FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Universidade de Coimbra
Faculdade de Ciências Tecnologia
Licenciatura em Engenharia
Informática - Introdução à
Inteligência Artificial
2018/2019
3º Ano – 2º Semestre

RELATÓRIO
TP1

Fight for Dei or Die
MinMax

Coimbra, 1 de Abril de 2019

2016225773 João Pedro Santos Rodrigues uc2016225773@student.uc.pt
2016225581 Cláudio André Ventura Alves uc2016225581@student.uc.pt

Introdução

O presente relatório tem como objetivo clarificar e explicar as decisões tomadas em relação ao problema exposto no Trabalho Pratico nrº1, que pretende aprofundar a análise do algoritmo MinMax desenvolvido na sua forma canónica em comparação com o corte Alfa-Beta e também permitir a análise empírica das funções de avaliação e de utilidade desenvolvidas.

O MinMax é uma forma clássica de resolver problemas de Procura Adversarial presentes em maior parte dos jogos. O segredo para uma boa implementação reside no desenvolvimento cuidado das suas funções de utilidade e avaliação, como também na otimização do próprio MinMax, tornando assim possível o seu uso natural e imediato com aplicação real neste tipo de jogos.

Modelação do problema

O problema em questão passa pela criação de uma árvore de procura para cada jogada com o objetivo de, ao saber todos os possíveis movimentos do adversário, através da avaliação desses mesmo movimentos (função de avaliação e utilidade), escolher a melhor jogada possível.

No contexto deste problema, cada estado representa a disposição das unidades no campo e também os atributos de cada uma dessas unidades.

Esta árvore tem como nó inicial, o estado atual do campo naquele instante. Através da função generatePossibleStates() são gerados os estados seguintes, o que significa, as jogadas possíveis seguintes. Esta função é chamada recursivamente, mudando a perspetiva a cada geração, com o intuito de gerar uma árvore com todas as jogadas possíveis, tanto do player como do adversário. O que determinará se um estado é final ou não (folha da árvore) será o facto de um dos jogadores deixar de ter unidades com vida ou a árvore chegar ao limite de nós expandidos/ profundidade, dados pelo utilizador (maximumNodesToExpand / depth).

À medida que são criados nós folha, esses mesmos nós/estados são avaliados, ou através da função de avaliação ou através da função de utilidade, com o objetivo de, seguidamente, através do algoritmo MinMax recursivo, verificar qual a melhor jogada possível tendo em conta a árvore de jogadas criada.

Através desta modelação do problema é possível verificar que o que irá definir uma boa solução para o problema será a boa construção do algoritmo MinMax, mas principalmente uma função de avaliação e utilidade lógica e bem construída.

Função de utilidade

A função de utilidade tem como fim avaliar os estados onde o jogo acaba (ou o player fica sem unidades, ou o adversário fica sem unidades).

Esta função retorna valores no domínio [-999999 – depth, +999999 + depth]. Pontuação máxima, significa vitória (adversário com 0 unidades), pontuação mínima, significa derrota (player com 0 unidades).

Foi tomada a decisão de utilizar a depth como ajuda à pontuação destes estados, pois quanto mais acima na árvore, mais rápido o jogo acaba, o que influencia tanto positivamente como negativamente a tomada decisão. Isto é, se na jogada seguinte formos perder e seguindo outro ramo da árvore, formos perder apenas 3 jogadas à frente, é preferível escolher o ramo das 3 jogadas. Logo o nó mais acima vai ter menos pontuação que o nó mais abaixo e vice-versa, se ganharmos na jogada seguinte, essa jogada tem que ter mais pontuação do que se formos ganhar apenas daqui a 4 jogadas.

Validação do MinMax com e sem Corte Alfa-Beta

Mapa Debug	MinMax	10k N por jogada						
Attack Vs Random	Profundidade 1	Profundidade 2	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6	Profundidade 7	Profundidade 8
Tempo	21N 9T	190N 17T	399N 5T	1474N 5T	8778N 7T	23764N 5T	30005N 5T	30005N 5T
Mémoria	0	0	0	0	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso
Mapa Debug	MinMaxAlphaBeta	10k N por jogada						
Attack Vs Random	Profundidade 1	Profundidade 2	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6	Profundidade 7	Profundidade 8
Tempo	24N 11T	157N 15T	606N 13T	1501N 7T	3863N 5T	14254N 7T	42290N 9T	30000N 5T
Mémoria	0	0	0	0	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso
Mapa Debug	MinMax	10k N por jogada						
Run Vs Random	Profundidade 1	Profundidade 2	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6	Profundidade 7	Profundidade 8
Tempo	21N 9T	128N 11T	286N 5T	1639N 5T	6368N 5T	41964N 9T	40009N 7T	40013N 7T
Mémoria	0	0	0	0	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso
Mapa Debug	MinMaxAlphaBeta	10k N por jogada						
Run Vs Random	Profundidade 1	Profundidade 2	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6	Profundidade 7	Profundidade 8
Tempo	19N 9T	124N 9T	363N 7T	1119N 7T	10908N 19T	14139N 7T	34377N 7T	30001N 5T
Mémoria	0	0	0	0	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso	Sucesso

Foram feitos testes para validação do MinMax sem o corte alfa-beta e com o corte alfa-beta.

Estes testes foram sempre realizados no mapa debug contra a estratégica random (randomStrategy).

Nesta validação provámos que o MinMax com uma estratégia simples, ou só atacar ou só fugir, consegue facilmente ganhar a uma estratégia random.

Observa-se uma tendência na diminuição de turnos para ganhar à medida que é aumentada a profundidade, o que vai de encontro ao conceito do MinMax, quanto maior a árvore de procura, mais jogadas são possíveis de gerar, e consequentemente escolher o melhor caminho para a vitória (Sucesso).

Ao compararmos a proporcionalidade entre nós expandidos e número de turnos, concluímos que usando o corte alfa-beta no MinMax para o mesmo número de turnos, são expandidos menos nós, o que se torna numa vantagem computacional, tanto em termos de memória como em termos de rapidez de execução.

Funções de avaliação

As funções de avaliação servem para dar valores heurísticos aos estados que não são finais, mas estão no limite de expansão. Assim é possível, mesmo não chegando a um estado final, escolher o melhor caminho possível.

Num mapa grande com várias unidades, a função de utilidade é raramente chamada, pois seria necessária uma árvore bastante grande para conseguir chegar a um nó final (à vitória). E é nestes casos que se reflete a importância da função de avaliação, pois o objetivo destes algoritmos de Inteligência Artificial é serem computacionalmente rápidos para criar um contexto de jogo real, logo a sua árvore de procura não pode ser muito grande e então são necessárias boas funções de avaliação para ser possível escolher a melhor jogada possível.

Map0	MinMaxAlphaBeta	30k N por jogada	Max75T	
Attack vs Random	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6
Tempo	6906N 39T	20195N 75T	266602N 41T	451937N 75T
Mémoria	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso

Map0	MinMaxAlphaBeta	30k N por jogada	Max75T	
Run vs Random	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6
Tempo	17989N 75T	74913N 75T	171101N 75T	923306N 75T
Mémoria	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Fail

Attack Only

Esta estratégia tem como via a vida perdida das unidades do adversário e a eliminação de unidades do mesmo, ou seja, quanto mais vida as unidades perderem no seguimento das jogadas e quanto mais elementos do adversário forem eliminados, melhor. É priorizado a eliminação de unidades do adversário e também o ataque aos Assassins e aos Mages, pois estes são as melhores unidades de ataque, logo quanto mais rápido forem eliminadas, melhor.

Score = A + B + 2*C + 2*D + 1000*E

Onde A é a vida perdida dos Protectors, B é a vida perdida dos Warriosr, C é a vida perdida dos Mages, D a vida perdida dos Assassins e E é a diferença entre as unidades do Player e do adversário.

Esta função de avaliação foi validada através de testes no mapa Debug e Map0 contra a estratégia Random. É possível verificar que a taxa de sucesso é de 100%. Em termos de tempo é possível verificar que ao aumentar a profundidade, o número de nós expandidos também aumenta e em relação à memória, a única memória usada é apenas e só a da árvore, pois todas as verificações da função de avaliação têm em conta os atributos momentâneos do estado a avaliar.

Run Only

Esta estratégia tem como via a vida perdida das unidades do Player e a não eliminação de unidades do mesmo, ou seja, quanto menos vida as unidades do player perderem no seguimento das jogadas e quanto menos unidades o player perder, melhor. É priorizado a preservação de unidades e o não ataque aos Assassins e aos Mages, pois estes são as melhores unidades de ataque, logo se não forem eliminadas, melhor.

Score = -A - B - 2*C - 2*D – 1000*E

Onde A é a vida perdida dos Protectors, B é a vida perdida dos Warriosr, C é a vida perdida dos Mages e D a vida perdida dos Assassins e E é a diferença entre as unidades do adversário e do Player.

Esta função de avaliação foi validada através de testes no mapa Debug e Map0 contra a estratégia Random. É possível verificar que a taxa de sucesso diminui com a profundidade. Em termos de tempo é possível verificar que ao aumentar a profundidade, o número de nós expandidos também aumenta e em relação à memória, a única memória usada é apenas e só a da árvore, pois todas as verificações da função de avaliação têm em conta os atributos momentâneos do estado a avaliar.

Strategy

Esta estratégia tem como objetivo uma defesa equilibrada em conjunto com um ataque em bloco, tirando principalmente partido dos erros do adversário. Esta função passa por uma junção/aperfeiçoamento das estratégias definidas anteriormente(apenas atacar e apenas fugir), tendo também agora em conta os bónus dados pelos Protectors e pelos Warriors e tentando dar mobilidade aos Assasins e aos Mages para estes não se encurralarem, pois são unidades chave para a vitória devido as suas características ofensivas.

ScoreA= -A+ 10* Bonus

Bonus é um valor retornado pela função CountBonus onde conta o número de bónus atribuído a unidade em questão, priorizando o bónus de ataque. Bonus =1* NumeroDeHpBonus + 2* NumeroDeAtackBonus

ScoreA é o score relacionado com os Protectors, onde A é a vida que estes perderam.

ScoreB= -B + 10* Bonus

ScoreB é o score relacionado com os Warriors, onde B é a vida que estes perderam.

Strategy (continuação)

ScoreC = 5* NumeroDeCasasVizinhasDisponiveis + 25*NumeroDeMages – 3*C + 30*Bonus

ScoreC é o score relacionado com os Mages, onde C é a vida que estes perderam.

ScoreD=10*NumeroDeCasasVizinhasDisponiveis+50*NumeroDeAssassins – 4*D + 30*Bonus

ScoreD é o score relacionado com os Assasins, onde D é a vida que estes perderam.

ScoreE= 1000*E

ScoreE é a diferença entre as unidades do adversário e do Player.

ScoreF= -A' – B' - 2*C' - 2*D' – 25*NumeroDeMages – 50* NumerodeAssasins

Onde A' é a vida perdida dos Protectors, B' é a vida perdida dos Warriosr, C' é a vida perdida dos Mages e D' a vida perdida dos Assassins. Referente às personagens do adversário.

Score= ScoreA + ScoreB + ScoreC + ScoreD + ScoreE+ ScoreF

Map0	MinMaxAlphaBeta	30k N por jogada	Max75T	
Strategy vs Random	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6
Tempo	16738N 75T	113022N 75T	174825N 31T	247179N 23T
Mémoria	0	0	0	0
Resultado	Sucesso	Sucesso	Sucesso	Sucesso
Map0	MinMaxAlphaBeta	30k N por jogada	Max75T	
Strategy vs Attack	Profundidade 3	Profundidade 4	Profundidade 5	Profundidade 6
Tempo	36344N 75T	58856N 75T	Bug	1226412N 75T
Mémoria	0	0	Bug	0
Resultado	Sucesso	Sucesso	Bug	Sucesso

Esta função de avaliação foi validada através de testes no mapa Map0 contra a estratégia Random e contra a estratégia Attack. É possível verificar que a taxa de sucesso é de 100% o que vai de encontro ao nosso objetivo final, ter uma estratégia vencedora. Em termos de tempo é possível verificar que ao aumentar a profundidade, o número de nós expandidos também aumenta e em relação à memória, a única memória usada é apenas e só a da árvore, pois todas as verificações da função de avaliação têm em conta os atributos momentâneos do estado a avaliar.

Strategy (continuação)

Em suma, foram feitos bastantes testes ao longo do desenvolvimento desta função de avaliação, o que nos levou a optar por tomar estas decisões, chegando à conclusão que neste jogo um pequeno erro pode levar à derrota e ao aproveitar os pequenos erros do adversário torna a vitória muito mais fácil.

Conclusão

Com este projeto, aumentámos bastante o nosso conhecimento acerca do Algoritmo MinMax e da construção de funções de avaliação e utilidade lógicas. No MinMax, a profundidade da árvore de procura é um conceito bastante importante pois quanto mais jogadas possíveis seguidas forem simuladas, melhor é a escolha da melhor jogada possível. O corte alfa-beta é fulcral para se conseguir alcançar maior profundidade em menor tempo, cortando ramos da árvore que são desnecessários de analisar e assim, tornar o algoritmo mais adaptável à realidade.