Projeto de Programação Multiparadigma [v1.0]
2020/2021 (2º semestre)

Enunciado

Um projeto de programação multiparadigma tira partido das vantagens de diferentes paradigmas de programação de forma a produzir uma solução mais adequada (p.e. desempenho, leitura/interpretação, manutenção, adequação). Uma divisão clássica consiste na utilização do paradigma de programação orientado aos eventos para o desenvolvimento da **camada de apresentação** (*User Interface*) e outro(s) paradigma(s) (funcional, orientado aos objetos, imperativo, lógico) para o desenvolvimento da **camada de negócio**. A **camada de dados** completa uma típica arquitetura de 3 camadas.

Aviso: O trabalho entregue deve corresponder ao esforço individual de cada grupo, e em nenhum caso deve ser copiado código que será entregue. A deteção de código copiado será realizada por software especializado bastante sofisticado. Casos de plágio óbvio serão penalizados com a anulação do projeto, o que implica a reprovação à Unidade Curricular (UC). Adicionalmente, a situação será reportada à Comissão Pedagógica da ISTA/Conselho Pedagógico do ISCTE-IUL. Serão penalizados da mesma forma tanto os alunos que fornecem código (se for o caso) como os que copiam código.

Grupos de Trabalho

O trabalho deverá ser realizado por grupos de **3 elementos** (exceções necessitam de aceitação explícita por parte do coordenador da UC).

Datas

- Entrega nº1: até 12 de abril (via e-learning) contendo a 1ª parte (no mínimo com duas das tarefas T1 – T5)
- Discussão nº1: durante uma das aulas da semana 19Abr – 23Abr
- Entrega nº2: até 7 de maio (via e-learning) contendo a 1ª parte (melhorada) e a 2ª parte
- Discussão nº2: durante uma das aulas da última semana de aulas (10-14Mai)

Introdução

O objetivo deste projeto é o desenvolvimento de competências de programação que permita desenvolver, em colaboração, aplicações multiparadigma interativas de média escala. O projeto divide-se em duas partes. A primeira parte corresponde ao desenvolvimento da camada de negócio utilizando, obrigatoriamente, a linguagem de programação **Scala** seguindo os princípios de programação funcional pura (exceções serão explicitamente indicadas). A segunda parte corresponde ao desenvolvimento da camada de apresentação (interativa) que deverá ser desenvolvida utilizando a linguagem de programação **JavaFX**. Apesar de poder ser necessária para o projeto, as características de implementação da camada de dados não serão consideradas em termos de avaliação.

Parte 1: Camada de Negócio

Objetivos: Desenvolver um conjunto de métodos, utilizando o paradigma de programação funcional, capazes de garantir os objetivos funcionais da solução proposta. Para além do código fonte (em Scala) é necessário desenvolver o diagrama de classes representando a arquitetura da solução proposta.

Parte 2: Camada de Apresentação Interativa

Objetivos: Desenvolver uma *Graphical User Interface* que permite uma interação fácil e intuitiva com a aplicação.

Requisitos

O projeto deverá, adicionalmente, satisfazer os seguintes requisitos:

Funcionais

1. Permitir executar com sucesso as várias funcionalidades;
2. Manter estado entre execuções.

Não funcionais

1. Apresentar ambas interfaces: gráfica (*Graphical User Interface – GUI*) e textual (*text-based User Interface*);
2. Os utilizadores deverão conseguir interagir corretamente (com o menor número de erros cometidos) e com facilidade (equilíbrio adequado entre quantidade de informação apresentada e número de ações necessárias para realizar uma tarefa) sem necessidade de treino.

Tema

Neste trabalho prático ocupar-nos-emos com a aplicação de *quadtrees* na representação e processamento de imagens e, na criação de um álbum de imagens.
Existem dois métodos essenciais para a representação de imagens a cores:

1. Como bitmaps, matrizes bidimensionais contendo em cada posição informação relativa a um pixel (poderá ser um booleano (bit) caso se trate de uma imagem a preto e branco; um valor inteiro caso se trate de uma imagem em *grayscale* ou uma lista de 3 inteiros no caso das imagens a cores).
2. Como mapas vetoriais, sendo então cada imagem descrita por uma lista dos objetos que a constituem (por exemplo, quadrado de cor azul de lado 30 e coordenadas (20, 10) para o vértice inferior esquerdo).

As *quadtrees* proporcionam um terceiro método: uma otimização dos bitmaps em que se representa de forma atómica informação relativa a todo um bloco (retangular) de pixels. A ideia é considerar a imagem (que se considera ter uma forma retangular) dividida em quatro quadrantes iguais; cada um destes é em si uma imagem retangular que se divide de igual forma. A imagem assim dividida corresponde a um nó da *quadtree*, em que se armazena informação geométrica, nomeadamente as coordenadas de dois vértices opostos. Os seus descendentes são (as árvores correspondentes a) os seus 4 quadrantes. Quadrantes (ou secções) constituídos apenas por pixels da mesma cor não precisam de ser divididos: basta representá-los por folhas da árvore, onde é guardada informação relativa à cor. Naturalmente, em imagens com grandes manchas da mesma cor, é possível poupar espaço considerável por comparação com a representação por bitmap. No caso limite, a secção mais pequena é um pixel.

Tarefas

Pretende-se escrever os seguintes métodos:

T1. `makeQTree(b:BitMap):QTree` criação de uma *quadtree* a partir de um bitmap fornecido e método oposto i.e. para transformar uma *quadtree* num bitmap;

T2. `scale(scale:Double, qt:QTree):QTree` operação de ampliação/redução de uma imagem, segundo o fator fornecido (por exemplo 1.5 ampliará a imagem aumentando ambos os seus lados em 50%);

T3. `mirrorV / mirrorH (qt:QTree):QTree` operações de espelhamento vertical e horizontal;

T4. `rotateD / rotateR (qt:QTree):QTree` operações de rotação de 90 graus nos dois sentidos;

T5. `mapColourEffect(f:Colour => Colour, qt:QTree):QTree` mapeamento uniforme de uma função em toda a imagem. Deverá utilizar este método para ilustrar a aplicação dos efeitos *Noise*, *Contrast* e *Sepia*.

e ser possível realizar as operações típicas (**T6**) de um álbum de imagens (i.e., adicionar, remover, percorrer, procurar, trocar ordem das imagens, editar informação associada) e permitir diferentes formas de visualização (**T7**) (p.e. "*slideshow"*, grelha, etc.). Incluir T7 somente na *GUI*.

Alguns Tipos de Dados a Utilizar

```
type Point = (Int, Int)
type Coords = (Point, Point)
type Section = (Coords, Color)
trait QTree[+A]
case class QNode[A](value: A,
                    one: QTree[A], two: QTree[A],
                    three: QTree[A], four: QTree[A]) extends QTree[A]
case class QLeaf[A, B](value: B) extends QTree[A]
case object QEmpty extends QTree[Nothing]
```

A cada nó e cada secção (folha) de uma árvore estão associadas duas coordenadas, dos vértices superior esquerdo e inferior direito. Observe-se que a presença do *QEmpty* se deve à eventual necessidade de dividir imagens com apenas dois pixels em quadrantes; neste caso dois dos quadrantes serão vazios.

Material Fornecido

- *ImageUtil.java* – contendo métodos utilitários para lidar com imagens e cores (útil para ilustrar a transparência de integração destes dois paradigmas). Único código Java do projeto juntamente com uso do `java.io.FileInputStream;`
- *objc2_2*.png – imagem com 4 pixéis (dimensão 2 por 2) que pode ser utilizada para os testes iniciais. O método `readColorImage` da classe *ImageUtil.java* pode ser utilizado para transformar a imagem em `Array[Array[Int]]` e, posteriormente, transformado em `List[List[Int]]` antes de ser transformada numa `QTree`. O método `writeImage` da classe *ImageUtil.java* pode ser utilizado para gravar a matriz de *Int* de volta ao formato *png*;
- *info.txt* – contendo um exemplo de uma `QTree[Coords]`.

Exemplo de Divisão de uma Imagem em Quadrantes

A raiz da árvore representa a imagem representada via *quadtree*.

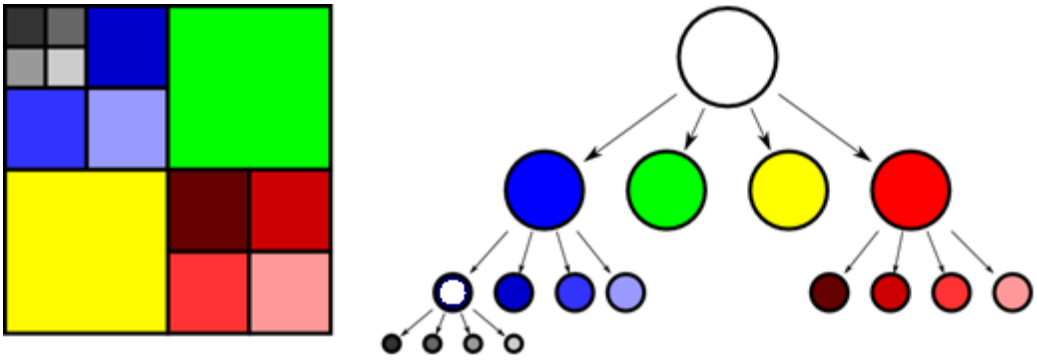

Figura 1 – Representação gráfica e respetiva representação em árvore simplificada de um quadtree

Avaliação

A realização do projeto é obrigatória para obter aprovação à UC. Não haverá qualquer possibilidade de obter aprovação à UC sem realizar o projeto. O projeto é composto por duas partes e realizado em grupo, no entanto, as discussões e aferições são individuais. As classificações possíveis na aferição individual são A, B, C ou D, definindo a nota final (*nf*):

- A - *nf* igual à classificação do projeto;
- B - *nf* igual a 85% da classificação do projeto;
- C - *nf* igual a 70% da classificação do projeto;
- D - reprovação à UC.

A **primeira parte** do projeto (com peso de **75%**) será inicialmente avaliado em termos funcionais (i.e., se os métodos produzem os resultados esperados) e se cumpre os requisitos funcionais (para ser usado numa primeira fase via *text-based User Interface* – a visualização das imagens neste caso será efetuada através de um *viewer* externo). A solução desenvolvida deverá, obrigatoriamente, aplicar as matérias de programação funcional introduzidos na UC, nomeadamente:

- Recursividade (no lugar de iteratividade) e *Pattern matching*;
- Imutabilidade (exceto na camada interativa);
- *Pure functions* (exceto na camada interativa);
- *Tail recursion*;
- Funções de ordem superior;
- Padrões recorrentes sobre listas (i.e. *map*, *filter* e *fold*);
- *Functional Object-Oriented programming*;
- *Partially Applied Functions* ou *Currying;*
- *Try* e *Option;*

Ainda que a funcionalidade exigida tenha sido concretizada, a não aplicação das matérias mencionadas pode fazer baixar a classificação. A avaliação também poderá ser penalizada em função da falta de qualidade do código (p.e. difícil manutenção, baixa eficiência, dificuldade de interpretação, duplicação de código, complexidade desnecessária).

A **segunda parte** do projeto (com peso de **25%**) será avaliada em termos da qualidade e usabilidade das interfaces (principalmente da GUI) desenvolvidas, do cumprimento dos requisitos e da correta aplicação da matéria sobre engenharia de sistemas interativos (*Event-driven Programming* e *UI patterns* e *Tree Testing*). A avaliação da GUI via *Tree Testing* deve ser realizada com 3 tarefas e 6 participantes[1].

Os alunos poderão obter feedback juntos dos professores das respetivas turmas sobre o progresso do projeto, e deverão seguir as recomendações dadas.

[1] Os elementos de cada grupo devem participar no *Tree Testing* de pelo menos dois grupos

Os projetos são classificados de acordo com os seguintes pesos:

1ª parte (75%):

- Requisitos funcionais – 35%
- Correta e completa aplicação da matéria de Programação Funcional – 25%
- Qualidade do código – 10%
- Diagramas de classes – 5%

2ª parte (25%):

- Qualidade da GUI desenvolvida – 15%
- Correta aplicação da matéria de sistemas interativos – 10%

Entrega

Submissões: ambas as entregas deverão ser, obrigatoriamente, realizadas por cada grupo através do e-learning.

Dados e formatação:

- Projeto com todo o código fonte desenvolvido em formato *Zip* ("File/Export/Project to Zip File…") **contendo imagens e dados para poder testá-lo**. O nome do projeto deve incluir o **número do grupo e nome dos vários membros** (*NúmeroGrupo_Nome1_Nome2_Nome3*).
  Sugestão: verifiquem que a importação do vosso projeto numa máquina diferente ocorre com sucesso antes de efetuarem a submissão;
- Diagrama de classes (ficheiro: Diagrama.png);
- Resultados do *Tree Testing* (ficheiro: Link_TreeTesting.txt) contendo o link para os resultados obtidos (optimalworkshop link);
- Possíveis comentários (ficheiro: Readme.txt).

Os 3 ficheiros (Diagrama, Link_TreeTesting e Readme) devem ser adicionados à raiz do projeto.

Discussão: a discussão é individual e todos os alunos terão de demonstrar ser capazes de fazer um trabalho com o nível de qualidade igual ao que assinaram. Ao entregar o trabalho, os alunos implicitamente afirmam que são os únicos responsáveis pelo código entregue e que todos os membros do grupo participaram de forma equilibrada na sua execução, tendo todos adquirido os conhecimentos necessários para produzir um trabalho do mesmo tipo individualmente.