Aula Prática 8

Objectivos

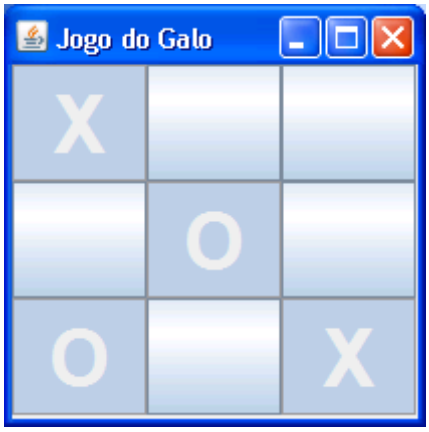
Introdução à utilização da biblioteca SWING
Utilização de Expressões Lambda

Problema 8.1

Implemente uma versão simples do chamado Jogo do Galo. Para esse fim considere que o programa apenas executa o jogo uma vez, começando com cruzes (`X`) ou bolas (`O`) consoante o argumento passado ao programa ("`java JogoDoGalo O`" ou "`java JogoDoGalo X`").

Sugere-se a utilização da classe `JToggleButton` para implementar cada casa e o `JPanel` com o *layout* `GridLayout` para dispr essas casa no tabuleiro do jogo.

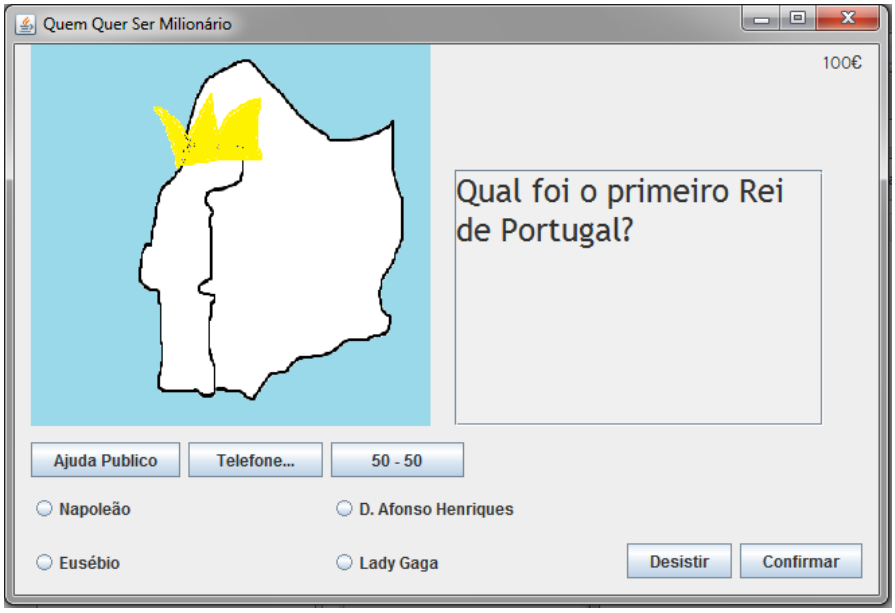
Nota: Na página da cadeira pode encontrar um ficheiro `jar` executável (`JogoDoGalo.jar`) onde pode verificar o comportamento desejado para o programa.



Problema 8.2

Implemente uma versão simples do jogo Quem Quer Ser Milionário. Para esse fim considere que o programa apenas executa o jogo uma vez, mostrando o resultado final ao jogador.

Na página da disciplina pode encontrar um ficheiro rar (QQSM.rar) que depois de descompactado contém um exemplo de programa e alguns ficheiros necessários para a execução do jogo. Comece por explorar as características do jogo demonstrativo.



a) Comece por compor os JComponents duma forma semelhante ao apresentado na figura de cima. Sugere-se a utilização dos JComponents:

- **JLabel:** Para tratar de pequenas etiquetas de texto ou imagens.
- **JButton:** Para programar os botões da aplicação.
- **JTextArea:** Para textos longos (e.g. a pergunta).
- **JRadioButton e ButtonGroup:** Para garantir que o jogador só seleciona uma opção.

Note, no entanto, que a composição gráfica do programa fica ao seu critério.

b) O programa deve carregar dum ficheiro a seguinte informação:

- *Path* das imagens
- Perguntas
- As quatro respostas possíveis para cada pergunta
- Níveis de dificuldade das perguntas
- Indicação da resposta correta a cada pergunta

Proponha e implemente um formato de ficheiro capaz de suportar toda esta informação.

c) No jogo Quem Quer Ser Milionário a dificuldade previsível aumenta, conforme o valor vai aumentando, começando pelo mais fácil, chegando ao mais difícil. A sequência de prémios é a seguinte:

25€	50€	125€	250€	500€
750€	1500€	2500€	5000€	10000€
16000€	32000€	64000€	125000€	250000€

O concorrente dispõe de quatro ajudas que poderá utilizar quando quiser, mas apenas

uma única vez durante o seu percurso, e que são:

- **50-50**: O computador elimina aleatoriamente duas respostas erradas, ficando apenas com a resposta certa e uma resposta errada.
- **Telefonar**: O concorrente telefona a alguém da sua escolha que possa saber a resposta.
- **Ajuda do público**: O auditório vota por meio de um dispositivo electrónico na resposta que achar correta.

No caso do nosso simulador de jogo, as ajudas Telefonar e Ajuda do Público devem gerar duas ou quatro respostas possíveis, respetivamente. Existe uma probabilidade associada a cada resposta possível e indexada à dificuldade da pergunta. Ou seja, quanto mais difícil a pergunta, mais dúbias deverão ser as alternativas. Utilize a classe `Random` para imprimir aleatoriedade.

Problema 8.3

Utilizando o trabalho realizado no guião 7 alínea 2, crie uma aplicação gráfica `BMPViewer` que implemente um visualizador de imagens em formato BMP (**não comprimido**). A aplicação deve responder aos seguintes requisitos:

a) Permitir navegar no sistema de ficheiros, mostrando apenas os ficheiros com extensão ".bmp".
b) Permitir a visualização gráfica de ficheiros BMP selecionados.
c) Ter um menu e/ou botões que permitam implementar todas as funcionalidades solicitadas no problema 7.2 (flip vertical/horizontal e redimensionamento). Os efeitos produzidos na imagem devem ser representados graficamente.
d) Permitir guardar o resultado das transformações da imagem no ficheiro original (Save) ou num ficheiro diferente do original (Save As).
e) O produto final deve ser disponibilizado com JAR.

Nota: Tendo como objectivo facilitar o processo de representação gráfica de um conjunto de pixels num componente gráfico, disponibilizamos uma subtipo de JPanel para o efeito:

```
class PanelImage extends JPanel{
      BufferedImage bi;

      /**
      * @param pixels - Byte Array with Pixels
      * @param w  - Image Width (columns)
      * @param h - Image Heigth (row)
      */
      PanelImage(byte[] pixels, int w, int h){
          bi = new BufferedImage(w, h, BufferedImage.TYPE_4BYTE_ABGR);
          bi.setRGB(0, 0, w, h, byteArrayToIntArray(pixels), 0, w);
          this.setPreferredSize(new Dimension(w, h));
      }

      public void paintComponent(Graphics g){
          g.drawImage(bi, 0, 0, this);
      }
```

```
    private int[] byteArrayToIntArray(byte[] arr){
        int[] ret = new int[arr.length / 4];
        for (int i = 0; i < ret.length; i++)
            ret [i] = (arr[i*4 + 3]<<24 | (arr[i*4 + 2]&0xff)<<16 |
                       (arr[i*4 + 1]&0xff)<<8 | (arr[i*4]&0xff));
        return ret;
    }
}
```

Exemplo de utilização:

```
JFrame f = new JFrame("...");
JPanel panel = new PanelImage(bmp.getPixelData(),
                              bmp.getWidth(), bmp.getHeight());
f.add(panel);
```