INSTITUTO SUPERIOR TÉCNICO

Procura e Planeamento

Campus Alameda

IST @ 2019/2020

5 de Maio de 2020

1. Introdução

Este projecto tem dois objectivos:

(1) Desenvolver um programa em ANSI Common Lisp que resolva diferentes puzzles do jogo Same Game. Devem ser utilizadas diferentes estratégias de procura estudadas na cadeira, e desenvolvidas heurísticas que permitam resolver este tipo de problemas da forma mais eficaz possível;

(2) Produzir um estudo que avalie as alternativas implementadas tanto do ponto de vista quantitativo como qualitativo.

2. Descrição do jogo Same Game

Este é um jogo para um único agente, inventado no Japão em 1985 por Kuniaki Moribe. Usa um tabuleiro rectangular de dimensões arbitrárias que, inicialmente, se encontra preenchido por completo com peças. Cada peça pode ter uma de várias cores diferentes. O objectivo do jogo é retirar tantas peças do tabuleiro quanto possível,

maximizando os pontos obtidos. Na figura 1 mostra-se uma representação do tabuleiro no início do jogo[1].

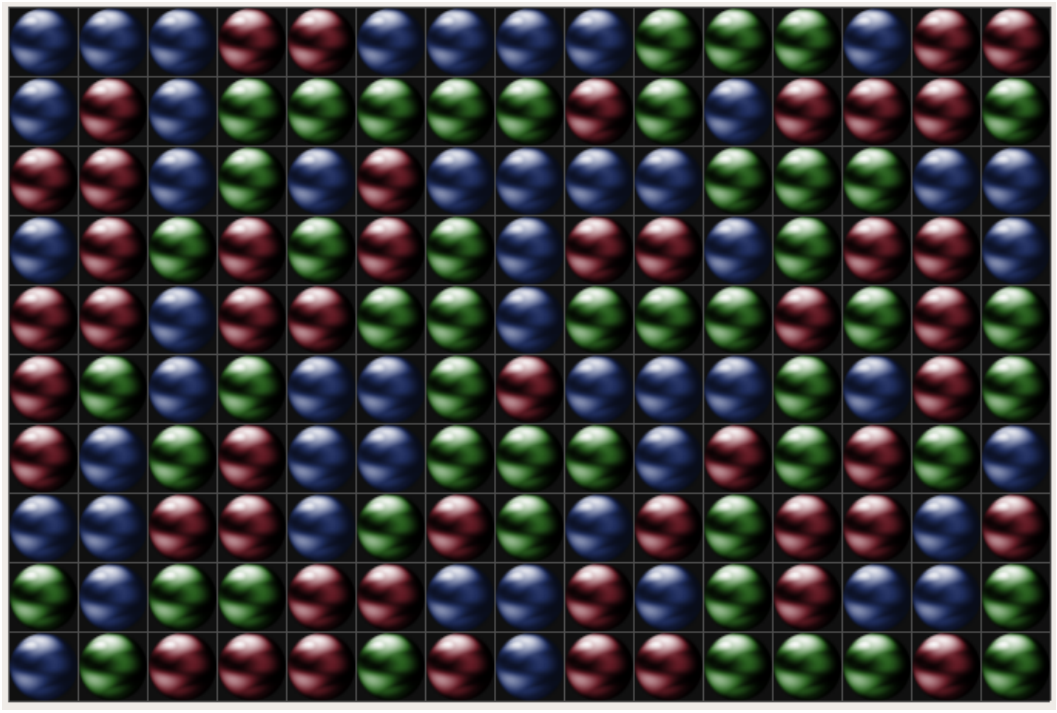
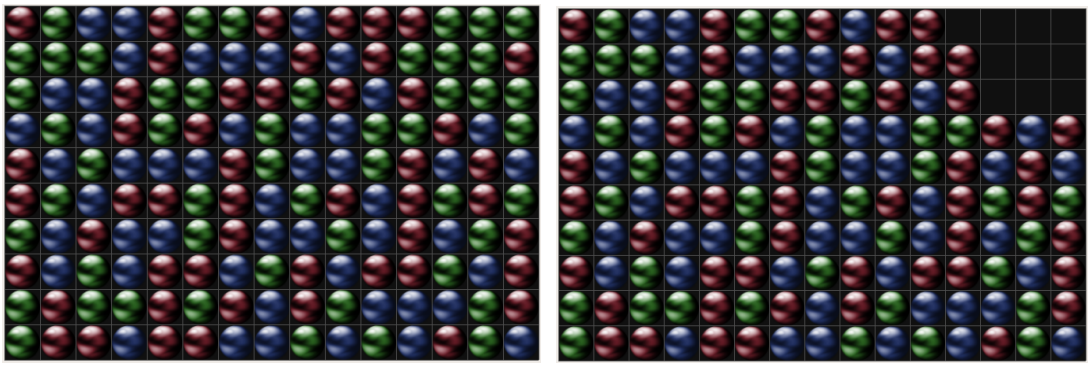


Figura 1: Exemplo de um puzzle do jogo Same Game.

A cada momento, podem ser retirados do tabuleiro conjuntos de pelo menos duas peças adjacentes da mesma cor. Consideram-se adjacentes, peças ligadas na horizontal ou vertical, mas não na diagonal. Os conjuntos de peças retirados incluem todas as peças de uma dada cor adjacentes entre si. Não é possível optar por retirar apenas parte de um grupo de peças adjacentes (figura 2). Os pontos atribuídos por cada jogada dependem do número de peças retiradas em simultâneo, de acordo com a fórmula $(n - 2)^2$, sendo *n* o número de peças removidas.

[1] As várias imagens mostradas ao longo deste enunciado foram retiradas do jogo Same GNOME, Copyright @ 2008 Callum McKenzie.

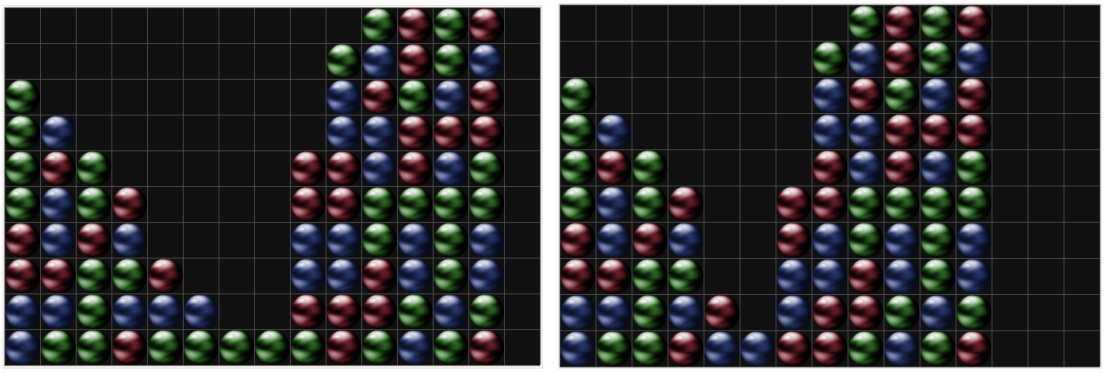


(a) Tabuleiro inicial (b) Tabuleiro após jogada

Figura 2: Exemplo de uma jogada.

Sempre que uma peça é removida, todas as que estão acima dela "caem" de modo a ocupar o espaço por ela deixado. Se, porventura, ao remover uma peça ficarem uma ou mais colunas completamente vazias, então todas as peças à direita dessa coluna movem-se para a esquerda de modo a que não existam zonas desconexas no tabuleiro, nem zonas vazias no lado esquerdo do mesmo (figura 3).

O jogo termina quando o tabuleiro estiver vazio ou quando não for possível retirar mais peças.



(a) Tabuleiro inicial (b) Tabuleiro após jogada

Figura 3: Exemplo de uma jogada.

3. Resolução do problema

O agente implementado deverá ser capaz de lidar com tabuleiros de quaisquer dimensões e com qualquer número de cores distintas.

O objectivo é maximizar a pontuação final que se calcula com a fórmula apresentada na secção anterior.

O Same Game é um jogo cuja resolução automática não é fácil, só sendo possível resolver puzzles de dificuldade não trivial se se escolher uma formulação adequada.

Dado o elevado factor de ramificação do problema em puzzles de maior dimensão, devem ser estudadas e implementadas estratégias de limitação de sucessores (estratégia de corte de sucessores) que permitam aumentar a capacidade do jogador automático.

Como o tipo de solução a desenvolver pode depender da complexidade dos problemas a resolver, seguse-se um conjunto de problemas, com diferentes características, que podem ser usados para testar as suas estratégias de procura:

- Tabuleiro de 10x4 com 3 cores

  (S5 ;; Tabuleiro de 4x10 com 3 cores

  ((2 1 3 2 3 3 2 3 3 3) (1 3 2 2 1 3 3 2 2 2) (1 3 1 3 2 2 2 1 2 1) (1 3 3 3 1 3 1 1 1 3)))

- Tabuleiro de 10x4 com 5 cores

  (S10 ;; Tabuleiro de 4x10 com 5 cores

  ((4 3 3 1 2 5 1 2 1 5) (2 4 4 4 1 5 2 4 1 2) (5 2 4 1 4 5 1 2 5 4) (1 3 1 4 2 5 2 5 4 5)))

- Tabuleiro de 15x10 com 3 cores

  (S15 ;; Tabuleiro de 15x10 com 3 cores

  ((3 3 3 2 1 2 3 1 3 1) (1 1 2 3 3 1 1 1 3 1) (3 3 1 2 1 1 3 2 1 1) (3 3 2 3 3 1 3 3 2 2)

  (3 2 2 2 3 3 2 1 2 2) (3 1 2 2 2 2 1 2 1 3) (2 3 2 1 2 1 1 2 2 1) (2 2 3 1 1 1 3 2 1 3)

  (1 3 3 1 1 2 3 1 3 1) (2 1 2 2 1 3 1 1 2 3) (2 1 1 3 3 3 1 2 3 1) (1 2 1 1 3 2 2 1 2 2)

  (2 1 3 2 1 2 1 3 2 3) (1 2 1 3 1 2 2 3 2 3) (3 3 1 2 3 1 1 2 3 1)))

- Tabuleiro de 15x10 com 5 cores

  (S20 ;; Tabuleiro de 15x10 com 5 cores

  ((5 1 1 1 2 1 4 2 1 2) (5 5 5 4 1 2 2 1 4 5) (5 5 3 5 5 3 1 5 4 3) (3 3 3 2 4 3 1 3 5 1)

  (5 3 4 2 2 2 2 1 3 1) (1 1 5 3 1 1 2 5 5 5) (4 2 5 1 4 5 4 1 1 1) (5 3 5 3 3 3 3 4 2 2)

  (2 3 3 2 5 4 3 4 4 4) (3 5 5 2 2 5 2 2 4 2) (1 4 2 3 2 4 5 5 4 2) (4 1 3 2 4 3 4 4 3 1)

  (3 1 3 4 4 1 5 1 5 4) (1 3 1 5 2 4 4 3 3 2) (4 2 4 2 2 5 3 1 2 1)))

4. Implementação

Parte da avaliação do trabalho desenvolvido vai decorrer automaticamente, pelo que é essencial que a especificação da interface seja seguida rigorosamente.

O conjunto de funções que implementa o jogador deve ser definido num único ficheiro, sendo que este deve poder ser compilado sem erros nem avisos. Uma dos avisos produzidos pelos compiladores é a falta de uma declaração de package num ficheiro de código, de forma a evitar este aviso deve ser incluída a seguinte forma Lisp:

```
(in-package :user)
```

Esta forma deve ser a primeira forma do ficheiro.

Refira-se ainda que os testes a realizar automaticamente impõem alguns limites espaciais e temporais considerados razoáveis:

(1) um limite à heap de 256Mbytes; e

(2) um tempo máximo por problema de 5 minutos num Intel Core 2 Duo a 2,8GHz, ou seja a eficiência temporal e espacial das soluções apresentadas é relevante, tal como já tinha sido afirmado anteriormente. A função get-internal-run-time e a constante INTERNAL-TIME-UNITS-PER-SECOND podem ser utilizadas para ajudar a controlar o tempo de execução.

4.1. Representação externa de um jogo

A representação externa de um jogo de Same Game é uma lista de listas em que as sub-listas correspondem às linhas do tabuleiro do jogo. Uma sub-lista tem uma representação do conteúdo da linha, (1) se a posição está desocupada o conteúdo é NIL e (2) se a posição está ocupada por uma peça o conteúdo é um inteiro maior ou igual a um correspondente à cor da peça.

Para o exemplo da figura 4 a representação externa seria:

```
((1 2 2 3 3) (2 2 2 1 3) (1 2 2 2 2) (1 1 1 1 1))
```

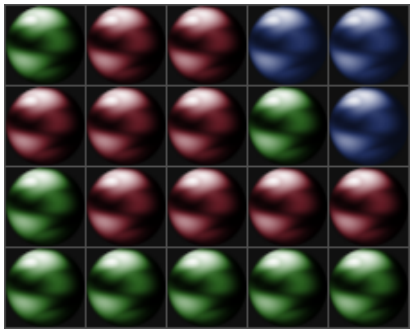


Figura 4: Exemplo de um puzzle de Same Game.

4.2. Interface

Deve ser implementada a função *resolve-same-game*, que permite resolver puzzles do jogo Same Game.

Esta função recebe como argumentos: um problema na sua representação externa definida anteriormente e a estratégia de procura a usar.

A função deve devolver uma lista com as posições em que cada jogada foi sucessivamente feita (sendo o primeiro elemento da lista a posição da primeira jogada). Essa posição indica um dos elementos do grupo que foi retirado em cada jogada (como são retiradas todas as peças adjacentes da mesma cor, é possível conhecer todo o grupo a partir da posição de um qualquer dos seus elementos). Cada posição não é mais do que uma lista com dois elementos que representam, respectivamente, a linha e a coluna em causa. Assume-se que as posições estão numeradas da esquerda para a direita e de cima para baixo, a partir de zero. Caso nenhuma jogada seja possível, a função deve devolver NIL.

As estratégias a suportar são:

```
"melhor.abordagem"
"a*.melhor.heuristica"
"a*.melhor.heuristica.alternativa"
"sondagem.iterativa"
"abordagem.alternativa"
```

Por exemplo, para o jogo de Same Game apresentado na figura 4, a invocação a esta função tomaria a forma

```
(resolve-same-game '((1 2 2 3 3) (2 2 2 1 3)
                     (1 2 2 2 2) (1 1 1 1 1))
                   "melhor.abordagem")
```

e retornaria, por exemplo, a seguinte lista:

```
((1 0) (1 0) (3 0))
```

4.3. Código Lisp fornecido

Para a realização deste projecto devem ser utilizados os algoritmos de procura, desenvolvidos em ANSI Common Lisp, disponíveis no *site* da cadeira. O código fornecido encontra-se no ficheiro '*procura.lisp*'. Este contém a implementação de vários algoritmos de procura. O funcionamento dos mesmos encontra-se descrito em '*procura.txt*'. O mais importante é compreender para que servem, não perceber exactamente como é que as funções estão definidas. Este ficheiro não deve ser alterado, se houver necessidade de alterar definições incluídas neste ficheiro estas devem ser redefiidas no ficheiro de código desenvolvido que contém a implementação realizada pelo grupo.

5. Estudo a desenvolver

A abordagem a desenvolver deve modelar o problema como uma procura num espaço de estados usando uma abordagem incremental e uma perspectiva de optimização. Ou seja deve devolver a melhor solução possível de acordo com o definido anteriormente.

- Devem ser testadas todas as técnicas de procura fornecidas no ficheiro "*procura.lisp*". Note-se que é necessário alterar as estratégias de forma a que terminem dentro do tempo limite devolvendo a melhor solução encontrada até esse momento.
- É preferível retornar ao fim do tempo limite um estado ainda com jogadas possíveis e, digamos, com 100 pontos, do que um estado já sem jogadas, um estado terminal, com 10 pontos. Por outras palavras, deve ser retornado o estado com maior pontuação - não tem que ser um estado terminal.
- Deve ser implementada e testada a estratégia de sondagem iterativa.

- Deve ser implementada pelo menos uma abordagem alternativa diferente das listadas acima.
- Adicionalmente, novas estratégias que facilitem a resolução dos problemas devem ser desenvolvidas. Estas podem incluir variações aos algoritmos básicos de procura, novos algoritmos de procura, estratégias híbridas, macro-operadores, sub-objectivos, etc. A análise do problema e dos primeiros resultados obtidos deverá fornecer indicadores quanto a abordagens válidas.

Procuras informadas:

- Para as técnicas de procura informadas, são necessárias heurísticas. Devem ser apresentadas pelo menos duas heurísticas de desempenho relevante. Essas heurísticas devem, tanto quanto possível, basear-se em ideias distintas (heurísticas que variam apenas em constantes, por exemplo, são consideradas a mesma heurística).

O desempenho das várias estratégias de procura utilizadas, bem como das heurísticas e das estratégias de corte desenvolvidas deve ser discutido no relatório, fazendo uma comparação dos resultados obtidos para alguns dos problemas usados para teste. Toda esta avaliação deve ser baseada em resultados quantitativos nomeadamente: custo das soluções, tempo da procura, nós gerados, nós expandidos, factor médio de ramificação, profundidade atingida, etc.

A discussão deve limitar-se a problemas que salientem características ou limitações relevantes do trabalho desenvolvido. Os alunos são encorajados a criar novos problemas, para além dos fornecidos como exemplo, que sejam adequados para ilustrar aspectos interessantes da discussão.

Todo o processo de decisão deve ser documentado no relatório a elaborar.

O relatório do projecto deve incluir ainda:

- A descrição da(s) modelação(ões) do problema e as razões subjacentes às decisões tomadas;
- A descrição das estruturas de dados desenvolvidas e as razões subjacentes ao seu desenvolvimento;
- A descrição das heurísticas desenvolvidas e as razões subjacentes ao seu desenvolvimento;

- A descrição das estratégias de corte implementadas e as razões subjacentes ao seu desenvolvimento;
- As opções tomadas durante o desenvolvimento do projecto;
- Uma discussão sobre a forma de conseguir obter melhores resultados, indicando genericamente o tipo de heurísticas/estratégias a desenvolver para isso ou alterações à modelação.

Ainda quanto ao relatório, convém salientar que a legibilidade do mesmo (para a qual contribuem não só a organização dos capítulos/secções como a correcção ortográfica a construção das frases e a fluidez do discurso) é de importância capital para a correcta compreensão do seu conteúdo. Devem evitar-se frases complexas (a utilização da voz passiva, de duplas negativas, etc.). Todas as figuras e gráficos apresentados devem ser legíveis e estar adequadamente legendados.

6. Critérios de avaliação

Os factores mais importantes para a avaliação do projecto são (não necessariamente por esta ordem):

- Execução correcta e elegância do programa;
- Qualidade da(s) modelação(ões) do problema;
- Qualidade das heurísticas e estratégias desenvolvidas;
- Eficiência das estruturas de dados escolhidas para a representação do problema;
- Qualidade do estudo desenvolvido (inclui a descrição do trabalho desenvolvido, os resultados obtidos e conclusões);
- Apreciação global.

7. Inscrições e entrega do projecto

Os elementos de cada grupo, que deverão ser no máximo 2, devem inscrever-se no Fénix.

A entrega será dividida em duas fases, a primeira relativa ao código desenvolvido e a segunda relativa ao relatório.

A entrega do código desenvolvido deve ser feita até às 23:59 do dia 27 de Maio de 2020, da seguinte forma:

- o ficheiro com o programa deve ser entregue de forma electrónica usando o sistema Fénix. O nome do ficheiro Lisp a entregar deve ser "GXXX.lisp", em que "XXX" é o número do grupo, por exemplo "G007.lisp";

A entrega do relatório deve ser feita até às 23h59 do dia 29 de Maio de 2020 da seguinte forma:

- o ficheiro com o relatório em formato electrónico (ficheiro em formato “pdf” ou alternativamente em formato “word") deve ser entregue de forma electrónica usando o sistema Fénix. O nome do ficheiro a entregar deve ser "GXXX.pdf" ou "GXXX.doc", em que "XXX" é o número do grupo, por exemplo "G007.pdf";

Todo o material entregue pelos alunos (documentação e ficheiro com o código) deve conter a identificação do grupo (número do grupo, fornecido durante a inscrição do grupo) e o número e nome de cada elemento do grupo (ver folha anexa).

Atenção: Não são aceites entregas fora do prazo!

8. Novidades do projecto

No caso de haver novidades relativas ao projecto, estas serão afixadas na página da cadeira pelo que esta página deve ser visitada diariamente.

INSTITUTO SUPERIOR TÉCNICO

Procura e Planeamento

Campus Alameda

Projecto (2019/2020)

Número do grupo (obtido a partir do sistema Fénix):

Nome: _____ Número: _____

Nome: _____ Número: _____

Classificação: _____ (a preencher pelo docente)

Soma das horas gastas exclusivamente para fazer este trabalho: _____

Limite para entrega do código: dia 27/05/2020, às 23:59 horas

Limite para entrega do relatório: dia 29/05/2020, às 23:59 horas.