Jogo para Dois Jogadores
2B - NEUTRON

Diogo Silva
up201603438
up201603438@fe.up.pt

Inês Marques
up201605542
up201605542@fe.up.pt

Tomás Figueiredo
up201607932
up201607932@fe.up.pt

***Resumo*—Neste relatório, podemos encontrar uma breve descrição do jogo e algumas referências à forma como o pretendemos implementar.**

***Index Terms*—Inteligência Artificial, Pesquisa, Algoritmo MiniMax, Javascript, Adversários, Algoritmo $\alpha\beta$**

I. Introdução

Este relatório tem como principal objetivo, explicar as regras do jogo e a sua implementação em Javascript. Para tal, iremos descrever os problemas encontrados e a suas respetivas soluções na implementação dos algoritmos pedidos. Estruturando assim o relatório na descrição e formulação do problema, trabalho relacionado, conclusões e referências.

II. Descrição do Problema

O jogo implementado é o Neutron. O ***Neutron*** é um jogo no qual os jogadores têm um conjunto de partículas subatómicas e devem forçar um neutrão a deslocar-se para o seu lado. É um jogo abstrato para dois jogadores num tabuleiro quadrado, tipicamente, de 5x5 ou de 7x7. Cada jogador tem um conjunto de 5 peças (7 caso o tabuleiro seja maior) coloridas de preto ou vermelho. Existe também uma peça central, como explicado anteriormente, que representa o neutrão e é colorida de azul. As peças movem-se em linha reta numa das 8 direções e não podem parar até encontrarem um obstáculo. Para este trabalho, fizemos uma versão onde é possível os dois jogadores serem ambos humanos, bots ou um humano e outro bot.

III. Formulação do Problema

A. Representação do Estado

A representação do nosso jogo irá ter as seguintes estruturas:

1) Um objecto do tipo *GameManager* que não só tratará de preparar o jogo com as opções escolhidas pelo utilizador, mas também de lançar as jogadas realizadas pelos jogadores e fazer display das jogadas realizadas como quem foi o vencedor.
2) Um objecto do tipo *Game* que supervisionará o jogo, conhecendo sempre o estado actual do mesmo, como também se responsabiliza por realizar todas as acções relacionadas com o tabuleiro, como: dar todas as jogadas possíveis para uma determinada peça, coordenadas de uma peça no tabuleiro, colisões, estado do tabuleiro, etc.
3) Um objecto do tipo *AI* que tratará de realizar o algortimo *Minimax* de forma a realizar as suas jogadas.
4) Um objecto do tipo *Canvas* que tratará de fazer todos os displays relacionados com o jogo e também das interacções do utlizador com o tabuleiro.

B. Estado Inicial

Como referido anteriomente, o jogo começa com todas as peças de cada jogador posicionadas horizontalmente em bordas opostas do tabuleiro com o neutrão no centro do mesmo.

C. Testes Objetivo

O objectivo do jogo consiste em os jogadores tentarem mover o neutrão para a sua linha de origem ou bloqueá-lo completamente de forma a que o adversário não o consigo mover.
Assim, o teste objectivo baseia-se em verificar se a posição do neutrão e as posições à sua volta.

D. Operadores

Em cada jogada, o jogador tem as opções de mover a peça em todas as direções, incluido diagonais.

E. Pré-Condições

Para movimentar a peça é necessário primeiro que se mantenha dentro dos limites do tabuleiro e segundo, que não haja nenhuma peça a bloquear o caminho.

F. Efeitos

Cada um dos operadores vai alterar as coordenadas da peça selecionada, o operador ”cima”diminui a coordenada Y da peça e o operador ”baixo”aumenta, o operador ”esquerda”diminui a coordenada X da posição da peça e o operador ”direita”aumenta e as diagonais alteram ambas as coordenadas X e Y com a junção de ”baixo”e ”cima”.

G. Custo

O custo será o menor número de jogadas até chegar a solução adicionando também o menor número de jogadas do agente mais próximo da solução.

IV. Trabalho Relacionado

Criado por Robert A. Kraus, o jogo Neutron foi publicado pela primeira vez em Julho/Agosto de 1978, na magazine inglesa ***Games & Puzzles***.

Devido aos problemas de balanço do jogo, em que o primeiro jogador tem uma vantagem comparativamente ao segundo jogador, foi criada uma variante que visa equilibrar o jogo. Esta variante, denominada de *CoNeutron*, criada por Mark Own e Julian Richardson, difere no jogo em 2 aspetos:

1) a condição para vencer é levar o neutron para a linha do jogador oposto
2) não existe a exceção de na primeira jogada não ser possível mover o neutron

V. Implementação do Jogo

A. Classes e Organização de Código

Para o desenvolvimento deste projecto foi escolhida a linguagem *Javascript* devido à sua simplicidade e facilidade no que toca a interfaces gráficas. Temos uma interface para o utlizador escolher que tipo de jogo pretende jogar como também a janela gráfica onde o vamos desenhar. Em relação aos ficheiros utilizados, para além dos ficheiros index.html e os ficheiros .css necessários para estruturar a interface gráfica para o utilizador, temos 5 ficheiros: main.js, game.js, ai.js, vanvas.js e utils.js.

O ficheiro main.js trata de dar inicio ao jogo e supervisiona o decorrer da partida.

O ficheiro game.js contém os principais elementos base como a construção do board, o movimento das peças e a verificação do estado do jogo.

O ficheiro ai.js implementa todas as ações relacionadas com a inteligência nos modos de jogo onde são usados bots. É neste ficheiro que implementamos o minimax com cortes alfa-beta e usamo-lo para escolher a melhor jogada do bot. Este algoritmo usa a função heuristic que avalia qual a melhor jogada a ser efetuada. Foram implementados 3 níveis de dificuldade para jogos com I.A.

B. Algoritmos

O *Minimax* é um algoritmo para minimizar a possível perda máxima. Pode ser considerado como a maximização do ganho mínimo. Começa-se com dois jogadores, cobrindo ambos os casos em que os jogadores tomam caminhos alternados por jogada ou simultaneamente. Pode-se estender o conceito para jogos mais complexos e para tomada de decisão na presença de incertezas. Nesse caso, não existe outro jogador, as consequências das decisões dependem de fatores desconhecidos. O algorítimo *Minimax* pode fazer muito processamento, o que pode fazer com que o algorítimo seja lento, por isso, é importante fazer-se otimizações para que ele seja efetuado de forma rápida durante a execução do jogo. Para isso podem ser usados mecanismos como cortes alfa-beta, que foram ensinados nas aulas. Também pode ser importante, principalmente em dispositivos com baixo poder de processamento como telemóveis, ou até em dispositivos mais potentes, que os códigos sejam otimizados para evitar-se gastos com processamento e tempo desnecessários pois como este algoritmo faz muito processamento, principalmente nas análises das primeiras partidas, tais gastos podem ser muito aumentados e causarem um grande impacto na queda de desempenho.

C. Lógica Geral do Problema

Correndo o programa, o jogador escolhe entre os três tipos de jogo (**Player vs Player**, **Player vs Bot**, **Bot vs Bot**) e consoante a sua escolha, escolherá também a dificuldade do seu adversário. A não ser que o jogo seja entre dois bots, os jogadores têm a opção de se moverem ou moverem o neutrão. Os movimentos limitam-se apenas à escolha da direção, pois os jogadores só se podem mover e mover o neutrão uma casa por turno. Se existe um bot em jogo, ele vai avaliar a melhor jogada com a ajuda do algoritmo *Minimax*. A função *checkCompleted* vai estar constantemente a avaliar se um dos jogadores já chegou à vitória. Na consola aparece o tempo que o bot demorou a fazer uma jogada (no caso de haver bots em jogo) e o histórico do board.

VI. Experiências e Resultados

Para testarmos a eficiência do nosso algoritmo e implementação, corremos o jogo com mapas de diferentes tamanhos e em modos de jogo diferentes.

Assim, testamos as seguintes configuraçõess em situações de AI vs AI:

1) Ambos os AIs usam o algoritmo *Minimax*, sendo que ambos têm a mesma profundidade mas a heurística de um é melhor do que à do outro.
2) Ambos os AIs usam o algoritmo *Minimax*, sendo que ambos têm a mesma heurística mas a profundidade de um é superior à do outro.
3) Ambos os AIs usam o algoritmo *Minimax*, sendo que um tem uma profundidade menor e uma pior heurística e o outro tem uma profundidade maior e uma heurística melhor.

Nestas três situações verificamos que a profundidade usada no algoritmo *Minimax* tem um peso bastante grande no processamento da melhor escolha, sendo que no nosso caso (profundidade 3) o AI demora mais de 5 segundos a fazer a sua escolha. Verificamos também que em situações de profundidade idênticas, o tempo de execução acaba por ser o mesmo, mas aquele que tiver uma melhor heurística consegue jogadas superiores. Por fim, quando um AI utiliza tanto uma profundidade maior, como uma heurística melhor, apesar do elevado tempo de procura, as suas jogadas acabam sempre por o levar à vitória.

VII. Conclusões e Perspetivas De Desenvolvimento

No final do projeto, podemos dizer com confiança que percebemos a utilidade do algoritmo *Minimax* assim como a sua implementação com cortes alfa-beta.

Concluímos também que apesar de haver uma forte ligação

entre a profundidade de procura e o número de vitórias, este nem sempre é a melhor forma de implementar o algoritmo *Minimax*. Apesar de os cortes alfa-beta terem um peso muito grande na melhoria do desempenho do algortimo quando a profundidade aumenta, deve sempre haver um elevado foco na procura de uma boa e rubusta heurística de forma a possibilitar uma fluída execução mas mantendo uma boa capacidade de procura de solução. Em geral, estamos satisfeitos com o resultado final e com aquilo que aprendemos com o desenvolvimento deste projeto.

REFERÊNCIAS

[1] https://boardgamegeek.com/boardgame/6978/neutron
[2] https://en.wikipedia.org/wiki/Neutron$_{(game)}$