeto (Branch Game), https://github.com/arcanjo45/IART

Software utilizado para a realização deste trabalho:

1. Eclipse Mars SDK (versão 4.5.1) – para a implementação do código.
2. Adobe Photoshop (versão CC 1.2) – para a edição das várias imagens utilizados no projeto.
3. Microsoft Word (Office 2015) – para a redação deste relatório.

Todas as imagens apresentadas neste relatório referem-se ao estado atual do projeto.