



TÉCNICO LISBOA

# Inteligência Artificial

1.º Semestre 2014/2015

## Enunciado Projecto – *Fill-a-Pix*

### 1 Introdução

Neste projecto pretende-se implementar um algoritmo que seja capaz de resolver qualquer instância de um puzzle *Fill-a-Pix*. Um puzzle *Fill-a-Pix* é uma variante particular de um *Nonograma*<sup>1</sup>, puzzles lógicos japoneses onde se deve colorir ou deixar de colorir uma casa de acordo com um conjunto de regras lógicas. Ao colorir todas as casas de um *Nonograma*, é descoberta uma imagem específica.

No caso particular de um puzzle *Fill-a-Pix*, quando uma casa especifica um número (entre 0 e 9), esse número representa o número de casas pretas no quadrado 3x3 formado pela própria casa e pelas 8 casas circundantes (caso existam). A figura seguinte mostra vários exemplos deste tipo de restrições.

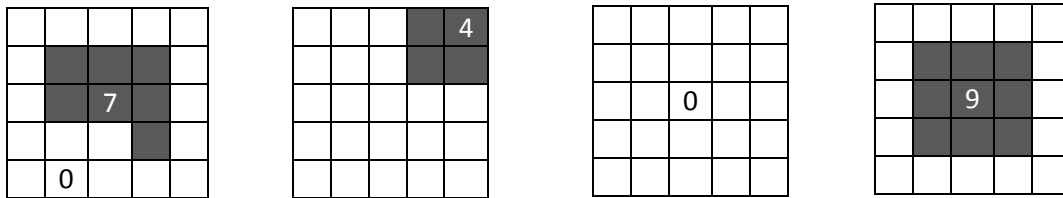


Figura 1 – exemplos de restrições ao número de casas pretas circundantes.

Resolver o puzzle consiste em assinalar todas as casas que devem ser preenchidas a preto, e todas as que devem ficar em branco, de modo a respeitar todas as restrições especificadas no puzzle.

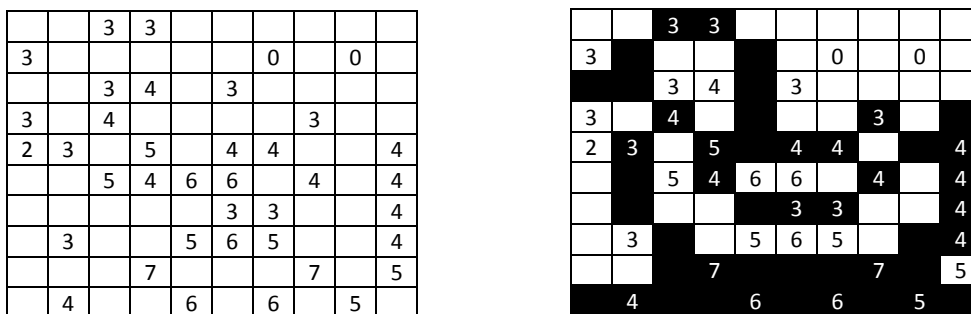


Figura 2- Exemplo de puzzle *Fill-a-Pix*<sup>2</sup> e correspondente solução.

<sup>1</sup> <http://en.wikipedia.org/wiki/Nonogram>

<sup>2</sup> Retirado de <http://www.conceptispuzzles.com/index.aspx?uri=puzzle/fill-a-pix>

## 2 Trabalho a realizar

O objectivo do projecto é escrever um programa em Lisp que resolva automaticamente puzzles *Fill-a-Pix* modelando-os como um Problema de Satisfação de Restrições<sup>3</sup> (PSR).

Para tal deverão realizar várias tarefas que vão desde a implementação dos tipos de dados usados na representação do PSR e das restrições, até à implementação da inteligência artificial para resolver os puzzles usando uma ou mais variantes de algoritmos que trabalham sobre PSRs. Uma vez implementado o algoritmo, os alunos deverão fazer um estudo/análise comparativa das várias versões do algoritmo, bem como das funções heurísticas implementadas.

Também **não é necessário testarem os argumentos recebidos** pelas funções. Podem assumir que os argumentos recebidos por uma função estão sempre correctos.

### 2.1 Tipos Abstractos de Informação

#### 2.1.1 Tipo Restrição

O tipo Restrição é utilizado para representar uma restrição de um PSR. As operações básicas associadas a este tipo são:

- `cria-restricao`: lista de variáveis x predicado → restrição

Este construtor recebe uma lista das variáveis<sup>4</sup> envolvidas na restrição, e uma função que verifica a restrição (i.e. um predicado que recebe um PSR e verifica se o PSR satisfaz a restrição). Retorna uma restrição correspondente.

- `restricao-variaveis`: restrição → lista de variáveis

Este selector, dada uma restrição *r*, retorna uma lista com todas as variáveis envolvidas na restrição. A lista de variáveis retornada tem que ter a mesma ordem que a lista inicialmente recebida quando a restrição é criada.

- `restricao-funcao-validacao`: restrição → predicado

Este selector, dada uma restrição *r*, retorna a função de avaliação usada para verificar se a restrição é satisfeita. O predicado retornado recebe um PSR como argumento e retorna T se o PSR verificar a restrição.

#### 2.1.2 Tipo PSR

O tipo PSR é utilizado para representar um problema de satisfação de restrições, que guarda informação acerca das variáveis, domínios e restrições de um PSR. As operações básicas associadas a este tipo são:

- `cria-psr`: lista variáveis x lista de domínios x lista de restrições → PSR

Este construtor recebe uma lista de variáveis, uma lista de domínios<sup>5</sup> com o mesmo tamanho da lista de variáveis, e uma lista de restrições. Devolve um PSR com as variáveis, domínios e restrições

---

<sup>3</sup> Constraint Satisfaction Problem (CSP) na terminologia anglo-saxónica

<sup>4</sup> Uma variável vai ser representada como uma string.

<sup>5</sup> Para simplificar, um domínio vai ser representado simplesmente como uma lista de valores possíveis. Por exemplo, a lista (0 1) representa um domínio com 2 valores possíveis para uma variável: 0 e 1.

recebidas. A associação entre variáveis e domínios é determinada pela ordem em que os elementos aparecem na lista: o 1.º domínio corresponde à 1.ª variável, e assim sucessivamente.

- `psr-atribuicoes`: PSR → lista atribuições

Este selector recebe um PSR, e retorna uma lista com todas as atribuições<sup>6</sup> do PSR. A ordem das atribuições retornadas não é relevante.

- `psr-variaveis-todas`: PSR → lista variáveis

Este selector recebe um PSR e retorna uma lista com todas as variáveis do PSR, independentemente de estarem ou não atribuídas. A ordem das variáveis retornadas tem que ser a mesma das variáveis recebidas na criação do psr.

- `psr-variaveis-nao-atribuidas`: PSR → lista de variáveis

Este selector recebe um PSR e retorna uma lista com as variáveis do PSR que ainda não foram atribuídas. A ordem das variáveis retornadas tem que ser consistente<sup>7</sup> com a ordem das variáveis recebidas na criação do psr.

- `psr-variavel-valor`: PSR x variável → objecto

Este selector recebe um PSR e uma variável. Se a variável tiver um valor atribuído, é retornado o valor correspondente. Se a variável não estiver atribuída, é retornado NIL.

- `psr-variavel-dominio`: PSR x variável → domínio

Este selector recebe um PSR e uma variável e retorna o domínio correspondente a essa variável. Os valores do domínio retornado têm que ter a mesma ordem que a recebida na criação do psr.

- `psr-variavel-restricoes`: PSR x variável → lista restrições

Este selector recebe um PSR e uma variável, e retorna uma lista com todas as restrições aplicáveis a essa variável. A ordem das restrições retornadas tem que ser consistente com a ordem da lista de restrições recebida na criação do psr.

- `psr-adiciona-atribuicao!`: PSR x variável x valor → {}

Este modificador recebe um PSR, uma variável e um valor, e **altera** o PSR recebido, atribuindo o valor à variável. Se a variável já tinha sido atribuída, o novo valor substitui o valor anterior. O valor de retorno desta função não está definido<sup>8</sup>.

- `psr-remove-atribuicao!`: PSR x variável x valor → {}

---

<sup>6</sup> Para simplificar, vamos representar uma atribuição como um par (variável . valor).

<sup>7</sup> A ordem de duas listas l1 e l2 é consistente sse para qualquer conjunto de 2 elementos de l1 {e1,e2}, esse conjunto não aparece em l2 com a ordem trocada, e vice-versa. Por exemplo, (1 2 3 4) é consistente com (1 3 -1), mas não é consistente com (2 1).

<sup>8</sup> Ou seja, não interessa o que é retornado, mas também não podem usar o valor de retorno quando invocam a função.

Este modificador recebe um PSR e uma variável, e **altera** o PSR recebido, removendo qualquer atribuição à variável que tenha sido feita anteriormente. A variável passa efetivamente a não estar atribuída. O valor de retorno desta função não está definido.

- `psr-altera-dominio!`: PSR x variável x domínio  $\rightarrow \{\}$

Este modificador recebe um PSR, uma variável, e um domínio e **altera** o PSR recebido, alterando o domínio associado à variável recebida, para que passe a ser o domínio recebido. O valor de retorno desta função não está definido.

- `psr-completo-p`: PSR  $\rightarrow$  lógico

Este reconhecedor recebe um PSR e retorna T se estiver completo, i.e. se tiver uma atribuição para todas as variáveis. Retorna NIL caso contrário.

- `psr-consistente-p`: PSR  $\rightarrow$  lógico, inteiro

Esse reconhecedor recebe um PSR e retorna 2 valores<sup>9</sup>. O primeiro valor é um valor lógico que é T se o PSR for consistente, ou seja, se todas as restrições do PSR se verificarem para as atribuições existentes, e NIL caso contrário. O 2.º valor retornado corresponde ao número de testes de consistência<sup>10</sup> que foram necessários para determinar a consistência.

- `psr-variavel-consistente-p`: PSR x variável  $\rightarrow$  lógico, inteiro

Este teste recebe um PSR e uma variável, e retorna 2 valores. O primeiro valor é um valor lógico que é T se a variável é consistente tendo em conta as atribuições do PSR. Uma variável é consistente se todas as restrições directamente relacionadas<sup>11</sup> com essa variável são verificadas. Caso contrário o 1.º valor retornado é NIL. O 2.º valor retornado corresponde ao número de testes de consistência que foram necessários para determinar a consistência.

- `psr-atribuicao-consistente-p`: PSR x variável x valor  $\rightarrow$  lógico, inteiro

Este teste recebe um PSR, uma variável e um valor, e retorna 2 valores. O primeiro valor é T se a atribuição do valor à variável é consistente com as restantes atribuições do PSR, e NIL caso contrário. O 2.º valor retornado corresponde ao número de testes de consistência que foram necessários para determinar a consistência. **Atenção:** este método deve garantir que no fim da sua execução, o PSR recebido tem que ficar exactamente igual a como estava no início.

- `psr-atribuicoes-consistentes-arco-p`: PSR x variável x valor x variável x valor  $\rightarrow$  lógico, inteiro

Este teste recebe um PSR, uma variável v1, e um valor para essa variável, uma variável v2 e um valor para essa variável, e retorna 2 valores. O primeiro valor é T se as duas atribuições forem consistentes em arco entre si. Ou seja, se a atribuição a v1 é consistente com a atribuição a v2<sup>12</sup>, considerando as outras atribuições já existentes. É NIL caso contrário. O 2.º valor retornado

---

<sup>9</sup> Vejam a primitiva `values` do Lisp para retornar mais do que um valor.

<sup>10</sup> O número de testes de consistência corresponde à quantidade de restrições testadas, ou seja ao número de funções de avaliação invocadas.

<sup>11</sup> Ou seja, não é necessário verificar todas as restrições.

<sup>12</sup> Aqui não se devem testar todas as restrições, mas apenas as restrições que envolvam as duas variáveis.

corresponde ao número de testes de consistência que foram necessários para determinar a consistência. **Atenção:** este método deve garantir que no fim da sua execução, o PSR recebido tem que ficar exactamente igual a como estava no início.

## 2.2 Funções a implementar

Para além dos tipos de dados especificados na secção anterior, é obrigatória também a implementação das seguintes funções.

### 2.2.1 Funções de conversão

Estas funções estão relacionadas com a conversão de um problema do puzzle *Fill-a-Pix* para uma representação em PSR:

- `fill-a-pix->psr: array → PSR`

Este transformador recebe um array correspondente a uma tabela bidimensional  $m \times n$  de um puzzle *Fill-a-Pix* por resolver (ou seja apenas contem as restrições para as várias casas), e retorna um PSR que representa o problema de resolver esse puzzle particular (representado como um PSR). A ordem das variáveis, domínios e restrições usadas para criar o psr é muito importante. Um domínio deverá ser ordenado do valor mais baixo para o mais alto (caso seja um número), e as variáveis e restrições devem ser criadas percorrendo o array de cima para baixo e da esquerda para a direita.

- `psr->fill-a-pix: PSR x inteiro x inteiro → array`

Este transformador recebe um PSR resolvido, um inteiro que representa o número de linhas  $l$ , e outro que representa o número de colunas  $c$ , e devolve um array bidimensional de  $l$  linhas e  $c$  colunas, contendo para cada posição linha/coluna a atribuição da variável correspondente do PSR. Deve ser usado o 0 para representar uma casa branca, e o 1 para representar uma casa preta.

### 2.2.2 Procura Retrocesso Simples

As funções descritas nesta subsecção correspondem à versão mais simples a implementar para resolver um PSR. Estas funções deverão ser implementadas na 1.ª entrega, e uma vez implementadas, podem experimentar resolver os puzzles mais simples.

- `procura-retrocesso-simples: PSR → PSR, inteiro`

Esta função recebe um PSR, e tenta resolvê-lo usando uma procura com retrocesso simples, i.e. sem usar qualquer heurística para escolher a próxima variável e valor<sup>13</sup>, e sem qualquer mecanismo de inferência. Retorna 2 valores, o primeiro é o PSR resolvido, ou NIL caso não exista solução. O segundo valor corresponde ao número de testes de consistência efectuados<sup>14</sup> durante a procura. Não há qualquer problema em alterar o PSR recebido. No entanto devem também ter o cuidado do algoritmo ser independente do problema, ou seja deverá funcionar para este

---

<sup>13</sup> Neste caso, as variáveis e valores são escolhidos pela ordem em que aparecem.

<sup>14</sup> Vão ter que usar o 2.º valor retornado pelos testes de consistência. Para saber como o fazer, consultem a primitiva `multiple-value-bind` do lisp.

problema do Puzzle Fill-a-Pix, mas deverá funcionar também para qualquer outro problema<sup>15</sup> de Satisfação de Restrições.

- resolve-simples: array → array

Esta função recebe um array com um puzzle *Fill-a-Pix* por resolver, tenta resolvê-lo usando a procura-retrocesso-simples e retorna o resultado final como um array do problema *Fill-a-Pix* resolvido. Se não houver solução, deve ser retornado NIL.

### 2.2.3 Procuras mais avançados

Nesta subsecção serão descritas algumas das funções a implementar para a 2.ª entrega do projecto. Correspondem a versão mais elaboradas do algoritmo de procura por retrocesso e da função resolve.

- procura-retrocesso-grau: PSR → PSR, inteiro

Esta função recebe um PSR, e tenta resolvê-lo usando uma procura com retrocesso, sem qualquer mecanismo de inferência/propagação de restrições, mas usando a heurística de grau para escolher a próxima variável a ser instanciada. O segundo valor retornado corresponde ao número de testes de consistência efectuados durante a procura. Não pode ser usada qualquer heurística para escolha do próximo valor a ser testado.

- procura-retrocesso-fc-mrv: PSR → PSR, inteiro

Esta função recebe um PSR, e tenta resolvê-lo usando uma procura com retrocesso, usando obrigatoriamente Forward Checking como mecanismo de propagação de restrições, e a heurística MRV (Minimum Remaining Values) para escolha da próxima variável a escolher. Não pode ser usada qualquer heurística para escolha do próximo valor a ser testado. Retorna o PSR resolvido, ou NIL caso não exista solução. O segundo valor retornado corresponde ao número de testes de consistência efectuados durante a procura. Não há qualquer problema em alterar o PSR recebido. No entanto devem também ter o cuidado do algoritmo ser independente do problema.

- procura-retrocesso-mac-mrv: PSR → PSR, inteiro

Esta função recebe um PSR, e tenta resolvê-lo usando uma procura com retrocesso, usando obrigatoriamente o algoritmo Maintain Arc Consistency (MAC) como mecanismo de propagação de restrições, e a heurística MRV (Minimum Remaining Values) para escolha da próxima variável a escolher. Não pode ser usada qualquer heurística para escolha do próximo valor a ser testado. Retorna o PSR resolvido, ou NIL caso não exista solução. O segundo valor retornado corresponde ao número de testes de consistência efectuados durante a procura. Não há qualquer problema em alterar o PSR recebido. No entanto devem também ter o cuidado do algoritmo ser independente do problema.

- resolve-best: array → array

Esta função recebe um array com um puzzle *Fill-a-Pix* por resolver, tenta resolvê-lo usando a melhor procura/combinção de heurísticas, e retorna o resultado final como um array do problema *Fill-a-Pix* resolvido. Se não houver solução, deve ser retornado NIL. Esta função irá ser

---

<sup>15</sup> Portanto, é desaconselhado o uso de variáveis globais, e usar/aceder directamente a funções específicas deste puzzle. Tudo o que precisarem vai estar dentro do tipo PSR recebido.

a função usada para avaliar a qualidade da vossa versão final. Se assim o entenderem, nesta função já podem usar implementações específicas para o puzzle *Fill-a-Pix*.

Para além destas funções explicitamente definidas (que serão testadas automaticamente), **na 2.ª fase do projecto deverão implementar outras técnicas, heurísticas e testá-las. Cabe aos alunos decidir que técnicas/heurísticas adicionais irão precisar para que o vosso algoritmo final resolve-best seja o melhor possível.**

### 2.3 Estudo e análise dos algoritmos e heurísticas implementadas

Na parte final do projecto, é suposto os alunos compararem as diferentes variantes das procuras, algoritmos e heurísticas usadas para resolver o problema de Satisfação de Restrições, e perceberem as diferenças entre elas. Para isso deverão medir vários factores relevantes, e comparar os algoritmos num conjunto significativo de exemplos. Deverão também tentar analisar/justificar o porquê dos resultados obtidos.

Os resultados dos testes efectuados deverão ser usados para escolher a melhor variante do algoritmo e as melhores heurísticas a serem usadas. Esta escolha deverá ser devidamente justificada no relatório do projecto.

Nesta fase final é também pretendido que os alunos escrevam um relatório sobre o projecto realizado. Para além dos testes efectuados e da análise correspondente, o relatório do projecto deverá conter também informação acerca da implementação de alguns tipos e funções. Por exemplo, qual a representação escolhida para cada tipo (e porquê); qual a representação utilizada para as variáveis/domínios e restrições do Problema de Satisfação de Restrições; como é que as restrições foram implementadas, que outro tipo de variantes do algoritmo foram implementadas e como foram implementadas; quais as funções heurísticas implementadas e como é que foram implementadas, etc. Será disponibilizado um *template* do relatório para que os alunos tenham uma ideia melhor do que é necessário incluir no relatório final do projecto.

## 3 Ficheiro exemplos

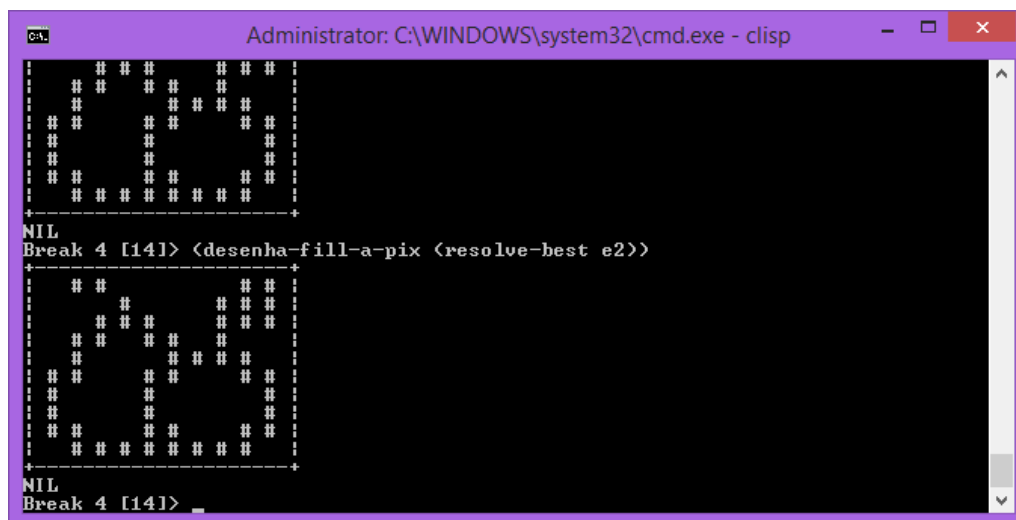
Foi fornecido um ficheiro de exemplos juntamente com o enunciado, que poderá ser usados para testar o vosso código. Para terem acesso aos puzzles de exemplo, deverão copiar o ficheiro fornecido para a mesma directoria onde se encontra o vosso projecto, e deverão colocar as seguintes instruções no início do vosso projecto.

```
(load (compile-file "exemplos.lisp"))
```

O ficheiro de exemplos fornecido implementa também uma função que pode ser usada para desenhar um array *Fill-a-Pix* resolvido no ecrã. Por exemplo, se fizeram a seguinte instrução:

```
(desenha-fill-a-pix (resolve-best e2))
```

Vão obter a seguinte imagem:



```
Administrator: C:\WINDOWS\system32\cmd.exe - clisp

NIL
Break 4 [141]> <desenha-fill-a-pix <resolve-best e2>>
+-----+
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
| # # # # # |
+-----+
NIL
Break 4 [141]>
```

Figura 3- Exemplo de chamada à função que desenha um puzzle Fill-a-Pix resolvido

## 4 Entregas, Prazos, Avaliação e Condições de Realização

### 4.1 Entregas e Prazos

A realização do projecto divide-se em 3 entregas. As duas primeiras entregas correspondem à implementação do código do projecto, enquanto que a 3.ª entrega corresponde à entrega do relatório do projecto.

#### 4.1.1 1.ª Entrega

Na primeira entrega pretende-se que os alunos implementem os **tipos de dados** descritos no enunciado (**secção 2.1**) bem como as funções correspondentes às **secções 2.2.1** e **2.2.2**. A entrega desta primeira parte será feita por via electrónica através do sistema **Mooshak**, até às **23:59** do dia **10/11/2014**. Depois desta hora, não serão aceites projectos sob pretexto algum<sup>16</sup>.

Deverá ser submetido um ficheiro .lisp contendo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo. Não é necessário incluir os ficheiros disponibilizados pelo corpo docente.

#### 4.1.2 2.ª Entrega

Na segunda entrega do projecto os alunos devem implementar o resto das funcionalidades descritas no enunciado (ver **secção 2.2.3**), incluindo as várias variantes, e as funções heurísticas pedidas. Deverão também já nesta fase fazer os testes que permitirão escolher qual melhor o algoritmo/heurística (embora não seja necessário incluir os testes no ficheiro de código submetido). A entrega da segunda parte será feita por via electrónica através do sistema **Mooshak**, até às **23:59** do dia **01/12/2014**. Depois desta hora, não serão aceites projectos sob pretexto algum.

<sup>16</sup> Note que o limite de 10 submissões simultâneas no sistema **Mooshak** implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns grupos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.



Deverá ser submetido um ficheiro .lisp contendo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo. Não é necessário incluir os ficheiros disponibilizados pelo corpo docente.

#### 4.1.3 3.ª Entrega

Na terceira entrega, os alunos deverão trabalhar exclusivamente no relatório do projecto. Não será aceite/avaliada qualquer nova entrega de código por parte dos alunos. Não existem excepções. O relatório do projecto deverá ser entregue (em .doc ou .pdf) exclusivamente por via electrónica no sistema **FENIX**, até às **12:00** do dia **9/12/2014**.

### 4.2 Avaliação

As várias entregas têm pesos diferentes no cálculo da nota do Projecto. A 1.ª entrega (em código) corresponde a **30%** da nota final do projecto (ou seja 6 valores). A 2.ª entrega (em código) corresponde a **35%** da nota final do projecto (ou seja 7 valores). Finalmente a 3.ª entrega, referente ao relatório, corresponde aos restantes **35%** da nota final do projecto (ou seja 7 valores).

A avaliação da 1.ª entrega será feita exclusivamente com base na execução correcta (ou não) das funções pedidas.

A avaliação da 2.ª entrega terá duas componentes. A 1.ª componente vale 6 valores e corresponde à avaliação da correcta execução das funções pedidas e à qualidade do algoritmo de resolução de Problemas de Satisfação de Restrições (quão bom ou mau é o vosso algoritmo a resolver puzzles *Fill-a-Pix*). A avaliação da qualidade do vosso algoritmo é feita usando-o para resolver uma série de puzzles com dificuldade incremental. Será tida em consideração a quantidade/complexidade dos puzzles resolvidos em tempo razoável, bem como a rapidez a resolvê-los.

A 2.ª componente vale 1 valor e corresponde a uma avaliação manual da qualidade do código produzido. Serão analisados factores como comentários, facilidade de leitura (nomes e indentação), estilo de programação e utilização de abs. procedimental.

Finalmente, a avaliação da 3.ª entrega corresponde à avaliação do relatório entregue. No *template* de relatório que será disponibilizado mais tarde, poderão encontrar informação mais detalhada sobre esta avaliação.

### 4.3 Condições de realização

**O código desenvolvido deve compilar em CLISP 2.49 sem qualquer “warning” ou erro.** Todos os testes efectuados automaticamente, serão realizados com a versão compilada do vosso projecto. Aconselhamos também os alunos a compilarem o código para os seus testes de comparações entre algoritmos/heurísticas, pois a versão compilada é consideravelmente mais rápida que a versão não compilada a correr em lisp.

**No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII, sob pena de falhar todos os testes automáticos.** Isto inclui comentários e cadeias de caracteres.

É prática comum a escrita de mensagens para o ecrã, quando se está a implementar e a testar o código. Isto é ainda mais importante quando se estão a testar/comparar os algoritmos. No entanto, **não se**

**esqueçam de remover/comentar as mensagens escritas no ecrã na versão final** do código entregue. Se não o fizerem, correm o risco dos testes automáticos falharem, e irão ter uma má nota na execução.

A avaliação da execução do código do projecto será feita automaticamente através do sistema *Mooshak*, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efectuar uma nova submissão pelo menos 15 minutos depois da submissão anterior. Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos não implica, pois, que esse projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correcto.

Duas semanas antes do prazo da 1.ª entrega (isto é, na Segunda-feira, 28 de Outubro), serão publicadas na página da cadeira as instruções necessárias para a submissão do código no *Mooshak*. Apenas a partir dessa altura será possível a submissão por via electrónica. Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverão portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretendem que seja avaliada. Não serão abertas excepções.<sup>17</sup>

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Projectos muito semelhantes serão considerados cópia e rejeitados. A detecção de semelhanças entre projectos será realizada utilizando *software* especializado<sup>18</sup>, e caberá exclusivamente ao corpo docente a decisão do que considera ou não cópia. Em caso de cópia, todos os alunos envolvidos terão 0 no projecto e serão reprovados na cadeira.

## 5 Competição do Projecto

Vai ser realizada uma competição entre todos os projectos submetidos para determinar quais os melhores projectos a resolver puzzles do tipo *Fill-a-Pix*. Para poderem participar na competição, a vossa implementação tem que passar todos os testes de execução referentes às funções e tipos de dados definidos neste enunciado. A função a ser usada na competição é a função *resolve-best*.

Os 3 melhores classificados na competição, ou seja os projectos que consigam resolver os puzzles mais difíceis (em caso de empate, é utilizado o tempo utilizado como critério de desempate) serão premiados com as seguintes bonificações:

- 1.º Lugar – 1,5 valores de bonificação na nota final do projecto
- 2.º Lugar – 1,0 valores de bonificação na nota final do projecto
- 3.º Lugar – 0,5 valores de bonificação na nota final do projecto

---

<sup>17</sup> Note que, se efectuar uma submissão no Mooshak a menos de 15 minutos do prazo de entrega, fica impossibilitado de efectuar qualquer outra submissão posterior.

<sup>18</sup> Ver <http://theory.stanford.edu/~aiken/moss/>