Instituto Superior de Engenharia de Lisboa

PROGRAMAÇÃO ORIENTADA POR OBJETOS

Relatório do Terceiro Trabalho Prático

Docente

Prof. José Simão

Alunos

43552 Samuel Costa

43884 Pedro Rocha

Junho 2017

Índice

Introdução

Este relatório, referente ao terceiro de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Programação Orientada por Objetos e acompanha a implementação em Android do jogo, designado *Circuit*, desenvolvido no âmbito do primeiro trabalho prático.

O trabalho desenvolvido no âmbito do primeiro trabalho prático constituiu importante ponto de partida para a realização do actual trabalho.

Ao longo do documento apresentam-se os principais objetivos metodológicos e práticos delineados pelo problema; a metodologia adotada; uma explicação dos procedimentos aplicados no desenvolvimento da aplicação; e, por fim, uma breve reflexão sobre o trabalho realizado em que se discutem os resultados obtidos, explicitando-se, conclusões referentes ao trabalho desenvolvido.

Objetivos

O trabalho prático visa, em termos gerais, aplicar as ferramentas introduzidas em contexto de sala de aula, em particular, o padrão de desenvolvimento **MVC**, bem como alguns aspetos da programação orientada por objetos, os conceitos de herança e polimorfismo, aprofundando as suas potencialidades de utilização, através do desenvolvimento de uma aplicação designada *Circuit,* que permite desenhar as pistas que fazem as ligações entre terminais da mesma cor de um circuito impresso.

O trabalho também visa aprofundar alguns aspetos do desenvolvimento em Android, em particular a hierarquia de classes Activity, a utilização de *listeners*, a utilização de objetos da classe *Intent* e *Bundle*, e de widgets.

Definem-se os seguintes objetivos:

- do ponto de vista metodológico: identificar os procedimentos a aplicar no desenvolvimento da aplicação seguindo a abordagem model-view-controller (**MVC**);
- do ponto de vista prático, empregar e dominar os seguintes instrumentos da linguagem java: (i) respeitar a sintaxe do JAVA e as orientações de conceção de classes; (ii) fazer a utilização adequada de tipos enumerados; (iii) reconhecer a diferença entre estado e comportamento de um objeto; (iv) conceber classes base e classes derivadas; e por último (v) compreender e utilizar convenientemente os serviços fornecidos por uma hierarquia de classes.

Métodos e Recursos utilizados

Para a realização deste trabalho foi utilizada a linguagem *JavaTM* Versão 8 *Update* 131. Foi também utilizado o ambiente de desenvolvimento *Android Studio* versão 2.3.3, com o *plugin PlantUML Integration* Versão 2.8.0.

Todos os elementos da linguagem necessários para a resolução dos problemas propostos foram adquiridos nas aulas da unidade curricular Programação Orientada por Objetos (POO).

Regras do Jogo

O objetivo do jogo é desenhar pistas de um circuito impresso, ligando entre si terminais da mesma cor com pistas. Cada posição do circuito não pode ser ocupada por mais de uma pista e algumas posições no circuito podem ter restrições, isto é, apenas podem ser preenchidas por pista verticais ou horizontais.

O jogo é composto por níveis já compostos, e é possível passar para o nível seguinte, sempre que o jogador tenha completado o nível que está a jogar.

Considera-se o nível completado, quando todas as posições do circuito estão preenchidas com pistas que ligam os terminais da mesma cor entre si e todos os terminais estão preenchidos. Ressalva-se que cada terminal só pode ter uma ligação.

O jogo acaba quando o jogador desiste ou não houverem mais níveis para serem carregados.

MVC (Model-view-controller)

MVC é um padrão de arquitetura, que permite conceber o desenvolvimento de aplicações reduzindo a complexidade, favorecendo a manutenção e a facilidade de leitura, por redução do projeto a três componentes funcionais da aplicação:

Model – (modelo), é um repositório da informação necessária para a aplicação (ou parte dela), mantém constantemente atualizada toda a informação, que poderá passar a outros componentes, ou apenas aguardar que outros componentes lhe peçam informação.

View – (vista), é responsável pela representação gráfica do modelo.

Controller – (controlador), efetua a ligação entre Model e View nas interações do sistema com o exterior, controlando as alterações ao modelo, reagindo a essas alterações, e instruindo a vista para as representar.

Na figura 1 é apresentado um diagrama, baseado na linguagem UML- Unified Modeling Language, que explicita a relação entre as diversas classes que fazem parte do programa desenvolvido, evidenciando essas três componentes da aplicação identificadas durante o desenvolvimento:

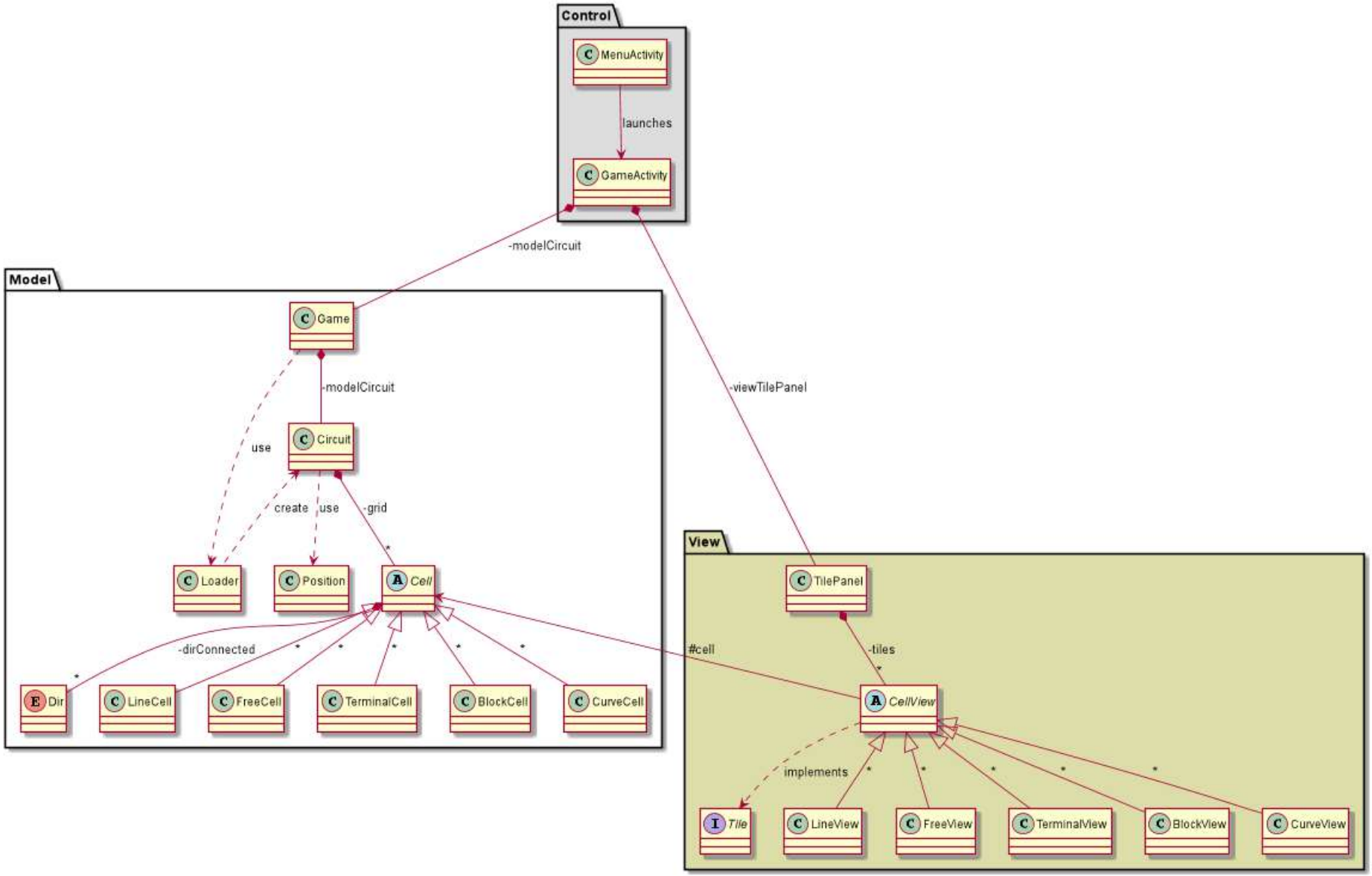

Figura 1 - Diagrama de Classes UML do programa desenvolvido

Procedimentos Aplicados

Como este relatório se fará acompanhar do código desenvolvido para a implementação do projeto, esta secção dará ênfase à descrição dos serviços adicionais disponibilizados nas diferentes classes e do seu contributo para a execução da aplicação, isto é, não são feitas referências explicitas ao código escrito, salvo em duas ocasiões, como se verá, na classe Circuit.

A sequência em que é feita a descrição pretende facilitar a correspondência entre a composição do projeto e a execução do programa.

Controller

Quando a aplicação é lançada, são os métodos *onStart* e *onCreate* de *MenuActivity* que são executados. Esta activity é composta por quatro botões, cada um correspondente a um nível do jogo. Quando é pressionado o botão com o texto *level* (nível) *n*, é começada uma *GameActivity,* e carregado o nível *n.* Para isso, o ficheiro layout de *MenuActivity* inclui uma *tag* em cada botão, como uma representação Unicode do número de nível correspondente. Foi também implementado o método *startNewGameFromLevel(Activity act , int level)* em GameActivity, que começa uma *GameActivity* passando um *Intent* a que foi previamente adicionada uma chave para o nível que se pretende carregar. Como foi definido um listener para todos os botões da activity, a redefinição do método *onClick* para esse listener chama o método *startNewGameFromLevel* passando-lhe o contexto e número do botão obtido a partir da tag, depois de convertido para inteiro.

Apesar da classe *MenuActivity* ser executada quando a aplicação é lançada, pode dizer-se que o núcleo da componente Controller do projeto é *GameActivity*. Esta classe gere o cumprimento das regras do jogo, traduzindo-as para a linguagem de programação (ver “Regras do jogo” no capítulo anterior).

O método onCreate de *GameActivity* instancia um objeto do tipo Game passando como parâmetro os assets para carregar o circuito do nível a partir de um ficheiro, regista os *listeners* para o botão “próximo nível”, para eventos de toque no ecrã e para auxiliar a contagem de tempo.

Este método pode ser invocado em três circunstâncias diferentes: o jogador inicia o jogo no nível 1, o jogo está a ser carregado de um estado guardado, ou o jogo está a ser iniciado num nível superior.

Se há um jogo gravado para carregar e há uma chave para o nível, isto é, a orientação do ecrã mudou, é chamado o método *startGame* de Game, passando como parâmetro o nível de jogo, o tempo de jogo e a informação codificada numa String relativa ao estado de jogo.

Se o nível não está a recomeçar de um estado previamente gravado, ie, o jogador carregou num dos botões de MenuActivity, é carregado o nível a partir de Intent. É carregado o TilePanel correspondente ao circuito gerado.

Foram criados métodos auxiliares para afetar o modelo com as interações do jogador, e esses métodos são chamados quando o evento para que o listener foi registado ocorre.

Model

a) Loader

É por recurso a um objeto da classe Loader que cada nível é carregado, como ficou explicado acima. O método load(int) retorna uma referência para um objeto do tipo Circuit referente ao circuito do nível passado como parâmetro. Este método lança uma exceção do tipo LevelFormatException se o carregamento não tiver sido efetuado com sucesso, sendo mostrada ao utilizador uma mensagem especificando o erro, e em alguns casos, em que linha do ficheiro de leitura foi encontrado o erro.

b) Cell

Com a classe abstrata Cell pretende-se que todas as células de tipos derivados de Cell (FreeCell, LineCell, TerminalCell, BlockCell e CurveCell) tenham um comportamento similar que será detalhado em cada implementação:

IsFull() – retorna indicação de que a célula esta ou não cheia, útil para saber se o nível esta preenchido e para evitar mais conexões com outras células.

canConnectFrom() – retorna indicação de ser possível ou não a célula ser conectada de um determinado terminal e de uma determinada direção.

canDisconnetfrom() – retorna indicação de ser possível ou não a célula ser desconectável de um determinado terminal e de uma determinada direção.

ConnectFrom(), disconnecFrom() – efetua a conexão/desconexão de uma terminal e direção.

fromSaved(char saved) – descodifica o estado da célula a partir de um byte.

toSave () – retorna um caracter. Utiliza um byte para codificar o estado da célula. Os quatro bits de menor peso são a chave para a cor da seleção, os quatro bits de maior peso representam as direções em que está ligada ou não (baixo,cima,esquerda,direita do bit de maior significância para o de menor peso.

c) Dir

No tipo enumerado Dir, que representa as direções de conexão possíveis, foram implementados os métodos de apoio:

deltaLin() – devolve a variação de posição de linha da célula no circuito em função da direção de conexão em relação ao centro da célula, isto é, se a célula tem uma conexão com a direção UP, implica que a célula conectada a si esta na posição linha+ deltaLin (linha + -1).

deltaCol() – O mesmo que deltaLin() mas para colunas.

complement() – devolve a direção contrária.

d) Position

Na classe Position, que representa uma determinada posição de Cell no Circuito, foram implementados alguns métodos de apoio:

equals(Position) – devolve true se o campo line e col são iguais entre duas posições.

newPositionFromDirection(Position, Dir) – devolve uma nova posição tendo por referência a posição atual e uma direção.

e) Circuit

A classe Circuit agrega um número de serviços disponíveis do package Model para a camada superior de abstração (classe Game). Foram implementados alguns métodos de apoio:

isOver() - devolve indicação de que o circuito esta totalmente preenchido (nível completo)

drag() - método invocado por Game após ação sobre o ecrã drag, que recebe duas posições. Se a célula de onde se pretende ligar estiver preenchida, determina a posição relativa das duas células identificadas pelas posições passadas como parâmetro. Se a posição for contígua, avalia se a célula de destino tem capacidade para ser ligada e nesse caso efetua a ligação entre as duas células nas direções correspondentes. Caso contrário, desliga as duas células. A listagem 1 apresenta parte do código escrito neste método para avaliar se as duas posições são contíguas e qual a posição da célula de destino em relação à célula de origem. Pensou-se que este trecho de código poderia ser implementado como um método relativePosition(Position) na classe Position com tipo de retorno Dir. Porém, uma vez que esta é a única ocorrência deste trecho de código e o método em que está inserido só é chamado num lugar do programa, optou-se por não fazer esta mudança.

```
        ...
if (fillFrom>0 && !cellFrom.isFull()) {
    int dif=0;
    Dir dirFrom = null;
    if (from.getLine()==to.getLine()) {
        dif=(from.getCol()-to.getCol());
        if (dif==1) dirFrom = Dir.LEFT ;
        else if (dif==-1) dirFrom = Dir.RIGHT;
        else dif=0;
    } else if (from.getCol()==from.getCol()){
        dif=(from.getLine()-to.getLine());
        if (dif==-1) dirFrom=Dir.DOWN;
        else if (dif==1) dirFrom=Dir.UP;
        else dif=0;
    }
...
```

Listagem 1 – parte da implementação de drag(Position, Position);

connectCells e disconnectCells – são dois métodos chamados pelo método drag que tentam conectar ou desconectar as células e devolve verdadeiro se foi possível.

unlink – método que é invocado por Game após término de toque sobre o ecrã. Desconecta a célula na posição passada como parâmetro e verifica se não ficam pistas no ar, isto é, verifica se as pistas estão ligadas ou não a um terminal. Se não estiverem ligadas a um terminal, desconecta toda essa pista.

disconnectIfLastNotTerminal – Recebe como parâmetro o caracter com que a célula de origem está preenchida, a posição da célula de origem na grelha, a direção em que se quer desconectar a célula de origem. Se a célula for um terminal devolve imediatamente falso. Em seguida, obtém, para cada direção em que a célula de origem está conectada, a posição da célula adjacente nessa direção. É chamado recursivamente este método para a nova posição, com o caracter de preenchimento e a direção complementar à direção em avaliação. Se retornar verdadeiro, são desconetadas as duas células envolvidas. Se a chamada recursiva retornar falso,

o método retorna falso. Se a célula de origem não estiver ligada em mais nenhuma direção, retorna verdadeiro. A listagem 2 apresenta em detalhe a implementação deste método.

```
private boolean disconnectIfLastNotTerminal(char fillFrom,Position
pos,Dir dirFrom){
    Cell cellFrom=this.getCellFromPosition(pos);

    if (!(cellFrom instanceof TerminalCell)) {
        for (Dir d : cellFrom.dirConnected) {
            if (d != Dir.EMPTY && d != dirFrom) {
                Position newPos = pos.newPositionFromDirection(d);
                if
(this.disconnectIfLastNotTerminal(fillFrom,newPos,d.complement())) {
this.getCellFromPosition(newPos).disconnectFrom(fillFrom,d.complement());
                    cellFrom.disconnectFrom(fillFrom,d);
                    return true;
                }
                return false;
            }
        }
        return true;
    }
    return false;
}
```

Listagem 2 – implementação do método disconnectIfLastNotTerminal(char,Position,Dir);

f) Game

A classe Game representa a camada mais alta de abstração do modelo do jogo. Guarda informação relativa a tempo de jogo, nível, e o estado do modelo de jogo ativo. Replica alguns serviços de Circuit a fim de disponibilizá-los para o exterior, grava o estado de jogo numa String separando a informação relativa a cada linha do tabuleiro por ";" e carrega, por recurso a um objeto da classe Loader, o circuito correspondente ao nível de jogo.

View

A classe TilePanel é parte de uma biblioteca fornecida.

GameActivity regista listener que implementa interface onTileListener, a ser utilizado por TilePanel.

À hierarquia de classes CellView, foram acrescentadas duas classes, que derivam de CellView, que espelham em termos de nome as alterações feitas ao modelo. CellView guarda uma referência para a célula que representa. O método onDraw desenha padrões geométricos específicos para cada tipo de célula, fazendo representar as ligações por coloração.

Conclusões

Com a realização deste trabalho, foi possível estruturar o desenvolvimento em torno de três componentes de um projeto. Constatou-se que a abordagem MVC torna o programa mais flexível, modular e incentiva à procura pela generalidade.

Em suma, este trabalho mostrou-se um bom exercício de aplicação e entendimento da estrutura de um programa utilizando noções do paradigma da programação orientada por objetos, e a aplicação de um padrão de desenvolvimento que se afigura proveitoso.