Fundamentos de Programação

Projecto – Segunda Parte

23 de Novembro de 2009

Mastermind

O jogo do Mastermind foi inventado em 1970 por Mordecai Meirowitz, um especialista em telecomunicações. O jogo não foi inicialmente considerado apelativo pelos grandes fabricantes de jogos e acabou por ser comercializado em 1971 por uma pequena empresa inglesa de plásticos, tendo obtido um sucesso instantâneo. O Mastermind recebeu o prémio "Game of the Year Award" em 1973 e é actualmente considerado como o jogo com maior sucesso dos anos 70.

Uma empresa de jogos famosa, a YNOS está a planear o lançamento de uma nova geração de plataformas de jogos, a PS4, em que um dos jogos oferecidos é uma nova versão do célebre Mastermind. Considerando que a linguagem Scheme é muito adequada para desenvolver jogos e estando a YNOS ciente da qualidade dos alunos do Instituto Superior Técnico, a YNOS lançou o desafio aos alunos da disciplina de Fundamentos de Programação de serem eles os programadores a desenvolver o jogo. Os alunos de Fundamentos de Programação já estão agrupados em grupos de projectos, sendo cada grupo o responsável pelo desenvolvimento completo de um jogo. Para manter a uniformidade com a fantástica interface gráfica a oferecer na PS4, a YNOS exigiu que os participantes neste trabalho utilizem uma interface gráfica que é propriedade da YNOS, cujo código não é revelado, e que é acedida através do protocolo de comunicação apresentado na Secção 3.

No final da disciplina, e com o patrocínio da YNOS, será realizado um campeonato entre os vários jogadores desenvolvidos. A YNOS conseguiu convencer os docentes da disciplina a premiar os quatro melhores programas com os prémios apresentados na Secção 6.

Em resumo, a finalidade deste projecto consiste no desenvolvimento de um programa em Scheme, para o jogo do Mastermind, o qual pode ser jogado entre dois jogadores, um dos jogadores pode ser humano e o outro jogador é "computacional". No seu trabalho irá desenvolver tanto o programa que faz a gestão do jogo como o programa que corresponde ao jogador computacional. O jogo a realizar é uma variante do Mastermind, cujas regras são apresentadas na Secção 1.

Não entre em pânico com a aparente complexidade do projecto. As sugestões apresentadas na Secção 4 correspondem a uma ajuda preciosa de passos a seguir de modo a que este projecto se torne uma experiência de aprendizagem estimulante e gira. O corpo docente da disciplina fará tudo quanto estiver ao seu alcance para ajudar os participantes neste desafio a demonstrarem à YNOS a qualidade dos alunos do Instituto Superior Técnico. No entanto, o grosso do trabalho é do vosso lado.

Figura 1: Tabuleiro do Mastermind.

1 As regras do jogo

O Mastermind é jogado por dois jogadores, um que gera um código secreto (também conhecido por o *segredo*) e outro que tenta adivinhá-lo.

Os jogadores interaccionam utilizando um tabuleiro (ver Figura 1[1]) com vários componentes:

1. Uma zona escondida, numa ponta, que cobre o segredo;
2. Doze linhas visíveis, cada uma das quais com lugar para quatro pinos chave (buracos maiores) e para quatro pinos resposta (buracos mais pequenos);
3. Um conjunto de pinos chave, com as cores "red", "aqua", "brown", "orange", "yellow" e "lime", que são colocados na zona do segredo e nos furos maiores em cada linha do tabuleiro;
4. Um conjunto de pinos resposta, de cor preta e vermelha (na figura, que corresponde a uma versão antiga deste jogo, os pinos resposta são vermelho e branco) que são colocados nos furos mais pequenos de cada linha.

Antes do início do jogo, os jogadores decidem qual deles vai ser responsável por gerar o código secreto (este jogador é chamado o *guardião do segredo*) e qual deles vai tentar adivinhar o segredo (este jogador é chamado o *explorador*).

1. O guardião do segredo começa por colocar na zona escondida do tabuleiro uma sequência de quatro pinos chave (o segredo) podendo estes pinos ser de qualquer

[1]Fotografia cedida por User:ZeroOne.

uma das cores disponíveis[2] e podendo haver repetição de cores.

2. O explorador tem que descobrir o segredo num máximo de doze jogadas, se não o conseguir fazer perde o jogo. Se conseguir adivinhar o segredo o seu resultado é tanto melhor quanto menor for o número de tentativas necessárias para o conseguir.

Em cada tentativa, o explorador coloca na primeira linha vazia a sequência de pinos chave que constitui a sua sugestão do que será o segredo (a *adivinha*, a qual apenas pode conter as cores possíveis para o segredo). Como resposta, o guardião do segredo coloca zero a quatro pinos resposta nos buracos mais pequenos da mesma linha. É colocado um pino preto por cada pino da tentativa que acerta na cor e na posição, é colocado um pino vermelho por cada pino cuja cor existe no segredo mas numa outra posição.

Assim que é fornecida uma resposta para a tentativa, sucedem-se novas tentativas e respectivas respostas até que o explorador adivinhe o segredo ou até que tenham sido feitas doze tentativas falhadas.

2 Objectivos do projecto

O projecto tem dois objectivos:

1. O desenvolvimento de um programa que corresponde a um jogador automático que tanto pode ter o papel de guardião do segredo como o papel de explorador;
2. O desenvolvimento de um programa de controle, para controlar um jogo entre os dois jogadores, o qual deverá apresentar uma comunicação adequada com um programa que gera uma interface gráfica para o jogo, programa esse que é designado por *IGM* (*Interface Gráfica com o Mastermind*).

A arquitectura do seu programa é constituída pelos módulos que se apresentam na Figura 2. No seu trabalho deverá desenvolver o jogador (que tanto terá o papel de guardião do segredo como o do explorador), deverá desenvolver o programa de controle e deverá utilizar a IGM.

2.1 O jogador automático

Deverá desenvolver um programa que saiba jogar o Mastermind. O seu programa poderá ser tão sofisticado quanto desejar.

Para questões relacionadas com a classificação do projecto, qualquer jogador, por mais estúpido que seja, desde que respeite as regras do jogo e *não faça tentativas repetidas*, terá a classificação completa.

No entanto, os jogadores inteligentes poderão ser recompensados com um bónus adicional com base nos resultados do torneio que vai ser realizado entre todos os grupos (ver Secção 6). Só serão admitidos ao torneio os jogadores que satisfaçam *todas* as especificações indicadas na Secção 3.

[2] As cores “red”, “aqua”, “brown”, “orange”, “yellow” e “lime”.

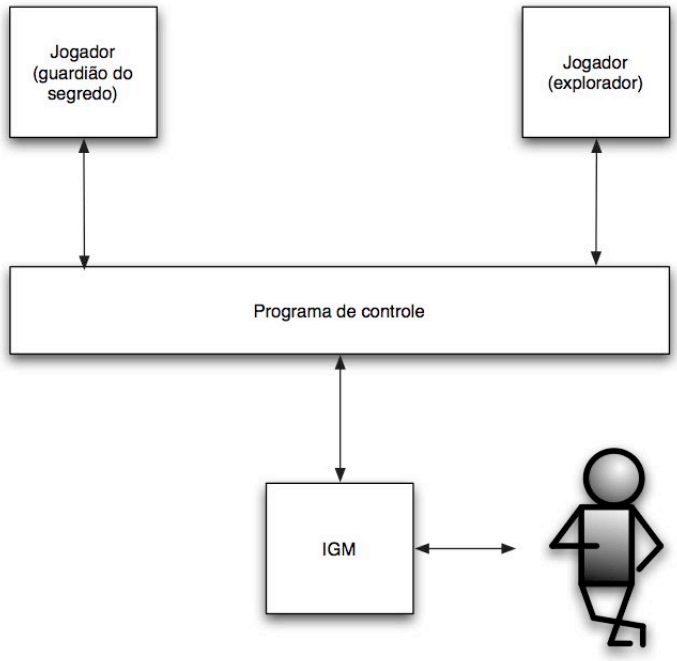

Figura 2: Módulos do programa e suas interacções.

2.2 O programa de controle

No seu projecto, deverá também escrever um programa que controle todo o jogo, desde o seu início até este terminar.

Para iniciar um jogo, o programa de controle tem de saber qual o papel do jogador humano:

- Se o jogador humano é o explorador, o programa de controle deve criar um jogador automático correspondente ao guardião do segredo e pedir-lhe um novo segredo.

  Deve depois iniciar um ciclo de jogadas em que vai pedindo ao jogador humano para introduzir uma adivinha (uma tentativa para descobrir o segredo). Esta adivinha é avaliada pelo guardião do segredo que responde de acordo com as regras do jogo. Se esta resposta não estiver correcta, o programa de controle deve declarar o jogador automático como perdedor.

- Se o jogador humano é o guardião do segredo, o programa de controle deve pedir-lhe para criar o segredo (sendo o programa de controle informado deste segredo) e deve criar um jogador automático com o papel de explorador.

  Deve, depois, iniciar um ciclo de jogadas em que pede ao explorador para introduzir uma tentativa para descobrir o segredo. Deve mostrar a tentativa produzida pelo jogador automático na interface gráfica bem como a resposta calculada (esta resposta é calculada pelo programa de controle para não sobrecarregar o jogador humano) de acordo com as regras do jogo.

O jogo termina quando se esgotaram todas as tentativas ou quando o explorador consegue adivinhar o segredo.

Figura 3: Final de um jogo.

2.3 A IGM

Será disponibilizado um ficheiro com o programa que permite ao seu programa utilizar uma interface gráfica, devendo o seu programa de controle comunicar adequadamente com a IGM. Lembre-se que este programa é propriedade da YNOS e o seu código não será revelado em circunstância alguma. Na Figura 3 apresenta-se um exemplo correspondente à interface no final de um jogo[3].

3 Protocolos de comunicação

Como acontece com muitos programas desenvolvidos no seio de empresas e de organizações, o seu programa terá de comunicar com outros programas.

Para gerir a comunicação entre programas, é definido um protocolo de comunicação, vulgarmente conhecido por API (do inglês *Application Programming Interface*), que define de um modo exaustivo todas as interacções possíveis entre os programas, bem como o conteúdo da informação partilhada.

Um protocolo de comunicação é constituído, quer por tipos abstractos de informação, quer por procedimentos com nomes e parâmetros previamente acordados, o que garante que os vários programas podem invocar procedimentos uns dos outros.

Nesta secção apresentam-se os protocolos que o seu programa tem de seguir. Para além dos tipos e procedimentos definidos nesta secção, cada grupo é livre de definir os tipos de informação e procedimentos que desejar, apenas sujeitos às restrições definidas nesta secção.

[3]Esta interface pode parecer um pouco pobre. Na realidade, a YNOS não quer revelar, por enquanto, a interface gráfica das novas PS4, tendo fornecido esta interface simplista para testar o jogo.

3.1 Paradigmas de programação e de comunicação

Convenciona-se que o jogador, o programa de controle e a IGM correspondem a procedimentos com estado interno comunicando entre si através do envio de mensagens.

O envio de mensagens é realizado recorrendo ao procedimento `em`, significando `e`nvia `m`ensagem, com o seguinte código, o qual *deve ser incluído* no seu programa:

```
(define (em obj mens . args)
  (apply (obj mens) args))
```

Neste procedimento:

- O argumento `obj` corresponde ao nome do procedimento ao qual a mensagem é enviada;
- O argumento `mens` (uma entidade do tipo símbolo) representa o tipo da mensagem enviada;
- Os argumentos opcionais, `args`, são zero ou mais objectos computacionais que são parte da mensagem.

Por exemplo, `(em mm-interface 'fpmm-apaga-linha 3)`, envia ao procedimento `mm-interface` uma mensagem do tipo `'fpmm-apaga-linha` e que tem um argumento adicional `3`.

As mensagens a trocar entre os vários procedimentos são apresentadas nas secções 3.7.2, 3.6.2 e 3.5.2.

3.2 Nomes

No seu programa *não podem ser definidos* nomes começados por "`fpmm-`" nem por '`mm-`" para além dos que são descritos na próxima secção.

3.3 Código pré-definido a utilizar

O código apresentado nesta secção, que inclui a invocação do procedimento que implementa a interface e do procedimento que implementa o programa de controle, deve ser incluído na solução a desenvolver:

```
(define mm-controle null)
(define mm-interface null)

(define (mastermind nome-utilizador tipo)
  (set! mm-interface (fpmm-cria-interface))
  (set! mm-controle (cria-controle nome-utilizador tipo)))
```

Ao ser avaliado o procedimento `mastermind` tem início um novo jogo[4].

Para carregar a interface coloque a seguinte forma no seu ficheiro[5]:

```
(require "interface.zo")
```

3.4 Tipos de informação

Combinou-se com a YNOS que a informação que é trocada entre os vários módulos do programa é definida de uma forma genérica através de tipos abstractos de informação. Os tipos de informação são apresentados na Secção 9 e já foram realizados na primeira parte do projecto.

Desde o momento de lançamento do enunciado da primeita parte do projecto, a YNOS solicitou a adição de um novo elemento ao tipo *pino-chave* (ver Secção 9.1) e solicitou que o tipo *jogadas* (apresentado na Secção 9.5) seja um tipo mutável.

Para a segunda parte do projecto, os grupos podem utilizar um ficheiro disponibilizado no Fénix que contém a implementação de todos os tipos de informação listados na Secção 9, com a excepção dos tipos *pino-chave* e *jogadas* que terão de ser modificados de acordo com as novas exigências.

Se desejarem, os grupos podem, em alternativa, utilizar a definição dos tipos de informação que realizaram na primeira parte do projecto, tendo em atenção que dois destes tipos devem ser modificados, o tipo *pino-chave* e o tipo *jogada*.

3.5 Jogador automático

3.5.1 Criação

O procedimento correspondente ao jogador deve corresponder a um procedimento cujo nome é "`cria-jogador`<número de grupo>", em que <número de grupo>[6] corresponde à identificação do seu grupo. Este procedimento devolve um procedimento correspondente a um jogador automático.

O procedimento correspondente ao jogador automático recebe como argumento um símbolo que indica se o jogador vai ser o explorador ou o guardião do segredo. Se receber o símbolo `segredo` o seu papel é ser o guardião do segredo; se receber o símbolo `advinha` o seu papel é ser o explorador. Por exemplo, para a invocação do jogador do grupo 3, temos as seguintes alternativas:

```
(cria-jogador3 'segredo)
```

e

[4] Note-se que a avaliação do texto do programa por si desenvolvido não deve dar início a um novo jogo sob a pena de o programa falhar todos os testes a realizar pelo corpo docente. Isto quer dizer que não deve ser colocada a forma (`mastermind` <nome-utilizador> <tipo>) ao nível de topo no texto do programa.

[5] O ficheiro `interface.zo` deve existir na mesma directoria do seu programa.

[6] Se o número de grupo for inferior a 10, escreva apenas um dígito, por exemplo, `cria-jogador3`, e não `cria-jogador03`.

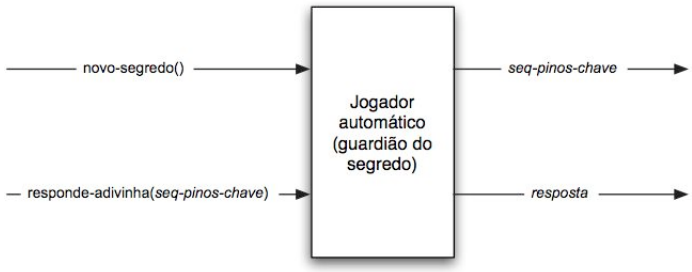

Figura 4: Mensagens (e respostas) para o guardião do segredo.

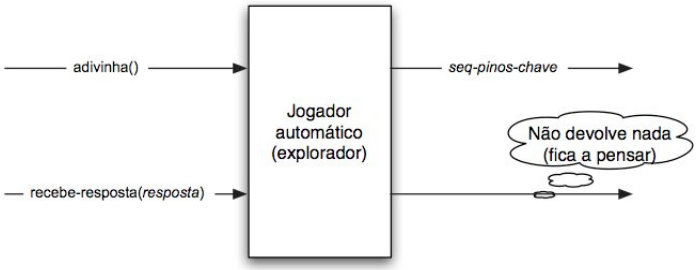

Figura 5: Mensagens (e respostas) para o explorador.

```
(cria-jogador3 'adivinha)
```

3.5.2 Comunicação com o jogador automático

O programa que implementa o jogador automático deve ser capaz de processar correctamente as mensagens descritas nesta secção (nesta descrição <jogador> representa o procedimento correspondente a um jogador).

Mensagens para o guardião do segredo. O guardião do segredo poderá receber dois tipos de mensagens (Figura 4).

- `(em <jogador> 'novo-segredo)`

  Esta mensagem é enviada pelo programa de controle, no início do jogo, para obter o segredo. O jogador devolve o segredo, um elemento do tipo *seq-pinos-chave*.

- `(em <jogador> 'responde-adivinha <seq-pinos-chave>)`

  Esta mensagem é enviada pelo programa de controle para obter uma resposta relativa à tentativa produzida pelo utilizador, um elemento do tipo *seq-pinos-chave*. O jogador devolve uma *resposta*.

Mensagens para o explorador. O explorador poderá receber dois tipos de mensagens (Figura 5).

- `(em <jogador> 'adivinha)`

  Esta mensagem é enviada pelo programa de controle, no decurso do jogo, para obter a jogada do <jogador>, um elemento do tipo *seq-pinos-chave*.

  Como resposta a esta mensagem, o <jogador> devolve a *seq-pinos-chave* correspondente à sua tentativa para adivinhar o segredo. O jogador deve manter internamente a informação sobre as tentativas produzidas (evitando assim a duplicação de tentativas).

- `(em <jogador> 'recebe-resposta <resposta>)`

  Esta mensagem é enviada pelo programa de controle e corresponde à avaliação da última tentativa feita pelo explorador. Ao receber esta mensagem o jogador deve actualizar a informação que tem sobre o jogo, não devolvendo nada.

3.6 Programa de controle

O programa de controle é um procedimento com estado interno. Este procedimento deve ser o valor do nome `mm-controle`, que deve ser acessível ao programa de interface, para que este possa enviar mensagens ao programa de controle.

O programa de controle dirige todo o jogo, enviando e recebendo mensagens de ambos os jogadores e da interface. O programa de controle é também o responsável por verificar se os jogadores estão a jogar de acordo com as regras e, em caso de não o estarem, termina imediatamente o jogo.

Para além disso, o programa de controle calcula a resposta que deverá ser fornecida ao explorador pelo guardião do segredo quando este é o jogador humano.

3.6.1 Criação

O procedimento correspondente ao programa de controle é criado através da invocação do procedimento `(cria-controle <símbolo> <tipo>)`, em que <símbolo> corresponde ao nome do utilizador e <tipo> corresponde ao papel que o jogador automático vai ter (ver Secção 3.5.1).

3.6.2 Comunicação com o programa de controle

O programa de controle deve ser capaz de receber e de processar correctamente as seguintes mensagens enviadas pela IGM (ver Figura 7):

- `(em mm-controle 'nova-adivinha <seq-pinos-chave>)`

  Esta mensagem é enviada ao programa de controle pela IGM quando o utilizador carrega no botão "Jogar" após ter introduzido uma nova tentativa para adivinhar o segredo.

- `(em mm-controle 'novo-segredo <seq-pinos-chave>)`

  Esta mensagem é enviada ao programa de controle pela IGM quando o utilizador carrega no botão "Jogar" após ter introduzido o segredo.

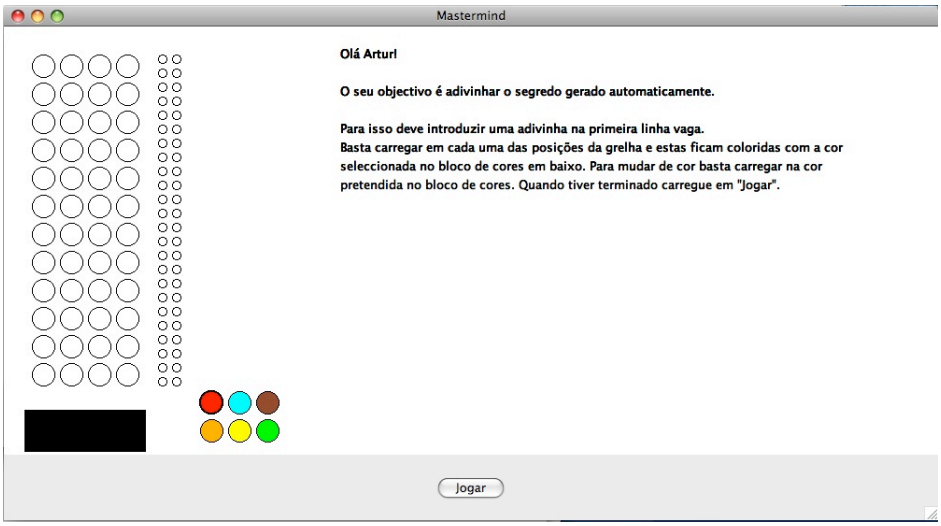

Figura 6: Tabuleiro inicial do jogo mostrado ao explorador.

- `(em mm-controle 'continuar)`

  Esta mensagem é enviada ao programa de controle pela IGM quando o utilizador carrega no botão "Jogar" após ter tido oportunidade de verificar a última tentativa realizada pelo jogador automático para adivinhar o segredo.

3.7 Programa para manipular a IGM

A IGM é um procedimento com estado interno com a qual o programa de controle comunica através de mensagens. A IGM usa uma janela através da qual é feita a comunicação entre o utilizador e o programa que controla o jogo. Quando é usado o botão "Jogar", é enviada uma mensagem ao programa de controle com a informação relativa a acções realizadas pelo utilizador.

3.7.1 Criação

O procedimento correspondente ao programa que implementa a interface deve ser criado através da invocação do procedimento `(fpmm-cria-interface)` – ver a Secção 3.3.

3.7.2 Comunicação com a IGM

A IGM é capaz de receber e de processar correctamente as seguintes mensagens:

Mensagem para iniciar a interface do jogo

- `(em mm-interface 'fpmm-inicia-jogo)`

  Cria a janela de jogo.

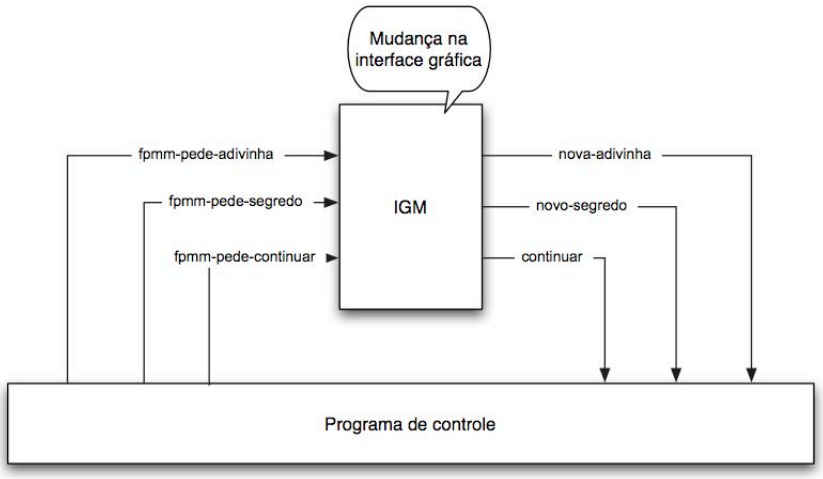

Figura 7: Mensagens que alteram a interface e que geram mensagens.

Mensagens respeitantes ao decorrer jogo

- `(em mm-interface 'fpmm-pede-adivinha` <inteiro>`)`

  Obtém do utilizador uma nova tentativa, uma <seq-pinos-chave>, a ser introduzida na linha dada por <inteiro> (0 e 11). Este inteiro deverá ser mantido e actualizado pelo programa de controle.

  Quando o jogador carrega no botão "Jogar" é enviada, ao programa de controle, a mensagem

  `(em mm-controle 'nova-adivinha` <seq-pinos-chave>`).`

- `(em mm-interface 'fpmm-pede-segredo)`

  Coloca a interface em modo de ler o segredo fornecido pelo utilizador. Obtém do utilizador o segredo, uma <seq-pinos-chave> , a ser introduzida na linha correspondente ao rectângulo preto.

  Quando o utilizador carrega no botão "Jogar", após ter introduzido o segredo, a seguinte mensagem é enviada ao programa de controle:

  `(em mm-controle 'novo-segredo` <seq-pinos-chave>`)`

- `(em mm-interface 'fpmm-pede-continuar)`

  Fica à espera que o utilizador carregue no botão "Jogar".

  Quando o jogado carrega no botão "Jogar", a seguinte mensagem é enviada ao programa de controle:

  `(em mm-controle 'continuar)`

- `(em mm-interface 'fpmm-mostra-adivinha` <seq-pinos-chave> <inteiro>`)`

  Mostra uma tentativa do utilizador, uma <seq-pinos-chave>, na linha <inteiro> .

- `(em mm-interface 'fpmm-mostra-resposta` <resposta> <inteiro>`)`

  Mostra uma resposta a uma tentativa do jogador automatico, uma <resposta>, na linha <inteiro> .

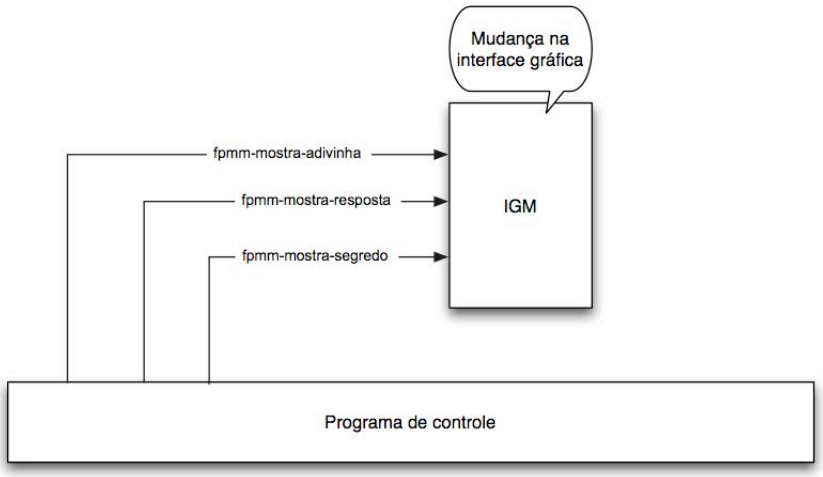

Figura 8: Mensagens que apenas alteram a interface e não geram mensagens.

- `(em mm-interface 'fpmm-mostra-segredo` <seq-pinos-chave>`)`

  Mostra o segredo, uma <seq-pinos-chave>, na linha correspondente ao rectângulo preto.

Mensagens para a alteração do texto que aparece na interface

As seguintes mensagens, enviadas à IGM permitem a alteração do texto que surge no lado direito da janela.

- `(em mm-interface 'fpmm-escreve-linha` <string> <inteiro>`)`

  Escreve, na parte direita da janela, que contém texto, o texto correspondente a <string>, o qual é colocado na linha cujo número é dado por <inteiro> (um inteiro entre 0 e 22).

- `(em mm-interface 'fpmm-apaga-linha` <inteiro>`)`

  Apaga, na parte direita da janela, que contém texto, o texto correspondente à linha cujo número é dado por <inteiro> (um inteiro entre 0 e 22).

- `(em mm-interface 'fpmm-fim-jogo` <string>`)`

  Escreve <string> na parte inferior direita da janela. Deve ser usado para declarar o vencedor ou perdedor. A partir desta altura a IGM fica bloqueada.

  No caso do utilizador ou do jogador automático ganharem o jogo, <string> deve ser "<nome> ganhou.", em que <nome> é o nome do utilizador ou "Computador", dependendo de quem ganhou;

  No caso da tentativa/resposta do jogador automático não ser válida, ou de repetir uma tentativa feita anteriormente, o seu programa deverá executar o passo anterior, substituindo a mensagem escrita por "Computador perdeu.".

  No caso da tentativa feita pelo utilizador não ser válida, ou de repetir uma tentativa feita anteriormente, o seu programa deverá executar o passo anterior, substituindo a mensagem escrita por "<nome> perdeu.".

4 Sugestão de passos a seguir

Desenvolver um programa com a complexidade descrita pode parecer uma actividade assustadora. Nesta secção apresentamos algumas sugestões para os passos a seguir no desenvolvimento do programa.

1. Assegure-se que tem todos os tipos de informação apresentados na Secção 9. Estes tipos serão fornecidos pelo corpo docente da disciplina, excepto no que respeita à implementação do tipo apresentado na Secção 9.5, o que deverá ser feito por si. Se utilizar os tipos que desenvolveu para a primeita parte do projecto, não se esqueça de fazer as alterações necessárias.

   Familiarize-se com estes tipos e teste cuidadosamente cada um dos seus procedimentos.

2. Começe a pensar no desenvolvimento do programa que corresponde ao jogador. Já se deve ter apercebido que este apresenta dois tipos distintos de comportamentos, consoante desempenhar o papel de guardião do segredo ou de explorador.

   O jogador corresponde a um procedimento com estado interno. Decida qual a informação que deve corresponder a este estado interno.

   O seu jogador deve estar preparado para receber as mensagens apresentadas na Secção 3.5.2 e produzir os comportamentos indicados. Escreva cada um destes comportamentos, um de cada vez, testando cuidadosamente o comportamento do seu jogador através da simulação de envio de cada mensagem.

   Não tente, nesta fase, que o seu explorador seja minimamente inteligente, mas apenas que jogue. Poderá querer utilizar um procedimento primitivo que, corresponde a um procedimento com estado interno, existente em Scheme, `random`, que recebe um argumento inteiro, n, e que devolve aleatoriamente um inteiro no intervalo de 0 a $n-1$. O facto de `random` ser um procedimento com estado interno significa que duas avaliações seguidas de `random` com o mesmo argumento devolvem valores diferentes. Poderá escrever um procedimento que converte o número devolvido por `random` para a côr de um pino chave.

   Lembre-se que o explorador não pode enviar duas vezes a mesma tentativa para decobrir o segredo. Neste aspecto é bom que o seu programa se *lembre* de todas as jogadas que fez, garantindo que nunca repete a mesma jogada.

3. Faça a "ligação" com a IGM, enviando, manualmente várias mensagens possíveis para o programa que corresponde à interface.

   Como as respostas a algumas destas mensagens correspondem a mensagens enviadas ao programa de controle (ver a Figura 8), sugerimos que use o seguinte código para testar o efeito do envio de cada mensagem, o qual indica qual o conteúdo da mensagem:

```
(define (cria-controle nome tipo)
  (define (novo-segredo seq)
    (display "controle: mensagem novo-segredo")
    (display seq)
    (newline))
 (define (continuar)
```

```
    (display "controle: mensagem continuar")
    (newline))
  (define (nova-adivinha seq)
    (display "controle: mensagem nova-adivinha")
    (display seq)
    (newline))
    (lambda (m)
      (case m
        ((novo-segredo) novo-segredo)
        ((continuar) continuar)
        ((nova-adivinha) nova-adivinha))))
```

Podendo testar este programa, por exemplo, com as seguintes mensagens:

```
(mastermind 'p 'adivinha)
(em mm-interface 'fpmm-inicia-jogo)
(em mm-interface 'fpmm-pede-segredo)
```

4. Depois de já ter uma boa ideia das mensagens enviadas e recebidas pelo programa de controle, começe a desenvolver o programa de controle. Já tem o jogador a trabalhar, já percebeu como interaccionar com a interface, é agora apenas questão de escrever um programa que interage adequadamente com os outros dois.

5. Quando tiver concluído o passo anterior, faça uma cópia dos seu programa e não a modifique mais. Esta poderá ser, em último caso, a versão final do seu programa se os próximos passos não correrem bem. Finalize também neste ponto toda a documentação do seu projecto. Verifique quais os aspectos que vão ser considerados na classificação (descritos na Secção 7) e veja como os pode maximizar. O seu trabalho deverá estar pronto para ser entregue e poderá obter a classificação máxima.

6. Se desejar dotar o seu programa de alguma inteligência, tentando ganhar alguns valores adicionais como resultado do torneio, comece agora a pensar no raciocínio que deverá ser seguido pelo explorador. Não se esqueça, não altere o programa que já fez e que deverá estar cuidadosamente guardado num ficheiro no seu computador.

5 Aspectos a evitar

Os seguintes aspectos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projecto):

1. Não pense que o projecto se pode fazer nos primeiros dias de Janeiro. Se apenas iniciar o seu trabalho neste período irá ver a Lei de Murphy mesmo em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).

2. Não pense que um dos elementos do seu grupo fará o trabalho por todos. Este é um trabalho de grupo e deverá ser feito em estreita colaboração (comunicação

e controle) entre os vários elementos do grupo, cada um dos quais com as suas responsabiliades. Tanto uma possível oral como perguntas no segundo teste sobre o projecto, servem para despistar estas situações.

3. Não tenha a ambição de fazer, de início, o jogador mais inteligente do mundo. Lembre-se que um jogador estúpido, mas que joga, é melhor classificado que que um jogador genial, mas que corresponde a um programa cheio de erros de execução.

4. Não descure a importância da documentação. Apenas o relatório vale 6 valores. Escreva a documentação ao longo do desenvolvimento do seu programa.

5. Não duplique código. Se dois procedimentos são muito semelhantes é natural que estes possam ser fundidos num único.

6. Não se esqueça que os procedimentos grandes são penalizados no que respeita ao estilo de programação.

7. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente erada.

8. Quando o ser programa gerar um erro, preocupe-se em descobrir *qual* a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.

6 Torneio

Será disputado um torneio de Mastermind entre os jogadores automáticos que tentam adivinhar o código que satisfaçam todas as especificações da Secção 3.

As regras detalhadas deste torneio serão publicadas na página da cadeira. Os quatro melhores jogadores (aqueles que foram seleccionados para as meias finais) serão recompensados com as seguintes notas adicionais à nota do projecto:

1. Primeiro classificado: 2,0 valores;

2. Segundo classificado: 1,5 valores;

3. Terceiro classificado: 1,0 valor;

4. Quarto classificado: 0,5 valores.

7 Classificação

A nota do projecto será baseada nos seguintes aspectos:

1. Execução correcta (40%).

   Esta parte da avaliação é feita recorrendo a um programa de avaliação automática que não só sugere uma nota face aos vários aspectos considerados, como também decide se o seu programa é apurado ou não para o torneio.

2. Facilidade de leitura, nomeadamente abstracção procedimental, abstracção de dados, nomes bem escolhidos, paragrafação correcta, qualidade (e não quantidade) dos comentários[7] e tamanho dos procedimentos (25%).

   O tamanho de cada um dos procedimentos contidos nos seus procedimentos de estado interno (não contando os comentários) deve ser inferior a um écran apresentado na janela de definições do Scheme, com a paragrafação adequada e com o tamanho "standard" para as letras.

3. Relatório (30%).

4. Estilo de programação (5%).

Os pesos das notas de cada uma das partes do projecto na nota final do projecto são os seguintes:

1. Primeira parte - 30%.
2. Segunda parte - 70%.

8 Condições de realização e prazos

A segunda parte do projecto deve ser entregue até às **15:00** horas do dia **4 de Janeiro de 2010**, na reprografia do DEI ou na portaria do Tagus (consoante o campus que corresponda ao grupo do projecto), e deverá constar de um relatório, incluindo uma listagem do código. Um modelo de relatório, que podem/devem adaptar de acordo com as vossas necessidades, será disponibilizado no sistema Fénix.

Projectos em atraso serão aceites durante a semana de 4 de Janeiro, sendo penalizados com **0,5 valores por cada dia de atraso**. A partir das **15:00** horas do dia **8 de Janeiro de 2010** não se aceitam quaisquer projectos, seja qual for o pretexto.

O relatório deve ser entregue dentro de uma capa ou encadernado, apresentando visivelmente o número do grupo e número e nome dos seus autores, na capa. Projectos que não sejam entregues nestas condições serão penalizados com três valores.

Para além disto, a submissão do código por via electrónica, através do sistema Fénix, é obrigatória e deverá ser feita nos mesmos prazos que a entrega do relatório. Se o código e o relatório forem entregues em dias diferentes, o desconto será o correspondente ao dia da última entrega.

O código do projecto deve estar contido num único ficheiro. Se durante o desenvolvimento usaram vários ficheiros devem, no final, antes de submeter o código, colocar todo o código num único ficheiro, chamado `FP0910-parte2-grupo<n>.scm`, em que `<n>` é o número do seu grupo. Por exemplo, o ficheiro do grupo número 5 deverá chamar-se `FP0910-parte2-grupo5.scm`. 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.

[7] Tal como na primeira parte do projecto, todos os comentários do seu programa devem ser feitos utilizando a opção "Comment Out with Semicolons". Programas que utilizem a opção "Comment Out with a Box" serão penalizados com três valores.

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

Projectos iguais, ou muito semelhantes, serão classificados com 0 (zero) valores. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projecto.

9 Resumo dos tipos de informação a utilizar

9.1 O tipo pino-chave

Os elementos do tipo *pino-chave* são um dos seguintes símbolos: `red`, `aqua`, `brown`, `orange`, `yellow`, `lime`, `white`[8].

O tipo *pino-chave* deve disponibilizar o seguinte reconhecedor:

- $pino\text{-}chave? : universal \mapsto lógico$

  $pino\text{-}chave?(arg)$ tem o valor *verdadeiro*, se arg é um *pino-chave*, e tem o valor *falso*, em caso contrário.

Para este tipo de informação, no seu programa, deverá também definir o procedimento que transforma elementos do tipo *pino-chave* para cadeias de caracteres:

$$pino\text{-}chave\text{->}string : pino\text{-}chave \mapsto string$$

$pinos\text{-}chave\text{->}string(p)$ devolve uma cadeia de caracteres construída a partir de p. Por exemplo, se p for um elemento do tipo *pino-chave* com o valor "`red`", então o valor de $pino\text{-}chave\text{->}string(p)$ é "`|pino-chave%red|`".

9.2 O tipo sequência de pinos chave

Uma sequência de pinos chave, ou *seq-pinos-chave*, é uma estrutura com 4 componentes do tipo *pino-chave*, em que a ordem é relevante. O tipo *seq-pinos-chave* deve disponibilizar as seguintes operações básicas:

1. Construtor

   - $cria\text{-}seq\text{-}pinos\text{-}chave : pino\text{-}chave \times pino\text{-}chave \times pino\text{-}chave \times pino\text{-}chave \mapsto seq\text{-}pinos\text{-}chave$

     $cria\text{-}seq\text{-}pinos\text{-}chave(p_1, p_2, p_3, p_4)$ tem como valor uma $seq\text{-}pinos\text{-}chave$ com os elementos p_1, p_2, p_3, p_4 e por esta ordem.

[8] É importante notar que um *pino-chave* com a cor `white` não pode aparecer como parte do segredo nem como de uma adivinha, esta cor é apenas utilizada pela IGM.

2. Selector

   - $elemento\text{-}i\text{-}seq\text{-}pinos\text{-}chave : seq\text{-}pinos\text{-}chave \times \{1,2,3,4\} \mapsto pino\text{-}chave$
     $elemento\text{-}i\text{-}seq\text{-}pinos\text{-}chave(s,i)$ tem como valor o elemento que se encontra na posição i de s.

3. Reconhecedor

   - $seq\text{-}pinos\text{-}chave? : universal \mapsto lógico$
     $seq\text{-}pinos\text{-}chave?(arg)$ tem o valor *verdadeiro*, se arg é uma *seq-pinos-chave*, e tem o valor *falso*, em caso contrário.

4. Teste

   - $seq\text{-}pinos\text{-}chave$=?: $seq\text{-}pinos\text{-}chave \times seq\text{-}pinos\text{-}chave \mapsto lógico$
     $seq\text{-}pinos\text{-}chave$=?$(s_1, s_2)$ tem o valor *verdadeiro*, se s_1 e s_2 são $seq\text{-}pinos\text{-}chave$ iguais, e tem o valor *falso*, em caso contrário.

Deve também definir o transformador do tipo *seq-pinos-chave* para cadeias de caracteres:

$$seq\text{-}pinos\text{-}chave\text{->}string : seq\text{-}pinos\text{-}chave \mapsto string$$

$seq\text{-}pinos\text{-}chave\text{->}string(s)$ devolve uma cadeia de caracteres construída a partir de s. Por exemplo, se a *seq-pinos-chave* contém os pinos de cor "red", "red", "aqua" e "red", o valor de $seq\text{-}pinos\text{-}chave\text{->}string(s)$ deve ser: "`|seq-pinos-chave%|pino-chave%red|%|pino-chave%red|%|pino-chave%aqua|%|pino-chave%red||`"

9.3 O tipo resposta

Uma *resposta* é constituída por dois inteiros, cada um deles entre 0 e 4, correspondentes, respectivamente, ao número de pinos chave certos na posição certa e ao número de pinos chave certos mas na posição errada. O tipo *resposta* deve disponibilizar as seguintes operações básicas:

1. Construtor

   - $faz\text{-}resposta : inteiro \times inteiro \mapsto resposta$
     $faz\text{-}resposta(p,v)$ devolve a resposta correspondente ao número de pinos chave certos na posição certa p e ao número de pinos chave certos mas na posição errada v.

2. Selectores

   - $resposta\text{-}pretos : resposta \mapsto inteiro$
     $resposta\text{-}pretos(r)$ devolve o número de pinos chave certos na posição certa de r.
   - $resposta\text{-}vermelhos : resposta \mapsto inteiro$
     $resposta\text{-}vermelhos(r)$ devolve o número de pinos chave certos na posição errada de r.

3. Reconhecedor
   - $resposta?$: $universal \mapsto lógico$
     $resposta?(arg)$ tem o valor *verdadeiro*, se arg é uma resposta, e tem o valor *falso*, em caso contrário.
4. Teste
   - $resposta$=?: $resposta \times resposta \mapsto lógico$
     $resposta$=?(r_1, r_2) tem o valor *verdadeiro*, se r_1 e r_2 são respostas iguais, e tem o valor *falso*, em caso contrário.

Deve também definir o transformador:

$$resposta\text{->}string : resposta \mapsto string$$

Por exemplo, se r é a $resposta$ obtida por $faz\text{-}resposta(3, 1)$, o valor de $resposta\text{->}string(r)$ deve ser "`|resposta%3%1|`".

9.4 O tipo jogada

Uma jogada é constituída por uma *seq-pinos-chave* e uma *resposta*. O tipo *jogada* deve disponibilizar as seguintes operações básicas:

1. Construtor
   - $faz\text{-}jogada$: $seq\text{-}pinos\text{-}chave \times resposta \mapsto jogada$
     $faz\text{-}jogada(s, r)$ devolve a jogada constituída pela $seq\text{-}pinos\text{-}chave$ s e a resposta r.
2. Selectores
   - $jogada\text{-}seq\text{-}pinos\text{-}chave$: $jogada \mapsto seq\text{-}pinos\text{-}chave$
     $jogada\text{-}seq\text{-}pinos\text{-}chave(j)$ devolve a $seq\text{-}pinos\text{-}chave$ da jogada j.
   - $jogada\text{-}resposta$: $jogada \mapsto resposta$
     $jogada\text{-}resposta(j)$ devolve a *resposta* da jogada j.
3. Reconhecedor
   - $jogada?$: $universal \mapsto lógico$
     $jogada?(arg)$ tem o valor verdadeiro, se arg é uma jogada, e tem o valor falso, em caso contrário.

Deve também definir o transformador:

$$jogada\text{->}string : jogada \mapsto string$$

Por exemplo, se j for a jogada que contém os pinos de cor "red", "red", "aqua" e "lime", tendo 4 pinos chave certos na posição certa e 0 pinos chave certos mas na posição errada, o valor de $jogada\text{->}string(j)$ deve ser "`|jogada%|seq-pinos-chave%|pino-chave%red|%|pino-chave%red|%|pino-chave%aqua|%|pino-chave%lime||%|resposta%4%0||`"

9.5 O tipo jogadas

O tipo $jogadas$ é constituído por uma sequência de 12 elementos do tipo *jogada*. Cada elemento é referenciado por um inteiro entre 1 e 12. O tipo *jogadas* deve disponibilizar as seguintes operações básicas:

1. Construtor
   - $jogadas(j) : jogada \mapsto jogadas$
     $jogadas(j)$ devolve um elemento do tipo *jogadas* em que todos os componentes são iguais e correspondem à jogada j.
2. Modificador
   - $altera\text{-}jogadas! : jogadas \times \{1,2,3,4,5,6,7,8,9,10,11,12\} \times jogada \mapsto jogadas$
     $altera\text{-}jogadas!(js, i, j)$ modifica o elemento do tipo *jogadas*, js, alterando o elemento na posição i, para j.
3. Selectores
   - $jogadas\text{-}i(js, i) : jogadas \times inteiro \mapsto jogada$
     $jogadas\text{-}i(js, i)$ devolve o componente da posição i de *jogadas* js.
4. Reconhecedores
   - $jogadas? : universal \mapsto lógico$
     $jogadas?(arg)$ tem o valor *verdadeiro*, se arg é do tipo jogadas, e tem o valor *falso*, em caso contrário.

Deve também definir o transformador:

$$jogadas\text{->}string : jogadas \mapsto string$$

o qual deve produzir uma cadeia de caracteres. Por exemplo, se js for do tipo *jogadas*, então o valor de $jogadas\text{->}string(js)$ tem o valor "`|jogadas%<jogada 1>%<jogada 2>% ... %<jogada 12>|`", em que `<jogada` i`>` ($1 \leq i \leq 12$) corresponde à cadeia de caracteres resultante da utilização do transformador $jogada\text{->}string$ a cada uma das jogadas em js.

Com este projecto, o
Instituto Superior Técnico
está a contribuir para que a futura
PS4 da YNOS seja um verdadeiro
sucesso comercial