Resolução do Labyrinth Robots utilizando Métodos de Pesquisa em Linguagem Java (Tema 7/Grupo 18)

1st César Pinho
Universidade do Porto
FEUP
Porto, Portugal
up201604039@fe.up.pt

2nd João Barbosa
Universidade do Porto
FEUP
Porto, Portugal
up201604156@fe.up.pt

3rd Rui Guedes
Universidade do Porto
FEUP
Porto, Portugal
up201603854@fe.up.pt

***Resumo*—No âmbito da disciplina de Inteligência Artificial, pretende-se desenvolver um programa em Java capaz de, através da utilização de diferentes métodos de pesquisa e heurísticas/conhecimento, resolver um jogo do tipo solitário, comparando posteriormente o seu desempenho. *Labyrinth Robots* foi o jogo escolhido. Este jogo consiste num puzzle 2D cuja representação é feita através de uma matriz bidimensional de tamanho variável. Em determinadas células dessa matriz encontram-se robôs que necessitam ser movimentados para células alvo não coincidentes com a posição inicial dos robôs.**

A descrição e a formulação deste jogo encontra-se detalhadamente descrita no presente artigo com o objetivo de explicitar o jogo escolhido com um modelo próximo da implementação efetuada a posteriori usando como base diversas fontes de trabalho relacionado. Escolheram-se cinco algoritmos de pesquisa, três de pesquisa não informada e dois de pesquisa informada, com duas heurísticas, para testar e comparar o seu desempenho neste jogo. Analisando os resultados obtidos no conjunto de problemas testados, concluímos que os algoritmos de pesquisa informada obtiveram melhor desempenho no conjunto de testes realizados.

***Keywords*—Inteligência Artificial, *Labyrinth Robots*, *Ricochet Robots*, Pesquisa em Profundidade, Pesquisa em Largura, Aprofundamento Progressivo, Pesquisa Gulosa, Algoritmo A*, Pesquisa Heurística.**

I. INTRODUÇÃO

Este projeto tem como objetivos, primeiramente, o desenvolvimento de um programa para a resolução de um problema recorrendo a diferentes métodos de pesquisa e heurísticas/conhecimento e, posteriormente, a comparação do desempenho dos algoritmos utilizados. A decisão do problema a abordar recaiu sobre o jogo Labyrinth Robots, um puzzle bidimensional cujo objetivo consiste na movimentação de robôs desde as suas posições iniciais até às suas posições finais (alvos) num dado mapa.

Quanto às estratégias de pesquisa, irão ser utilizados algoritmos de pesquisa não informada como a pesquisa em profundidade, pesquisa em largura e pesquisa com aprofundamento progressivo, bem como algoritmos de pesquisa informada, isto é, que utilizam heurísticas e conhecimento para alcançar a solução, sendo a pesquisa gulosa e o algoritmo A* algoritmos representativos deste tipo de pesquisa e que serão utilizados neste projeto.

O presente artigo começa por descrever o problema em questão na sua totalidade, possibilitando a compreensão do mesmo bem como os algoritmos e métodos a serem utilizados para o resolver, sendo que depois são apresentados trabalhos relacionados que serão tidos em consideração na implementação.Segue-se uma descrição da implementação do puzzle e dos algoritmos de pesquisa, usando os referidos artefactos, culminando numa apresentação dos resultados obtidos sobre a experimentação dos diferentes algoritmos e heurísticas, tirando conclusões sobre os mesmos.

De modo a seguir o conteúdo acima apresentado, o artigo encontra-se estruturado da seguinte forma: descrição do problema, formulação do problema como um problema de pesquisa, trabalhos relacionados, implementação do jogo, algoritmos de pesquisa, experiências e resultados, e conclusões e perspetivas de desenvolvimento, finalizando o documento com referências bibliográficas dos trabalhos referidos no presente artigo.

II. DESCRIÇÃO DO PROBLEMA

Labyrinth Robots é um jogo de resolução de puzzles na Google Play desenvolvido por Emil Fridell. Este tipo de puzzle é parecido com o Ricochet Robots, um jogo de tabuleiro de 1999 onde jogadores competem entre si na procura da melhor solução para um determinado problema.

Um problema deste género é caracterizado por um tabuleiro quadrangular com espaços livres e paredes. Neste ambiente, é colocado um certo número de robôs num destes espaços livres e, para alguns destes robôs, é selecionado um espaço livre no tabuleiro como um alvo do robô. Para resolver o problema, é necessário colocar cada robô no seu respetivo alvo, caso lhe tenha sido atribuído um.

Fig. 1. Exemplo de um problema retirado do jogo. Para resolver o problema, é necessário deslocar os robôs amarelo, verde e azul para o alvo da sua respetiva cor (O robô branco e o robô vermelho não têm alvo atribuído)

O interesse do problema provêm do movimento restrito dos robôs. Um robô pode-se mover nas quatro direções do tabuleiro, à semelhança de uma Torre de Xadrez, mas com a limitação de se mover nessa direção até ficar encostado a uma parede ou a outro robô.

Fig. 2. Exemplo a salientar os possíveis movimentos do robô amarelo. Apenas a direção pode ser selecionada, sendo que o robô passa a ocupar a última célula salientada nessa direção.

Este sistema de movimento, dependente do ambiente, e o uso dos próprios robôs como uma parte do ambiente, tornam este problema atrativo para testar diferentes algoritmos de travessia de árvores na resolução deste problema como um problema de pesquisa.

III. FORMULAÇÃO DO PROBLEMA

A resolução deste problema requer que seja encontrado um estado do tabuleiro, atingível a partir do estado inicial, que tenha as propriedades acima referidas necessárias para que o problema possa ser considerado resolvido. Assim sendo, estamos perante um problema de pesquisa.

A. Representação do Estado

Para a representação dum estado do problema, é necessário ter conhecimento do seu ambiente nesse momento (Tabuleiro, Robôs e Alvos). Destes três elementos, apenas os Robôs são dinâmicos, ou seja, sofrem alterações de estado para estado. Assim sendo, de modo a minimizar a redundância e, consecutivamente, a memória despendida, apenas a posição atual dos robôs fará parte da representação do estado, fora os dados relacionados com a implementação em si, não referidos nesta formulação. Representado uma única vez para todo o problema (P), o tabuleiro (B) será uma matriz de células, sendo que cada uma destas células pode ser um espaço livre (“ “) ou uma parede (”W”). As posições dos alvos (T_i) e dos robôs (R_i), sendo estas últimas correspondentes à representação do estado (S), serão guardados como listas de posições (A correspondência entre os alvos e robôs será feita através do índice). Cada posição é representada por duas medidas, a distância horizontal (x) e vertical (y) dessa posição à célula no canto superior esquerdo do tabuleiro.

Fig. 3. Neste exemplo, o robô branco tem a posição (7, 6) e o alvo verde uma posição (10, 5).

B. Estado Inicial

As posições iniciais dos robôs são especificadas pelo problema a resolver, tal como a informação estática não dependente do estado (o tabuleiro e as posições dos alvos).

C. Teste Objetivo

Um estado corresponde a um objetivo (Resolução do problema) se, para todos os alvos do problema, o robô correspondente está na mesma posição que esse alvo, ou seja:

$$S \rightarrow Goal : \forall T_i \in P, R_i = T_i \quad (1)$$

D. Operadores

Na transição de um estado S_{init} para um estado S_{end}, é necessário primeiro escolher um robô a mover e, de seguida, a direção para onde esse robô se mexe. Desta forma, para $\forall R_i \in S_{init}$, existem os seguintes operadores:

Nome	Pré-condições	Efeitos	Custo
SlideLeft	$B[R_i.x - 1, R_i.y] \neq "W"$	(1)	1
SlideRight	$B[R_i.x + 1, R_i.y] \neq "W"$	(2)	1
SlideUp	$B[R_i.x, R_i.y - 1] \neq "W"$	(3)	1
SlideDown	$B[R_i.x, R_i.y + 1] \neq "W"$	(4)	1

Efeitos

$$while(B[--R_i.x, R_i.y] \neq "W"); R_i.x++;$$
$$while(B[++R_i.x, R_i.y] \neq "W"); R_i.x--;$$
$$while(B[R_i.x, --R_i.y] \neq "W"); R_i.y++;$$
$$while(B[R_i.x, ++R_i.y] \neq "W"); R_i.y--;$$

IV. TRABALHO RELACIONADO

Existe um repositório com código fonte do livro base da disciplina em Java [3]. Neste repositório podemos encontrar implementações genéricas dos algoritmos de pesquisa a aplicar neste projeto, adaptado às necessidades do nosso problema e dos nossos objetivos. Em relação ao Ricochet Robots, o jogo de tabuleiro com mecânicas idênticas ao Labyrinth

Robots, existe um artigo de 2005 que estuda a resolução deste problema sobre tanto o ponto de vista humano como do computador [4]. Deste artigo, há relevância no ponto 3.3, onde são referidas heurísticas inspiradas no humano. Michael Fogleman, um engenheiro de software com diversos projetos relacionados com puzzles e IA, dedicou tempo no Ricochet Robots, partilhando as suas decisões de implementação num conjunto de slides. As heurísticas e otimizações deste artefacto poderão vir a ser úteis [5].

V. IMPLEMENTAÇÃO DO JOGO

Com o objetivo de experimentar diferentes algoritmos de pesquisa no Labyrinth Robots, desenvolveu-se uma versão do jogo em Java. Neste sentido, existem três classes principais; a classe ***Level***, que representa o problema em si, a classe ***State***, que representa um estado específico do problema, e a classe ***Data***, que contém a informação sobre um dado robô e o seu alvo , denominado daqui adiante por Agente do problema.

O ambiente do jogo, dividido anteriormente em três componentes (Tabuleiro, Robôs e Alvos), é representado nestas três classes. Na classe ***Level***, estão guardados a representação do tabuleiro sob a forma de um *array* bidimensional, bem como o estado atual do problema. Um estado do problema contém uma tabela de dispersão com os vários Agentes do problema. Os alvos, informação independente do estado, são assim guardados junto das posições dos robôs por cada estado. Optou-se por implementar desta forma, de modo a facilitar a implementação da lógica de jogo necessária para aplicar os algoritmos de pesquisa, tendo em conta que o objetivo final é a análise comparativa entre diferentes algoritmos e heurísticas, e não a otimização do desempenho do Solver.

As diferentes maneiras de expandir um estado (Operadores) foram implementadas através de dois métodos; ***Level::getActions*** que retorna todos os operadores que podem ser aplicados ao estado atual, verificando as pré-condições, e ***Level::getResult*** que retorna um estado sucessor, resultado da aplicação dum operador (Action) ao estado atual. A verificação das pré-condições para os operadores e a atualização do estado é efetuada na classe ***Data*** para cada Agente específico (***Data::getActions*** e ***Data::action***). Como todos os operadores têm o mesmo custo, não há necessidade de manter o custo de cada operador (sendo que a profundidade do estado reflete o próprio custo).

O teste objetivo é implementado através do método ***Level::testGoal***, que verifica, para cada Alvo, se foi atingido pelo Robô correspondente (***Data::cmp***).

Para permitir ao utilizador visualizar de forma amigável o puzzle, foi implementado o método ***Level::display***, que imprime na linha de comandos uma representação de fácil visualização do problema.

De modo a poder comparar o desempenho dos algoritmos e heurísticas selecionados para diferentes problemas, é possível importar o problema a partir dum ficheiro, com os dados em formato matricial, semelhante à sua representação visual, através do método ***Level::readFile***.

VI. ALGORITMOS DE PESQUISA

Implementado o jogo, é necessário proceder à implementação de vários algoritmos e heurísticas a serem testados e comparados. O repositório de algoritmos de IA em Java referido à priori [3] tem já uma implementação genérica dos vários algoritmos que pretendemos usar. Usando este recurso, os algoritmos foram rapidamente implementados, sendo apenas necessário adicionar as heurísticas pretendidas para o projeto e *memoization* de estados para impedir árvores infinitas, referidas adiante.

Escolheram-se cinco algoritmos, três de pesquisa não informada e dois de pesquisa informada, estes últimos utilizando heurísticas. Quanto à pesquisa não informada, implementou-se a pesquisa em largura e em profundidade (algoritmos de travessia de grafos aplicados a árvores de pesquisa), bem como aprofundamento progressivo, um algoritmo de pesquisa que usa várias iterações de pesquisa em profundidade limitada, com o limite a incrementar a cada iteração. Desta forma, alguns nós da árvore são explorados múltiplas vezes, mas ramos demasiado profundos da árvore não são percorridos desnecessariamente. A pesquisa em largura tem também esta vantagem, sem o processamento extra, mas a fronteira da árvore a ser gerada cresce exponencialmente. A pesquisa de custo uniforme não é usada, pois, tendo em conta que os operadores têm todos o mesmo custo, o seu funcionamento é igual à pesquisa em largura (expandir os estados com menor profundidade primeiro).

Quanto aos algoritmos de pesquisa informada, utilizou-se um algoritmo guloso e A*. Ambos usam uma função de avaliação para decidir o próximo estado na fronteira da árvore a expandir, com a diferença de que o A*, para além de avaliar o estado usando uma heurística, tem também em consideração o custo até chegar a esse estado, correspondente, neste caso, à profundidade desse mesmo estado. Para estes algoritmos, foram utilizadas duas heurísticas; uma mais básica que avalia o alinhamento dos robôs com os alvos em ambas as coordenadas (***Level.AgentAlignmentHeuristic*** e ***Data::getAlignmentValue***), e outra mais avançada que pré-processa o problema e aponta, para cada célula do tabuleiro, o número de movimentos necessários para atingir cada alvo, assumindo que os robôs podem parar durante o seu movimento ao estilo de uma Torre de Xadrez (***Level.FreeMovementHeuristic***, ***Level::initPreProcess*** e ***Data::preProcessMatrix***). Esta última heurística foi proposta por Fogleman [5].

Para garantir um bom funcionamento dos algoritmos, sobretudo a pesquisa em profundidade e gulosa, é necessário garantir que não existem ciclos (ramos com profundidade infinita). Para tal, basta fazer *memoization* dos estados percorridos. Assim, é facilmente detetado se um estado já foi percorrido, pelo que o corte desse ramo impede o processamento de estados repetidos (***Level.searchInfo*** e ***Level::getActions***). O uso de *memoization* é uma das várias otimizações indicadas por Fogleman [5].

VII. EXPERIÊNCIAS E RESULTADOS

Usando um conjunto de problemas definidos em ficheiros, testaram-se os cinco algoritmos para cada problema, recolhendo métricas sobre o seu desempenho, entre estas, o tempo (ms) e memória gastos (MB), e o número de estados expandidos. De modo a tornar a comparação mais justa, desativou-se o embaralhamento dos operadores. Os resultados obtidos podem ser visualizados nas seguintes tabelas:

Algoritmo	Tempo	Memória	Custo	Nós expandidos
BFS	53	5,2	6	145
DFS	944	147,5	50	62032
IDS	151	2,8	6	377
Greedy [1]	36	4,2	9	18
Greedy [2]	27	4,2	9	9
A* [1]	50	4,7	6	45
A* [2]	29	4,2	6	7

Tabela 1: Resultados obtidos num puzzle simples com 2 robôs e 2 alvos (Nível 5)

Algoritmo	Tempo	Memória	Custo	Nós expandidos
BFS	3286	166	13	153350
DFS	1093	90,2	26212	29626
IDS	15689	93,0	13	1085172
Greedy [1]	51	4,7	15	25
Greedy [2]	101	7,3	13	152
A* [1]	1046	130,9	13	26132
A* [2]	386	30,9	13	3916

Tabela 2: Resultados obtidos num puzzle de dificuldade média com 3 robôs e 3 alvos (Nível 15)

Algoritmo	Tempo	Memória	Custo	Nós expandidos
BFS	–	–	–	–
DFS	–	–	–	–
IDS	–	–	–	–
Greedy [1]	313	44,6	77	2444
Greedy [2]	399	59,4	78	3035
A* [1]	–	–	–	–
A* [2]	14425	981,5	23	408616

Tabela 3: Resultados obtidos num puzzle de dificuldade alta com 4 robôs e 3 alvos (Nível 25)

1 - *AgentAlignmentHeuristic*.
2 - *FreeMovementHeuristic*.

Recorrendo às três tabelas anteriores é possível observar que os três algoritmos não informados obtiveram resultados maioritariamente em conformidade com o expectável, dum ponto de vista teórico. A pesquisa em largura teve resultados consistentes com a sua análise de tempo e memória assintótica.

Por outro lado, a pesquisa em profundidade obteve resultados dispersos. Devido à natureza do problema, a árvore gerada é normalmente altamente desequilibrada. Assim, para certos problemas (sobretudo problemas de dimensão reduzida), a pesquisa em profundidade poderá ocupar mais memória do que a pesquisa em largura devido à enorme amplitude de profundidades dos ramos da árvore (O custo no segundo puzzle e os nós expandidos no primeiro dá-nos uma ideia do quão profundos alguns destes ramos podem chegar a ser).

O aprofundamento iterativo combate este problema, já que são definidos limites de profundidade para cada pesquisa em profundidade, mas a sua complexidade temporal aparenta piorar mais do que a pesquisa em largura pelos resultados obtidos. Uma possível razão para isto tem a haver com o mecanismo para estados duplicados (*memoization*) que não é tão eficaz na pesquisa em profundidade (é possível ter de processar estados repetidos porque podem ter menor custo, algo que não acontece na pesquisa em largura, em que o estado processado é definitivamente aquele com menor custo).

Os algoritmos de pesquisa informada têm um desempenho substancialmente melhor do que os de pesquisa não informada, sendo que, para problemas de grande dimensão, tornam-se nos únicos algoritmos capazes de encontrar uma solução num espaço de tempo razoável. O algoritmo guloso é mais eficaz do que o A*, mas não garante a solução ótima na maioria dos casos. Mesmo assim, as heurísticas usadas provaram ser eficazes em obter boas soluções em problemas com dificuldade razoável. Em problemas complicados, as heurísticas são capazes de muito rapidamente obter uma solução, sendo úteis quando a sua qualidade é pouco relevante. Entre as duas heurísticas, a mais sofisticada obteve melhores resultados sendo normalmente capaz de guiar melhor o processo de pesquisa do que a heurística básica. Esta melhoria, faz com que esta heurística consiga obter melhor desempenho mesmo em problemas de curta dimensão, apesar do pré processamento inicial envolvido.

VIII. CONCLUSÕES E PERSPETIVAS DE DESENVOLVIMENTO

De modo a compreender melhor as aplicações e vantagens de algoritmos de pesquisa, escolhemos um jogo do tipo solitário (Labyrinth Robots), formulamo-lo como um problema de pesquisa, pesquisamos por trabalhos realizados em jogos semelhantes e implementações de algoritmos de pesquisa e, usando-os como guia, implementou-se o jogo e cinco algoritmos de pesquisa em Java, testando cada um deles em diversos puzzles.

Pelos resultados obtidos, concluímos que os algoritmos de pesquisa informada são mais eficientes e que o desempenho dos algoritmos está em conformidade com os resultados teóricos, com exceção da pesquisa em profundidade, provavelmente devido ao desequilíbrio da árvore e do processamento de uma árvore com mais estados, dado que os cortes não são efetuados de forma eficiente, devido ao método de travessia da árvore subjacente ao algoritmo.

Como trabalho futuro, um método de memorização de estados mais eficiente (que permita, por exemplo, reutilizar a sub-árvore de um estado já processado quando este reaparece mas com um custo inferior) ou um ordenamento melhor dos operadores, baseado numa análise teórica mais funda do problema poderão melhorar o desempenho da pesquisa em

profundidade e do aprofundamento iterativo, aproximando-os mais do seu desempenho teórico. No ponto de vista da implementação, esta pode ser bastante otimizada, no caso de se pretender testar estes algoritmos em problemas maiores, tendo em conta que este não foi o principal fator tido em conta durante a implementação, nem foi efetuado nenhuma análise à posteriori com este intuito.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Play.google.com. (2019). [online] Available at: https://play.google.com/store/apps/details?id=com.FridellGames.LabyrinthRobotsGame [Accessed 17 Mar. 2019].

[2] BoardGameGeek. (2019). Ricochet Robots. [online] Available at: https://boardgamegeek.com/boardgame/51/ricochet-robots [Accessed 17 Mar. 2019].

[3] GitHub. (2019). aimacode/aima-java. [online] Available at: https://github.com/aimacode/aima-java?fbclid=IwAR0rlZcEBg9rBzeuoziqQCLOFvrhP1vOKHWc4D2j7zRnwbw-Za566lxB110 [Accessed 17 Mar. 2019].

[4] Algo.inf.uni-tuebingen.de. (2019). [online] Available at: http://www.algo.inf.uni-tuebingen.de/mitarbeiter/katharinazweig/downloads/ButkoLehmannRamenzoni.pdf [Accessed 17 Mar. 2019].

[5] Speaker Deck. (2019). Ricochet Robots: Solver Algorithms. [online] Available at: https://speakerdeck.com/fogleman/ricochet-robots-solver-algorithms [Accessed 29 Mar. 2019].