FACULDADE DE CIÊNCIAS E TECNOLOGIA
DA UNIVERSIDADE DE COIMBRA
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

2015/2016
2º Semestre

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL
TRABALHO PRÁTICO 2
SOKOBAN

Ana Inês Mesquita Fidalgo – 2013134819 – aimf@student.dei.uc.pt – PL6
Andreia Filipa Palma Gonçalves – 2007018949 - andreiag@student.dei.uc.pt – PL4
Pedro Filipe Matos Godinho Gabriel Coelho – 2009116949 - pfcoelho@student.dei.uc.pt – PL2

13 de abril de 2016

ÍNDICE

INTRODUÇÃO

O segundo trabalho prático da unidade curricular de Introdução à Inteligência Artificial incidiu sobre agentes de procura aplicados ao jogo Sokoban. Novamente, o trabalho foi realizado no motor de jogos Unity com recurso à linguagem de programação C#.

No presente relatório descrevemos todo o trabalho desenvolvido. Em primeiro lugar, o jogo foi analisado e modelado como um problema de procura. De seguida, a partir do ficheiro fornecido pelos docentes, o qual já continha o algoritmo de largura primeiro, implementámos mais dois algoritmos de pesquisa cega: profundidade limitada e aprofundamento progressivo, e dois algoritmos de pesquisa informada: A* e pesquisa sôfrega. Para estes últimos desenvolvemos não só a heurística pedida no enunciado, como também quatro novas.

Posto isto, começámos a fase de experimentação. Testámos os nossos algoritmos para os quatro mapas fornecidos, assim como para dois novos que criámos. Após recolha dos dados, procedemos à análise teórica dos algoritmos comparativamente aos que obtivemos.

MODELAÇÃO

a. Breve introdução

O Sokoban é um jogo do tipo quebra-cabeças, no qual o jogador movimenta um determinado número de caixas para posições indicadas no mapa. O jogo termina quando todas as caixas estiveram nas posições finais. Deve obedecer a algumas regras:

- Não existem movimentos na diagonal, ou seja, o jogador só anda na vertical e horizontal, movimentando-se apenas uma célula de cada vez;
- O jogador só pode empurrar as caixas e uma de cada vez;
- O jogador não pode estar na mesma célula que uma caixa, assim como não há duas caixas numa. As paredes são obstáculos e limitam o mapa.

Para o nosso trabalho prático, foram fornecidos quatro mapas (Figuras 1 a 4) e desenvolvemos dois novos (Figuras 5 e 6). O jogador é representado por um pássaro e as posições objectivo por estrelas.

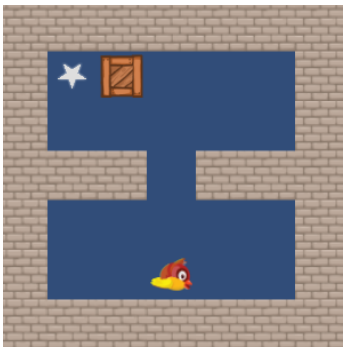

Figura 1 – Mapa 1

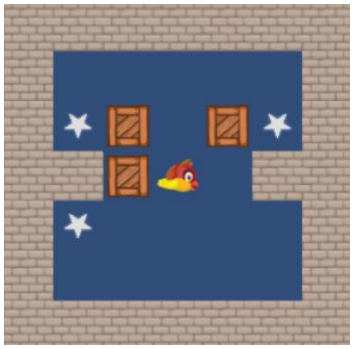

Figura 2 – Mapa 2

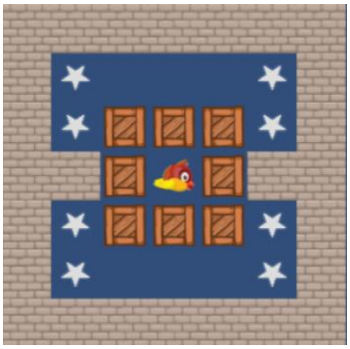

Figura 3 – Mapa 3

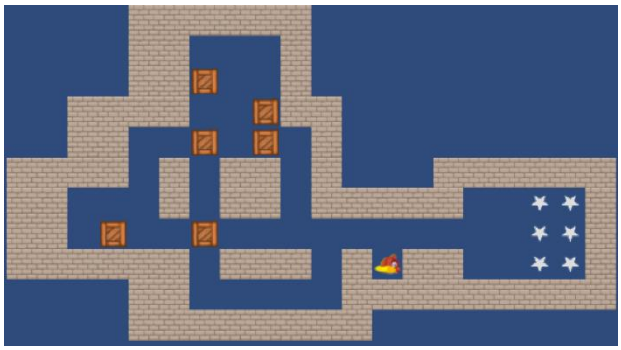

Figura 4 – Mapa 4

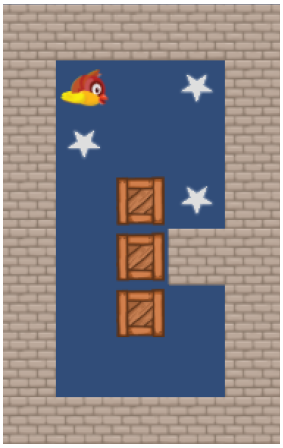

Figura 5 – Mapa 5

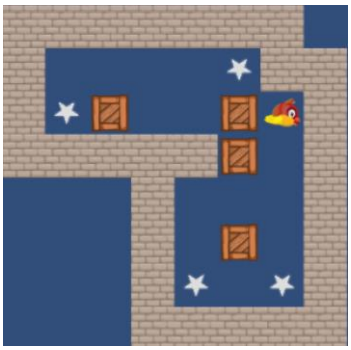

Figura 6 – Mapa 6

b. Problema de procura

O jogo pode ser modelado como um problema de procura. Para tal, temos as seguintes definições:

- Estado – o pássaro e as caixas numa certa posição no mapa. De referir que, devem estar em posições legais, isto é, segundo as regras definidas anteriormente;
- Estado inicial – mapa inicial, onde todas as caixas estão fora das estrelas e o pássaro ainda não se moveu (ver Figura 7);
- Estado final – todas as caixas estão nas respectivas estrelas (ver Figura 7);
- Operadores de mudança de estado – todos os movimentos possíveis, ou seja, Norte, Sul, Este, Oeste (nenhum elemento se pode mover na diagonal);
- Solução pretendida – todas as operações que mudem o estado inicial para estado final. Neste caso, todos os movimentos que o pássaro faz colocar as caixas nas estrelas;
- Custo associado – visto que o jogador apenas pode fazer um movimento de cada vez, o custo associado é de um;
- Heurísticas aplicáveis – relacionadas com cálculos de distâncias entre jogadores e caixas e entre caixas e posições objectivos; devem solucionar o problema com a solução mais económica.

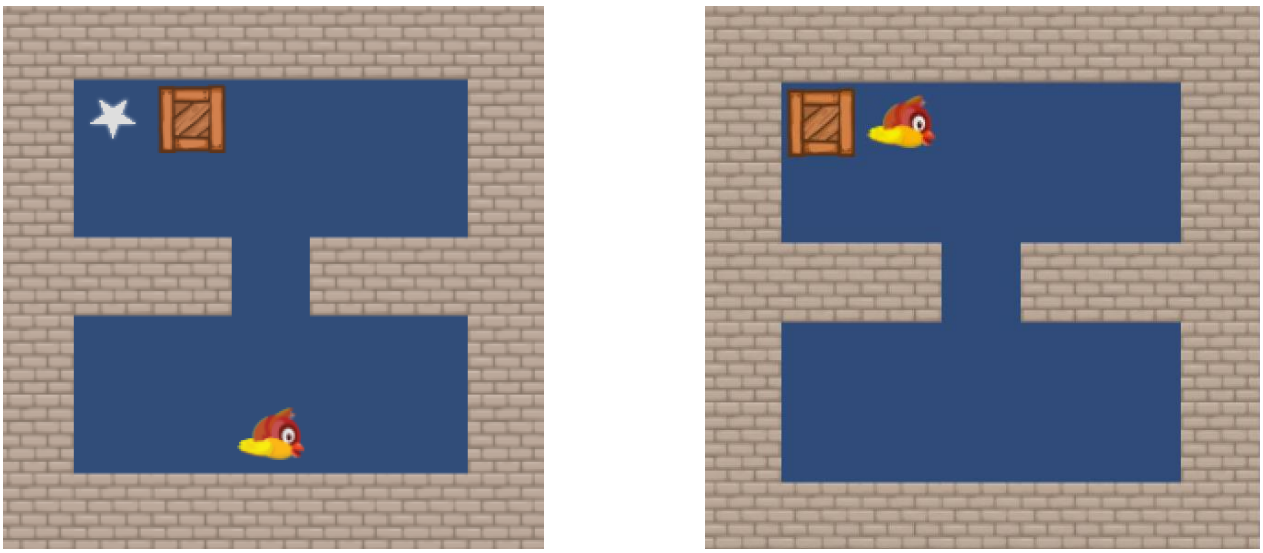

Figura 7 – Estado inicial e final, respectivamente

ALGORITMOS DE PROCURA

a. Procura cega

Os algoritmos de procura cega apenas se limitam a encontrar soluções por geração de modo sistemático de novos estados que são comparados com o estado final pretendido.

i. Profundidade limitada

O algoritmo de pesquisa em profundidade limitada é uma variante do algoritmo de profundidade primeiro. Como este tem um problema com caminhos infinitos, na profundidade limitada fixa-se o nível máximo da procura. Contudo, este algoritmo apresenta um novo problema que é qual o valor máximo a ser usado para o nível.

O script *DepthLimitedSearch* é a implementação deste algoritmo. A partir do script *BreadthFirstSearch* fornecido pelos docentes, fizemos as alterações necessárias:

- uso de uma *stack*;
- definição de uma variável *limit* que vai ser usada na condição que verifica se a profundidade do nó actual é menor que o limite definido.

ii. Aprofundamento progressivo

O algoritmo de aprofundamento progressivo resolve o problema anterior da pesquisa em profundidade limitada. Para tal, chama-se o algoritmo anterior, incrementando o limite do valor máximo.

A implementação do algoritmo encontra-se no script *ProgressiveDepth*. A única alteração efectuada foi na condição de quando a *stack* já não contém nós. Assim, o limite é incrementado, a *stack* e o *hashset* são "limpos" e volta-se a chamar a função *Start()*.

b. Procura informada

Os algoritmos de procura informada utilizam conhecimento específico do problema na escolha do próximo nó a ser expandido, ou seja, consideram características próprias do problema. Para estes algoritmos tivemos de desenvolver heurísticas que estão explicadas no capítulo seguinte.

i. Pesquisa sôfrega

O algoritmo de procura sôfrega consiste na escolha do nó mais promissor de acordo com o valor estimado por h(n). A árvore de procura é ordenada relativamente a h(n), sendo escolhido o nó com o valor mínimo que, hipoteticamente, está mais próximo da solução pretendida.

O script *GreedySearch* implementa este algoritmo utilizando a heurística descrita no enunciado. Utilizam-se duas *lists* de nós, uma função com um algoritmo de ordenação, *insertion sort* e uma variável que vai obter o valor devolvido pela heurística, *h_ node.* Desta forma, sempre que se expande um nó sucessor, a heurística é calculada com o estado deste sucessor. À *list* vai ser adicionado esse nó e, posteriormente, na outra *list* são todos ordenados por ordem crescente em função do valor de h.

Os scripts *GreedySearchdistCrateGoal, GreedySearchdistCratePlayer, GreedySearchdistEuclGoal* e *GreedySearchdistEuclPlayer* implementam as outras heurísticas. A única diferença a nível de código é a chamada da função respectiva a cada heurística para o cálculo de *h_ node.*

ii. A*

O algoritmo A* procura escolher a cada instante o melhor caminho passando pelo nó, utilizando a função f(n), que é definida por: f(n) = g(n) + h(n).

A implementação do A* encontra-se no script *AStar* e é idêntica ao algoritmo anterior. A única diferença é a ordenação que é feita relativamente a f.

Novamente, os scripts *AStardistCrateGoal, AStardistCratePlayer, AStardistEuclGoal* e *AStardistEuclPlayer* correspondem às restantes heurísticas.

HEURÍSTICAS

a. Número de caixas fora de posição

A primeira heurística foi-nos pedida no enunciado: "a estimativa do custo de transitar de um estado *s* até ao estado final é igual ao número de caixas que não estão numa posição destino". Para desenvolver esta heurística foi criada uma função *getRemainingGoals*, na qual se obtém o número de objectivos e, para cada caixa, verifica-se se já está na posição destino. Se estiver, decrementa-se o valor de *remainingGoals*. No final, retorna-o. Assim, para cada estado num determinado nó, verifica quantas caixas faltam colocar na posição objectivo.

b. Distâncias

Os mapas de Sokoban podem ser vistos como um plano que contém elementos que se movem para determinadas posições. Deste modo, achámos que as melhores heurísticas a implementar deveriam estar relacionadas com a distância de uns elementos relativamente a outros.

Ambas as heurísticas são admissíveis, pois nenhuma delas calcula excessivamente a distância entre elementos, procurando sempre a distância mínima. Independentemente da distância euclidiana entre um objecto e outro ser menor do que a sua distância segundo a de Manhattan como o jogador não pode andar na diagonal, acaba por percorrer a mesma distância tanto numa heurística como na outra.

i. Distância de Manhattan

A fórmula $\sum |a_i - b_i|$, soma das diferenças absolutas das coordenadas de dois pontos, dá-nos a distância entre dois determinados pontos. Assim, implementámos duas funções distintas: a primeira soma as distâncias mínimas entre o jogador e as caixas (*distCratePlayer*); por sua vez, a segunda calcula as distâncias mínimas entre as caixas e as posições objectivo (*distCrateGoal*).

As funções que aplicam esta fórmula encontram-se no script *SokobanProblem*. São chamadas nos respectivos scripts. Assim:

- *AStardistCratePlayer e GreedySearchdistCratePlayer – distCratePlayer;*

- *AStardistCrateGoal* e *GreedySearchdistCrateGoal* – *distCrateGoal.*

De notar, que esta heurística deixaria de ser admissível se o jogador pudesse deslocar-se na diagonal.

ii. Distância Euclidiana

Ao contrário da distância de Manhattan, a Euclidiana calcula a distância em "linha recta" entre dois pontos, através da fórmula $\sqrt{\sum(a_i - b_i)^2}$. Novamente, implementámos duas funções com o cálculo das distâncias mínimas descritas em cima.

Assim, as funções que se encontram no script *SokobanProblem*, *distEuclPlayer* e *distEuclGoal* correspondem respectivamente à distância entre o jogador e as caixas e, entre as caixas e as posições objectivo. Novamente os scripts correspondentes:

- *AStardistEuclPlayer e GreedySearchdistEuclPlayer – distCratePlayer;*
- *AStardistEuclGoal* e *GreedySearchdistEuclGoal – distCrateGoal.*

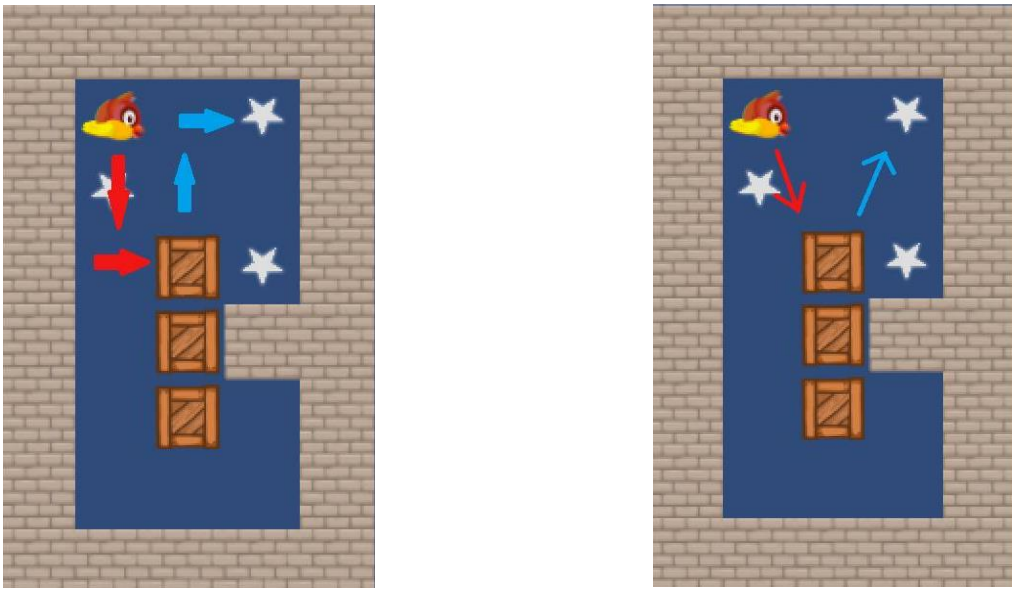

Figura 8 – Diferença entre o cálculo da distância entre dois pontos. À esquerda, distância de Manhattan e à direita, distância euclidiana.

EXPERIMENTAÇÃO

Depois da finalização prática do trabalho, executámos os nossos scripts para diferentes mapas. Com os valores obtidos, preenchemos as tabelas com os valores dos nós visitados, expandidos e o tamanho do caminho percorrido. De seguida, para cada mapa apresentamos uma tabela e as respectivas conclusões.

a. Mapa 1

			Visitados	Expandidos	Caminho
Procura Cega	Breadth First		22	58	5
	Depth Limited		17	47	5
	Progressive Depth		41	116	5
Procura Informada	A*	*getRemainingGoals*	21	55	5
		distCrateGoal	21	55	5
		distCratePlayer	9	27	5
		distEuclGoal	21	55	5
		distEuclPlayer	12	36	5
	Pesquisa Sôfrega	*getRemainingGoals*	19	50	5
		distCrateGoal	19	50	5
		distCratePlayer	6	18	5
		distEuclGoal	19	50	5
		distEuclPlayer	6	18	5

Este mapa tem menor complexidade relativamente aos outros, pois tem apenas uma caixa, uma posição objectivo e não há obstáculos entre o jogador e a caixa. Assim, é normal que os nós visitados e expandidos sejam reduzidos. Dada a esta baixa dificuldade do mapa, em termos de complexidade temporal, obtivemos um valor muito elevado por aprofundamento progressivo em comparação com os restantes algoritmos. Com as heurísticas que calculam as distâncias entre a caixa e a posição objectivo e para a largura primeiro obtivemos resultados semelhantes.

A pesquisa sôfrega com as heurísticas *distCratePlayer*, *distEuclPlayer* e o A* *distCratePlayer* apresentam valores muito mais reduzidos, isto é, têm uma melhor performance temporal. A profundidade limitada é o valor mais baixo da procura cega, mas ainda assim, não obteve a melhor performance comparativamente com os anteriores.

Todos os algoritmos testados por nós são melhores que o aprofundamento progressivo. Comparativamente a largura primeiro teve melhor performance, mas obteve também piores tempos que qualquer um dos algoritmos desenvolvidos.

Relativamente à complexidade espacial, por comparação de resultados em todos os algoritmos de pesquisa, obtivemos praticamente as mesmas conclusões anteriores em termos de performance por comparação entre algoritmos. Neste mapa o caminho efectuado pelo jogador foi igual e com o mesmo tamanho para todos as situações, devido à baixa complexidade do mapa.

b. Mapa 2

			Visitados	Expandidos	Caminho
Procura Cega	Breadth First		386	1 123	8
	Depth Limited		66	201	10
	Progressive Depth		767	2 301	10
Procura Informada	A*	*getRemainingGoals*	187	564	8
		distCrateGoal	46	145	10
		distCratePlayer	97	296	8
		distEuclGoal	54	168	10
		distEuclPlayer	109	334	8
	Pesquisa Sôfrega	*getRemainingGoals*	30	96	8
		distCrateGoal	36	115	10
		distCratePlayer	63	194	8
		distEuclGoal	36	115	10
		distEuclPlayer	227	697	24

Este mapa tem uma complexidade mais elevada que o anterior, pois apesar de não ter mais obstáculos entre o jogador e as caixas, tem mais caixas, ou seja, mais relações jogador-caixa, caixa-objectivo ou posições objectivo por atingir.

Relativamente à complexidade temporal, obtivemos novamente uma pior performance do aprofundamento progressivo, fazendo este o algoritmo menos eficiente. Comparativamente com o mapa 1, onde o A* *distCrateGoal* e a pesquisa sôfrega *distCrateGoal* tinham uma performance semelhante à largura primeiro, obtivemos para o mapa 2 valores muito inferiores de nós visitados, ou seja, com um melhor tempo que a largura primeiro. Também comparativamente ao mapa anterior, conseguimos um algoritmo com melhor performance que a largura primeiro: a pesquisa sôfrega

distEuclPlayer, no entanto, com uma das piores performances de todos os algoritmos testados (terceiro pior desempenho). Além disto, verificámos ainda que para este mapa, os algoritmos que obtiveram melhores performances a nível temporal foram a pesquisa sôfrega com as heurísticas *getRemainingGoals* (o melhor), *distEuclGoal* e *distCrateGoal*. Estes resultados deveram-se às especificidades do mapa, pois as distâncias entre as caixas e as posições objectivo eram inferiores comparativamente com as do mapa 1, onde as menores distâncias eram as do jogador às caixas. Isto levou a que os algoritmos com as heurísticas que utilizam a distância entre o jogador e as caixas, tenham tido tempos inferiores no primeiro mapa. Para além destes, temos ainda o A* *distCrateGoal*, A* *distEuclGoal* e a pesquisa sôfrega *distCratePlayer* também com melhor desempenho temporal que a profundidade limitada, ainda que a sôfrega *distCratePlayer* seja só por uma melhoria de três nós visitados.

Relativamente à complexidade espacial obtivemos resultados que nos levaram a tirar conclusões semelhantes às de complexidade temporal, por se encontrarem proporcionais em relação aos resultados da complexidade temporal. Excepto a pesquisa sôfrega *distEuclPlayer*, com um caminho de vinte e quatro passos, os restantes apresentaram um caminho semelhante com valores entre os oito e dez passos.

c. Mapa 3

			Visitados	Expandidos	Caminho
Procura Cega	Breadth First		134 251	382 032	29
	Depth Limited		2 274	6 260	35
	Progressive Depth		105 034	281 586	35
Procura Informada	A*	*getRemainingGoals*	126 745	362 926	29
		distCrateGoal	60 807	169 622	31
		distCratePlayer	13 918	38 640	29
		distEuclGoal	61 047	170 252	31
		distEuclPlayer	17 493	48 903	29
	Pesquisa Sôfrega	*getRemainingGoals*	3 506	9 568	37
		distCrateGoal	60 467	168 733	31
		distCratePlayer	15 730	44 459	33
		distEuclGoal	60 467	168 733	31
		distEuclPlayer	6 828	18 995	43

Ao contrário dos anteriores, o algoritmo de procura cega aprofundamento progressivo é mais eficiente que o de largura primeiro. Isto acontece pois a especificidade

deste mapa é superior aos anteriores, ou seja, tem mais caixas e posições objectivo, sendo que, inicialmente, as próprias caixas são obstáculos para o jogador. Conseguimos concluir em termos de complexidade temporal, que todos os algoritmos de pesquisa são melhores que o largura primeiro e todos, excepto o A* *getRemainingGoals*, são melhores que o aprofundamento progressivo, sendo que o que obteve melhor performance temporal foi o profundidade primeiro, seguido da pesquisa sôfrega com as heurísticas *getRemainingGoals* e *distEuclPlayer*.

Relativamente à complexidade espacial, também concluímos que o algoritmo que obteve uma pior performance foi o mesmo que obteve uma pior performance temporal, podendo-se dizer o mesmo para o que obteve uma melhor performance. O melhor caminho é efectuado com vinte e nove movimentações (largura primeiro, A* *getRemainingGoals,* A* *distCratePlayer,* A* *distEuclPlayer)*, mas a maioria dos nossos caminhos tiveram um custo entre trinta e um e trinte e cinco, exceptuando, a pesquisa sôfrega *getRemainingGoals* com trinta e sete e *distEuclPlayer*, pois como este algoritmo analisa qual o caminho seguinte com menor custo de desempenho e não o caminho total com o menor custo.

d. Mapa 4

Mesmo executando este map com qualquer um dos algoritmos e com valores elevados de *steps per frame*, não conseguimos obter qualquer resultado, pois os tempos de execução eram tão elevados que não conseguimos chegar a uma solução dentro de tempos aceitáveis.

e. Mapa 5

			Visitados	Expandidos	Caminho
Procura Cega	Breadth First		3 861	10 082	26
	Depth Limited		1 966	5 354	34
	Progressive Depth		19 639	53 178	34
Procura Informada	A*	*getRemainingGoals*	3 333	8 724	26
		distCrateGoal	1 289	3 348	28
		distCratePlayer	1 315	3 491	26
		distEuclGoal	1 661	4 310	28
		distEuclPlayer	1 690	4 492	26
	Pesquisa Sôfrega	*getRemainingGoals*	558	1 446	26
		distCrateGoal	520	1 333	28
		distCratePlayer	460	1 218	28

		distEuclGoal	520	1 333	28
		distEuclPlayer	328	870	28

O quinto mapa tem uma complexidade reduzida, mas superior ao do primeiro, pois tem mais caixas, sendo assim necessário realizar mais cálculos nos diferentes algoritmos. Em termos de complexidade temporal, ou seja, o número de nós visitados, todos as heurísticas da pesquisa sôfrega são melhores que todos os algoritmos de procura cega, nunca ultrapassando sequer os mil nós visitados.

O algoritmo com melhor desempenho foi a pesquisa sôfrega *distEuclPlayer* e o pior de todos foi o aprofundamento progressivo. Dentro dos algoritmos de procura informada, os piores desempenhos foram obtidos com a heurística *getRemainingGoals*, sendo todos melhores que a profundidade limitada, excepto o A* *getRemainingGoals*.
A nível de complexidade espacial, concluímos que o algoritmo com pior desempenho foi mais uma vez o aprofundamento progressivo e o melhor, tal como o é em termos temporais, foi a pesquisa sôfrega *distEuclPlayer*, sendo o último que não passou os mil nós expandidos. O melhor caminho é com vinte e seis passos adquirido em largura primeiro, A* *getRemainingGoals*, A* *distCratePlayer* e A* *distEuclPlayer* e sôfrega *getRemainingGoals*. Os piores resultados foram em profundidade limitada e aprofundamento progressivo, com trinta e quatro movimentos, sendo que os restantes algoritmos executaram o mapa com vinte e oito passos.

f. Mapa 6

			Visitados	Expandidos	Caminho
Procura Cega	Breadth First		150 766	392 447	33
	Depth Limited		16 660	43 606	41
	Progressive Depth		206 784	548 281	41
Procura Informada	A*	*getRemainingGoals*	97 021	252 739	33
		distCrateGoal	63 503	162 838	39
		distCratePlayer	20 590	54 666	33
		distEuclGoal	65 478	167 883	39
		distEuclPlayer	24 441	64 365	33
	Pesquisa Sôfrega	*getRemainingGoals*	5 925	14 879	51
		distCrateGoal	36 707	94 251	39
		distCratePlayer	29 480	79 203	61
		distEuclGoal	36 707	94 251	39
		distEuclPlayer	21 758	58 670	53

No mapa 6, o algoritmo com pior tempo foi o aprofundamento progressivo, seguido do de largura primeiro. Todos os nossos valores dos algoritmos que desenvolvemos são melhores que os de largura, pois nem sequer chegam aos cem mil nós visitados, enquanto que esse visita mais de cento e cinquenta mil nós. O que obteve melhor performance temporal foi a sôfrega *getRemainingGoals* com valores muito inferiores aos restantes, não chegando aos dez mil nós visitados, seguido da profundidade limitada e da pesquisa sôfrega *distEuclPlayer*.

Além disto, verificámos também que a pesquisa sôfrega com as heurísticas *distEuclGoal* e *distCrateGoal* obtiveram os mesmos resultados para os nós visitados e expandidos, tendo também realizado o mesmo caminho, pois como o jogador não admite movimentos na diagonal acabam por realizar o mesmo caminho, independentemente da distância euclidiana ser calculada na diagonal e na de Manhattan não o ser.

Em termos de complexidade espacial, o pior foi o aprofundamento progressivo, com quase quinhentos e cinquenta mil nós expandidos e o melhor foi a pesquisa sôfrega *getRemainingGoals* com apenas quinze mil nós expandidos, os restantes em termos de hierarquia de performances foram praticamente equivalentes. Para este mapa, obtivemos caminhos mais díspares entre os vários algoritmos de procura, sendo que o melhor caminho obteve trinta passos na largura primeiro, no A* com as heurísticas *getRemainingGoals*, *distCratePlayer* e no *distEuclPlayer*. O pior foi a pesquisa sôfrega *distCratePlayer*, com sessenta e um movimentos, seguido da heurística *distEuclPlayer* com cinquenta e três e da *getRemainingGoals* com cinquenta e um. Os outros valores obtidos foram trinta e nove ou quarenta e um movimentos.

Com esta análise concluímos que, embora um algoritmo seja mais eficiente a nível temporal e espacial, poderá não ser o que produz o caminho mais económico. No mapa 1 é indiferente qualquer um que se use por razões já referidas. Já no mapa 2 concluímos que o melhor tanto a nível temporal/espacial como a nível económico é o a pesquisa sôfrega *getRemainingGoals*. Quanto ao terceiro mapa melhor a nível temporal/espacial é o algoritmo de profundidade limitada mas o mais económico é A* *distCratePlayer*. No Mapa 5 verificámos que o de menor complexidade espacial e temporal foi a pesquisa sôfrega *distEuclPlayer*, mas a nível de custo é já é a heurística *getRemainingGoals*. Por último, no mapa 6 o de menor complexidade temporal e espacial é a sôfrega *getRemainingGoal*, mas o que produz melhor caminho com melhor complexidade temporal e espacial é o A* *distCratePlayer*.

Assim, conseguimos concluir que todos os algoritmos de procura com as heurísticas implementadas por nós, são completos pois todos encontraram uma solução.

Quanto a serem discriminadores, podemos verificar que dentro dos algoritmos de procura cega, o que encontra o caminho mais económico é o de largura primeiro, logo é discriminador, pois em todos os mapas encontrou o caminho com menor custo. Dentro do algoritmo A* as heurísticas *getRemainingGoals*, *distCratePlayer* e *distEuclPlayer* tornam-no discriminador, pois são os que fornecem os caminhos com o custo menor. Na pesquisa sôfrega, dado que alguns dos algoritmos encontram o caminho mais económico nuns mapas e não noutros, concluímos que não é discriminador.

ANÁLISE DOS ALGORITMOS E COMPLEXIDADES

O aprofundamento progressivo vai analisar o mesmo nós várias vezes, fazendo com que haja uma sobrecarga temporal, dando uma complexidade elevada que se pode verificar nos resultados obtidos em quase todos os mapas, dado que este algoritmo tinha sempre o tempo maior. Conseguimos ver ainda que, dado que o espaço mínimo depende da profundidade da solução, nos mapas com uma maior variedade de caminhos possíveis os seus valores são tendencialmente maiores.

Em relação à profundidade limitada, dado termos definido um limite, o algoritmo sabe que há um máximo ao qual pode chegar, nunca correndo o risco de chegar a um ponto infinito de pesquisa. Como tal, a pesquisa que este algoritmo vai executar com tempos bastante mais reduzidos em comparação aos outros algoritmos, o que também se verificou com os nossos resultados. A pesquisa sôfrega e o A* têm a mesma complexidade, mas o primeiro algoritmo como apenas avalia o próximo caminho com menos custo e não o custo mais pequeno na sua totalidade, acaba por ter tempos inferiores mas com caminhos mais caros, o que se pode verificar ao comparar os nossos resultados.

		Completo	Discriminador	Tempo Mínimo	Espaço Mínimo
Procura Cega	Breadth First	Sim	Sim	O(b^p)	O(b^p)
	Depth Limited	Sim	Não	O(b^l)	O(b*l)
	Progressive Depth	Sim	Não	O(b^p)	O(b*p)
Procura Informada	A*	Sim	Sim	O(b^p)	O(b^p)
	Greedy	Sim	Não	O(b^p)	O(b^p)

b = factor de ramificação (no nosso caso, 4)

p = profundidade da solução

l = limite definido

CONCLUSÃO

Com a realização do segundo trabalho prático, conseguimos consolidar os conhecimentos gerais sobre agentes de procura e seus algoritmos.

Relativamente, à parte da modelação não encontrámos dificuldades em definir o Sokoban como um problema de pesquisa. A implementação dos diferentes algoritmos de procura já foi um pouco mais complicada, pois tivemos alguns erros a nível de código que só foi possível descobrir aquando da experimentação, pois estávamos a obter resultados incomuns.

Quanto à experimentação, não tivemos muitos problemas, a não ser com o quarto mapa fornecido, que não conseguimos executar em tempo útil nem com as heurísticas que nós desenvolvemos. Conseguimos contornar este facto, elaborando mais dois mapas, que foram essenciais na análise empírica e comparativa dos diferentes algoritmos.

Por fim, notámos que a utilização do Unity já é bastante mais fácil do que no primeiro trabalho, sendo uma ferramenta bastante prática e intuitiva.