Laboratórios de Informática II

Illuminatus — Etapa 2

Versão 1.1

2011/2012

Sugestões

Seguem-se várias sugestões sobre como se deve programar (algumas das quais são avaliadas) e que deveriam ser seguidos por todos os programadores.

1. Especificação
   (a) Pensar antes de programar
   (b) Criar a especificação
   (c) Pensar nas estruturas de dados
   (d) Pensar nos algoritmos
2. Testes
   (a) Escrever os testes ao mesmo tempo que se especifica o problema
   (b) Escrever os testes antes de programar
   (c) Pensar em testes para todas as condições de excepção
   (d) Testar sempre o código de forma sistemática
   (e) Correr sempre os testes quando se altera o código
3. Controle de versões
   (a) Arquivar uma versão de cada vez que se implementa uma parte do código
   (b) Nunca submeter uma nova versão antes de testar
   (c) Usar um dos seguintes sistemas:
      - CVS (comando `cvs`)
      - Subversion (comando `svn`)
      - GIT (comando `git`)
4. Legibilidade do código
   - Escrever funções pequenas
   - Escrever funções auxiliares sempre que necessário
   - Programar de forma modular
   - Usar nomes auto-documentáveis para as funções e variáveis
   - Evitar linhas compactas (muito comum em instruções cíclicas e instruções condicionais devido a demasiadas condições)
   - Espaçar o texto dentro de uma linha
   - Indentar o código
   - Comentar o código sempre que isto aumente a sua legibilidade
5. Documentação do código

Erros a evitar

- Nunca escrever código sem pensar

  Este é um erro que é infelizmente muito comum. Nunca se deve começar a escrever o código sem ter a noção de como o projecto vai funcionar como um todo. Ao começar a escrever código sem pensar cometemos erros que podem tornar o código muito pouco legível e díficil de manter. E às vezes pode fazer os programadores perder horas para descobrir que tem que reescrever o código que já possuem porque este não serve.

- Não copiar e colar código

  Este é um erro muito frequente. Ao copiar e colar o código estamos a cometer vários pecados ao mesmo tempo. Por um lado, o código torna-se mais difícil de manter visto que se se precisar de modificar o código implica mexer em todas as cópias. Por outro lado, se o código que copiámos tiver erros será necessário corrigir todas as cópias.

- Escrever código ilegível

  O código legível é mais fácil de manter. Isto é sobretudo verdade quando se trabalha em equipa mas não só. Temos que nos habituar a trabalhar em equipa e o código legível é mais fácil de perceber e de modificar.

- Não escrever funções grandes

  Ao escrever funções grandes é mais difícil de se ler o código porque se perde o contexto. Isto resolve-se criando funções auxiliares. De notar que esta partição do código deve fazer-se nos sítios lógicos e sempre criando funções auxiliares com nomes sugestivos.

- Separar o código do interface com o utilizador

  As funções não devem interagir directamente com o utilizador já que perdem uma parte da sua utilidade. Por exemplo, ao implementar o código das estratégias nunca se devem imprimir nada no ecrã já que com isso não se podem utilizar essas funções para escreverem outras mais complicados (por exemplo dentro do comando `rsv`).

- Separar a estrutura de dados do código

  Em programas complexos devem-se criar funções que manipulem os dados que servem duma camada de separação entre a estrutura de dados e o resto do código pois assim ganha-se independência da estrutura de dados. Pode-se mudar a estrutura de dados com impunidade tendo só que reescrever as funções que interagem com os dados em vez de ter que mexer em todo o código.

Comandos a implementar

Pretende-se implementar os seguintes comandos:

an comando que anula um comando que mudou o estado do jogo.

est3 comando que implementa um passo da estratégia 3;

est4 comando que implementa um passo da estratégia 4;

est5 comando que implementa um passo da estratégia 5;

Módulo de listas ligadas

Este módulo é importante para o comando anular (e é obrigatório). Assim deve ser implementado antes de implementar o comando anular.

Comando `an`

O comando `an` anula um comando que modificou o estado do tabuleiro. Se este comando for utilizado várias vezes seguidas deverá ir anulando comandos sucessivamente até o tabuleiro ficar tal como foi carregado. Sugere-se a utilização do conceito de pilha para a implementação da anulação de comandos. O comando `cr` não só não é anulável como também limpa toda a informação anterior. Quando não há mais comandos para anular deve-se dar o erro correspondente.

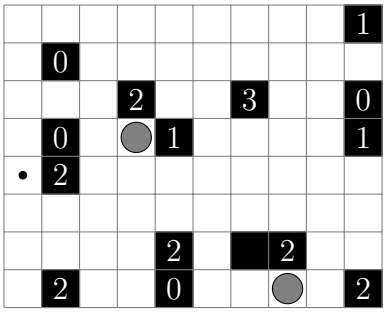
De notar que o comando `an` é muito importante para o comando `rsv` que resolve o tabuleiro e isso deve ser tido em conta na sua implementação.

Estratégia 3

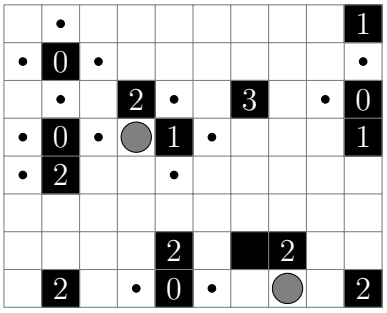
Primeiro uma clarificação sobre a estratégia 3: esta estratégia só é aplicada às casas vizinhas a casas que contém números e serve para marcar casas que não podem conter uma lâmpada. Esta estratégia deve ser dividida em duas partes:

- as casas ortogonais
- as casas diagonais

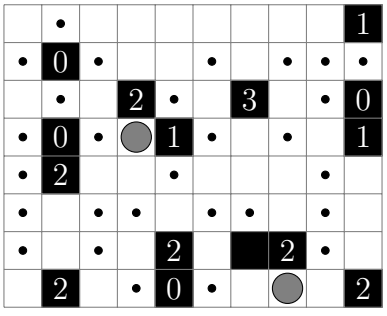
No que diz respeito às casas ortogonais isto acontece quando a restrição correspondente ao número já foi verificada. Consideremos o tabuleiro abaixo (repare que há uma casa que já foi marcada pelo jogador, sem razão nenhuma aparente) e pensemos só nas casas ortogonais.



Se virmos para começar só as casas ortogonais deveríamos chegar ao seguinte tabuleiro.

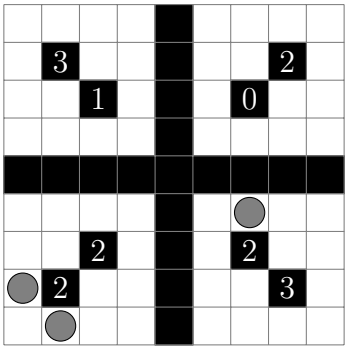


No caso das diagonais marcam-se as casas vizinhas de um número que não podem conter lâmpadas porque se o tivessem bloqueariam duas vizinhas ortogonais (que seriam iluminadas e por isso não poderiam conter lâmpadas) e isso impediria esse número de ser rodeado do número de lâmpadas necessárias.

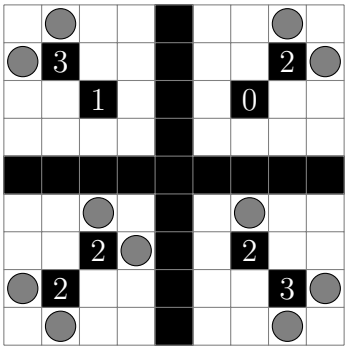


Estratégia 4

A estratégia 4 tenta colocar lâmpadas nas casas vizinhas de dois números que se tocam na diagonal. Como números vizinhos na diagonal partilham duas casas podemos por vezes inferir que as outras casas contenham lâmpadas.



Após a aplicação da estratégia 4 chegamos a este resultado. Repare que no caso do 1 e 3 e do 2 e do 3 (que são equivalentes) como estes só podem ter uma lâmpada em comum sabemos onde estão as outras lâmpadas do 3. No caso do 0 e do 2 e do 2 e do 2 (que são equivalentes) sabemos que as casas partilhadas não podem conter lâmpadas e logo sabemos onde estão as duas lâmpadas do 2.



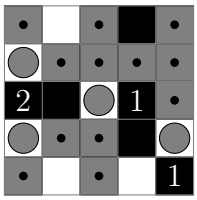
Estratégia 5

A estratégia 5 é um pouco mais complexa do que as anteriores. Assim, os alunos deverão estudar a estratégia de modo a descobrirem todas as situações possíveis e só depois avançar para a programação. Esta estratégia aplica-se olhando para as casas que ainda não estão iluminadas.

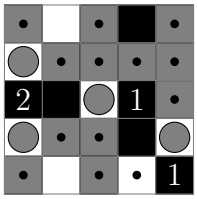
Uma casa está iluminada se:

1. Se contém uma lâmpada
2. Se há uma linha de casas não bloqueadas entre ela e uma lâmpada

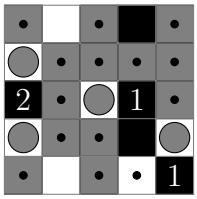
Esta estratégia aplica-se quando há só uma casa livre que precisa de conter uma lâmpada para iluminar uma dada casa. Isto pode acontecer por exemplo se a casa que pretendemos iluminar está livre e só colocando uma lâmpada nessa casa é que se pode iluminar a casa ou se a casa está marcada e existe só uma casa livre nas linhas ou colunas que dão para essa casa.



Repare que no caso da casa (2, 1) sabemos que para iluminar essa casa temos que colocar lá uma lâmpada. No caso da casa (2, 5) não sabemos (mas saberíamos se aplicássemos a estratégia 3 antes de aplicar a estratégia 5).



Vejamos agora outro caso. Este caso é muito semelhante ao anterior mas agora também temos que colocar uma lâmpada na casa (2, 5) pois é a única maneira de iluminar a casa (4, 5). Tal como no caso também é necessário uma lâmpada na casa (2, 1).



De notar este novo caso que também é ligeiramente semelhante. Neste caso a estratégia 5 passa por colocar a lâmpada na casa (2, 5) pois é a única maneira de iluminar a casa (4, 5). Repare que neste caso essa lâmpada também ilumina a casa (2, 1).

Sugestões para a implementação das estratégias

1. Começar por descobrir todas as situações possíveis
2. Escrever um teste para cada situação
3. Especificar a resolução de cada situação
4. Implementar
5. Correr os testes para garantir que todos os problemas foram resolvidos