Programação Orientada por Objetos

Covid

Relatório do 2º Trabalho Prático

N. 45827 Nome Daniel Azevedo

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2019/2020

12/07/2020

Introdução

O trabalho consiste em desenvolver uma aplicação para Android. O objetivo era implementar um jogo em que a enfermeira heroína tem que empurrar todos os vírus para os caixotes de lixo.

Para este trabalho foi aplicado o Modelo MVC (Model, View, Control). O modelo representa a informação e nada mais. O modelo não deve depender do controlador.
A vista manifesta a informação do modelo, e envia *feedback* da interação com o utilizador. A vista poderá ser independente das outras estruturas, ou ser o controlador e assim ficar dependente do modelo.
O controlador fornece o modelo à vista, e interpreta as ações do utilizador (por exemplo MotionEvents).
Este depende tanto da vista como do modelo. Em alguns casos o controlador e a vista são o mesmo objeto.

Para a realização deste trabalho foi-nos fornecido pelo docente:

-Algumas imagens;
-Package Tile;
-Ficheiro texto (com 2 níveis do jogo);
-Classe Loader (para facilitar o carregamento dos níveis);

Controlador

O controlador aqui denominado por GamesControl é responsável por iniciar a activity, e instanciar diversas classes necessárias para o uso da mesma, incluindo o layout da aplicação (xml). Nesta classe tenho os botões, as diferentes representações de tiles (como campos), diversos listeners (botões, movimento do vírus e herói, entre outros), e diferentes métodos dependentes do modelo (salvar, guardar, carregar, acabar o jogo…).

A alteração da representação do jogo ao longo do tempo é feita maioritariamente pelos dos métodos onHeroMoved e onCovidMoved. onHeroMoved têm 3 desfechos, derivados da direção recebida como input do utilizador.

-Caso o Tile que se encontre na posição correspondente à direção seja emptyView (vazio), o herói é repintado na nova posição, e na posição anterior do herói substituo por um Tile emptyView (usando o método setTile de TilePanel);

-Na hipótese de o Tile encontrado ser um virusView (representação do vírus), tenho que ter em consideração o seu "movimento" em conjunto com o herói. Para este efeito, verifico o Tile a seguir ao vírus, caso seja emptyView repinto o vírus na terceira posição, o herói na segunda e na primeira uso um emptyView;

-Por último verifico se o Tile inferior ao herói é do tipo TrashTile, o que por sua vez origina a sua morte, e que repinte o tile deste por um deadView (imagem de uma caveira).

Dois dos desfechos usufruem do método swap da Classe board, responsável por desencadear a troca de posições no modelo, atualizando desta forma o modelo em conjunto com a vista.
Na hipótese de um vírus ser movimentado, percorro um lista que contém os vírus referente a cada nível (carregada ao iniciar novo nível), de forma a identificar o vírus correspondente à posição onde este encontrava-se antes de ser substituído pela representação do herói. Ao encontrar acesso ao método setCoordinates do objeto Virus, e atualizo as suas coordenadas (incoerência com o modelo MVC!).

O método onCovidMoved verifica duas possíveis situações: caso as coordenadas x e y sejam iguais a -1, vou repintar o Tile virusView nas coordenadas Dx, Dy por um objeto empty. Na outra situação verifico apenas o Tile do vírus referente as coordenadas passadas, e se for vazio, substitui-o por um vírus, e repinto a posição anterior deste por um Tile vazio. De forma a simular o vírus a cair, como acontece com o herói. Usufrui-o do HearBeatListener implementado em TilePanel, para de x em x tempo forçar a descida (ao chamar o método moveHero em Board) tanto do herói como dos vírus, isto claro, se o tile na posição abaixo o admitir.

Modelo

O modelo é composto por todas as peças do tabuleiro, e cada uma destas estende da classe abstrata Piece. Esta implementação permitiu construir um tabuleiro no modelo com os diferentes tipos de peças. A classe Piece apenas têm um método abstrato, getLetter (que retorna o caractere da peça em específico). Objetos do tipo Hero e Virus por exemplo, já existia a necessidade de ter acesso às suas coordenadas, de forma a poder movimentá-los pelo tabuleiro. Para o efeito defini *getters* e *setters* para garantir um acesso mais controlado à informação contida nestas classes. Deverá haver uma melhor forma de elaborar a inicialização e atualização da class Board, mas não me surgiu outra ideia para o fazer. Uma que permitisse alterar o tabuleiro sem criar um novo e perder os *listeners* inicializados no controlador (anulando o movimento das peças movíveis), e ao mesmo tempo não comprometesse a informação da mesma.

Na classe Board também temos os métodos responsáveis pela alteração das posições das peças do tabuleiro: -moveHero e moveCovid verificam a possibilidade de efetuar um movimento. moveHero em conjunto com moveVerifier implementam regras (não passar as dimensões do tabuleiro, e não permitir movimento a partir das peças Wall por exemplo), algumas das quais especificadas anteriormente no controlador. Em caso de se verificar as condições, permito a mudança da parte visual no controlador.

Vista

Nesta estrutura defini que todas a classes associadas a uma peça estendiam de uma superclasse ImgTile, a qual por sua vez também estende de MasterTile (implementa a interface Tile, o que obriga a esta ou alguma das suas subclasses a redefinir os seus métodos). As subclasses de ImgTile herdam assim os métodos draw, setSelect (não usado), e um Paint. Todas elas para serem instanciadas têm que chamar o construtor da sua classe mãe, através da chave super, e enviar como parâmetro o contexto, e identificador ambos recebidos do controlador para o construtor da classe Img fornecida pelo docente. Esta vai criar a imagem transmitida para as dimensões adequadas do quadriculado do tabuleiro. Grande parte da funcionalidade da Vista está no controlador, em que alguns dos métodos já fornecidos invalidam a vista por mim, e apenas é necessário enviar as respetivas coordenadas e o tipo de tile a ser usado para repintar aquela zona.