

**Programação Orientada por Objetos   
3º trabalho**

|  |  |
| --- | --- |
|  |  |
|  |  |

**Implementação do jogo “Snake” em Android**

**Grupo 12 – Noite**

Nº 36070 Vítor Manuel Bairrada Mexia

Nº 33297 Vítor Manuel Ferreira Varagial

Nº 44490 Cristiano Emanuel Garcia Morgado

Licenciatura em Engenharia Informática e de Computadores  
Semestre de Inverno 2018/2019

14/01/2019

Conteúdos

[1. Introdução 3](#_Toc535253514)

[2. Implementação 3](#_Toc535253515)

[2.1 - Descrição do Modelo: 4](#_Toc535253516)

[2.2 - Descrição do Controlo (Android): 5](#_Toc535253517)

[2.2.1 – MainActivity 5](#_Toc535253518)

[2.2.2 – DefeatActivity 7](#_Toc535253519)

[2.2.3 – VictoryActivity 8](#_Toc535253520)

[3. Desenvolvimentos futuros 9](#_Toc535253521)

[4. Conclusão 10](#_Toc535253522)

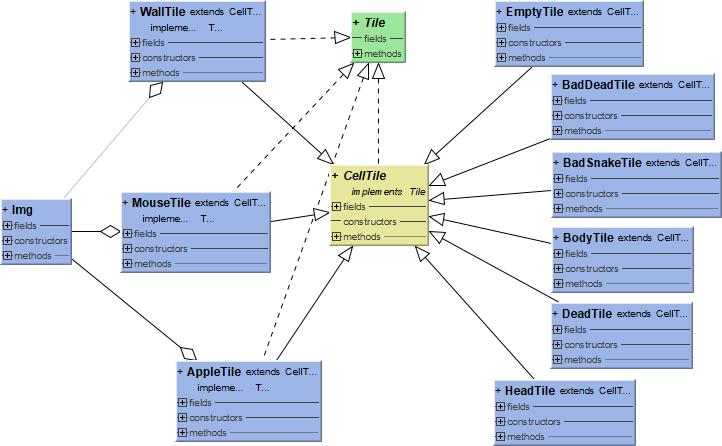
# Introdução

Pretendemos com este trabalho usar o modelo da aplicação “Snake” criada no 1º trabalho e com mínimas alterações criar o mesmo jogo em Android. A migração do jogo para a plataforma Android poderia ter sido feita sem qualquer modificação à lógica interna do jogo mas, no entanto, a mesma acabou por ser alvo de algumas alterações numa tentativa de optimizarmos o código e aplicar conhecimentos que adquirimos ao longo do semestre.

# Implementação

Depois de fazermos algumas melhorias ao modelo do primeiro trabalho criámos um projeto com uma *Empty Activity*, importámos as classes e *Resources* fornecidas assim como o modelo do trabalho1. Copiámos também a estrutura do *CellTile* e todas as classes estendidas*.*

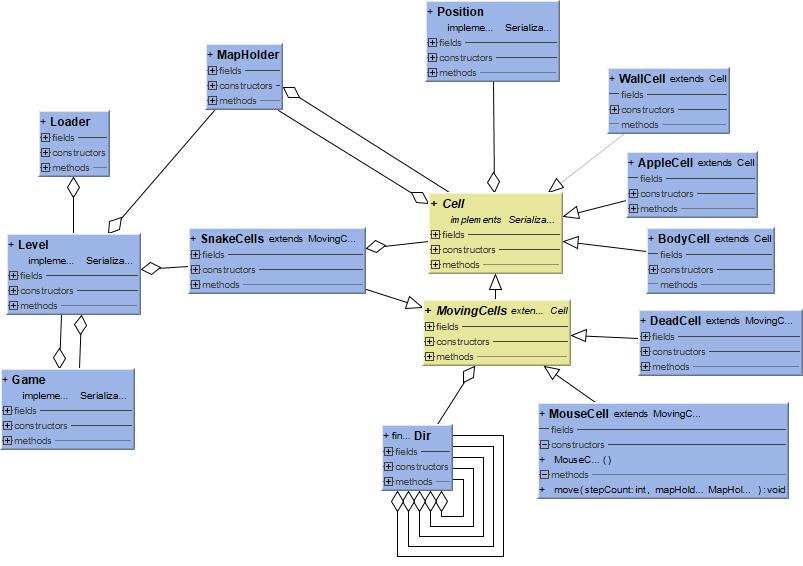
Começámos por implementar a classe *Tile* em *CellTile* implementandos os métodos *draw*  e *setSelected* para cada um dos cellTiles, usando para o *MouseTile*, o *WallTile* e *AppleTile* a classe *Img* fornecida para facilitar a apresentação das imagens fornecidas.



Criámos a estrutura do *activity\_main* usando o editor xml. Criámos um TilePanel usando o resource *stylable* fornecido. Para criarmos os *status panels*. Cada *Status Panel* é composto por 3 *TextView’s*, um exterior sem texto, um para o texto e outro para os números (Apples, Score and Level).

## 2.1 - Descrição do Modelo:

* **Class Level** – responsável pela criação do mapa de jogo e da inserção inicial, durante o setup do jogo, das várias células que constituem o nível. O Level define a interface Observer, que define métodos a ser utilizados para informar a interface gráfica sobre atualizações à área de jogo;
* **Class Position** - Esta classe tem como única função guardar os valores inteiros que representam uma posição no mapa (line, column) e disponibiliza métodos de get e set para cada um deles.
* **Class Cell** - A classe Cell representa os blocos que compõe o mapa de jogo. A classe **Cell** é estendida por um conjunto de classes cuja única função é distinguir-se pelo seu tipo para a lógica interna do modelo e também para o viewer e sem qualquer campo ou método adicional de interesse, e como tal não iremos entrar em detalhe sobre as mesmas, listando-as apenas:
  + **MovingCells** – contem todos os métodos em comum que se mexem ou têm ligação a estas
    - **SnakeCells** – representa a cabeça das cobras e é compostas de BodyCells (utilizador e badsnakes). Implementa a lógica comum a todas as cobras de jogo.
      * **PlayerCell –** Implementação específica da cobra controlada pelo jogador.
      * **EnemyCell –** Implementação das cobras inimigas e toda a sua lógica interna.
    - **MouseCells** – representa o rato
    - **DeadCells** – represente as cabeças de cobras mortas
  + **WallCell** – Representação de uma célula de parede\obstáculo
  + **BodylCell** – Representação de uma célula de cauda da Snake
  + **AppleCell** – representa as maçãs
* **MapHolder –** guarda a matriz de jogo e as células livres assim como gera posições random, direcções baseadas numa posição e posições livres
* **Game –** gere o score e carrega o nível seguinte
* **Loader –** Lê o ficheiro ‘pede’ a criação das cells correspondentes aos símbolos no ficheiro.



## 2.2 - Descrição do Controlo (Android):

Nesta fase do projeto o controlo da aplicação em Android é realizado na MainActivity que é, como descrito anteriormente, a actividade que controla a exibição do jogo. Para a entrega final, no dia da discussão do projeto, planeamos ter uma nova Activity inicial que irá funcionar como menu operacional do jogo onde se poderá escolher visualizar a tabela de *high scores* tal como iniciar um novo jogo e com a opção de escolher começar a partir de um nível que não o primeiro, desde que todos os níveis posteriores já tenham sido vencidos.

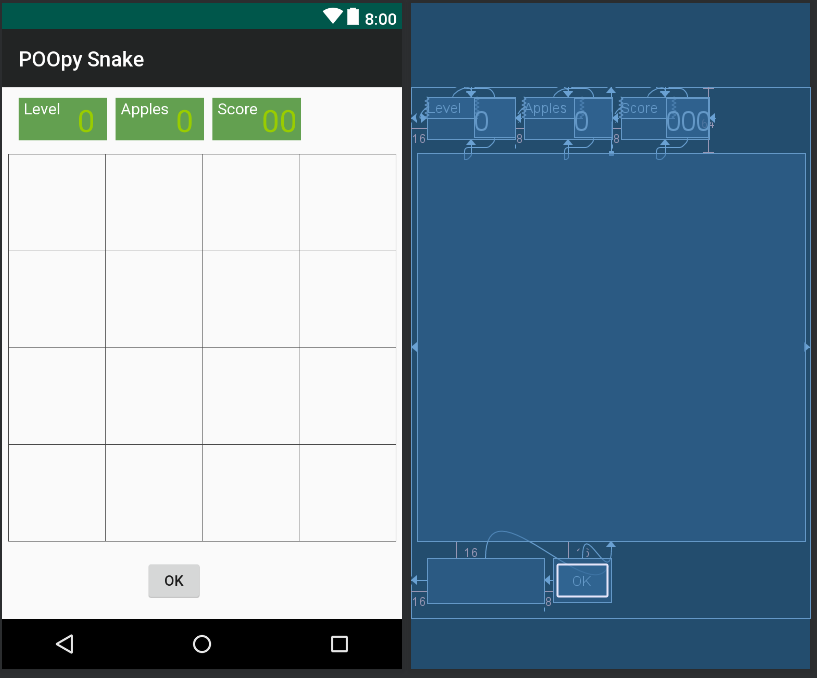
Neste capítulo deixamos uma explicação das actividades já implementadas.

### 2.2.1 – MainActivity

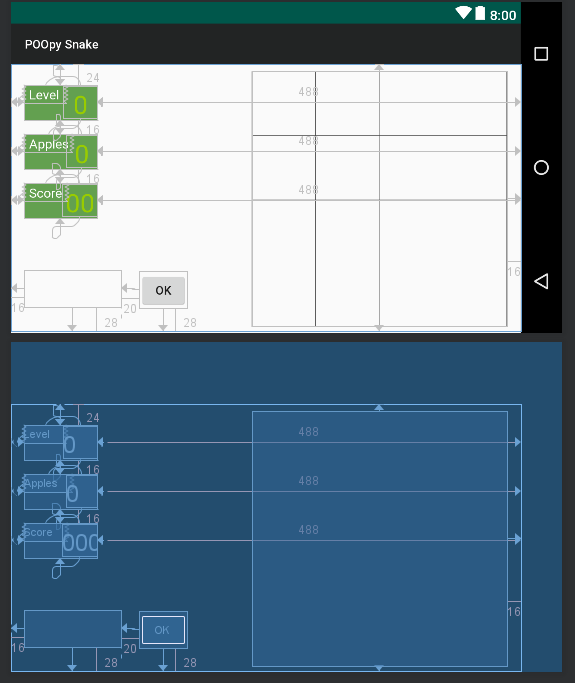
No *MainActivity* (onCreate) criámos todas as View’s e associamos às view’s criadas no editor usando *findViewById*. Depois de iniciarmos a variável view (TilePanel) implementamos o concretizamos a interface OnTileTouchListener. Tomando partido dos métodos onDrag(), onDragEnd() e onDragCancel() para detetar movimentos de *swipe* do jogador que ditam a direção a tomar pela Cobra.

No MainActivity, implementámos os seguintes métodos:

* **setupGameView** – faz o setSize do Panel, e cria todos os tiles que compõe o Panel. Neste métos também inciamos o HeartBeatListener.
* **loadNextLevel** – lê o ficheiro de texto guardado na pasta raw dos resources (levels) e passa o conteúdo do ficheiro para o modelo que por sua vez constrói o TilePanel de acordo a composição do mesmo. Incia o level observer (setObserver)
* **displayNextLevelButton** – controla a visibilidade do botão e da label assim como implementa o setOnClickListener do botão. Este botão é utilizado para aguardar confirmação do utilizador durante a transição de níveis e só está visível nestes momentos.
* **finishGame** – remove o HeartBeatListener e dependendo se o jogo foi ganho ou perdido inicia a Activity correspondente, finalizando a MainActivity.



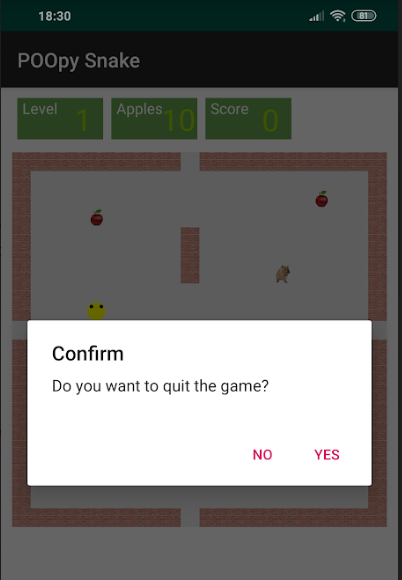
1 Main Activity - Orientação vertical



2 Main Activity - Orientação horizontal

Adicionalmente foram implementados os seguintes ***Overrides*** da classe Activity:

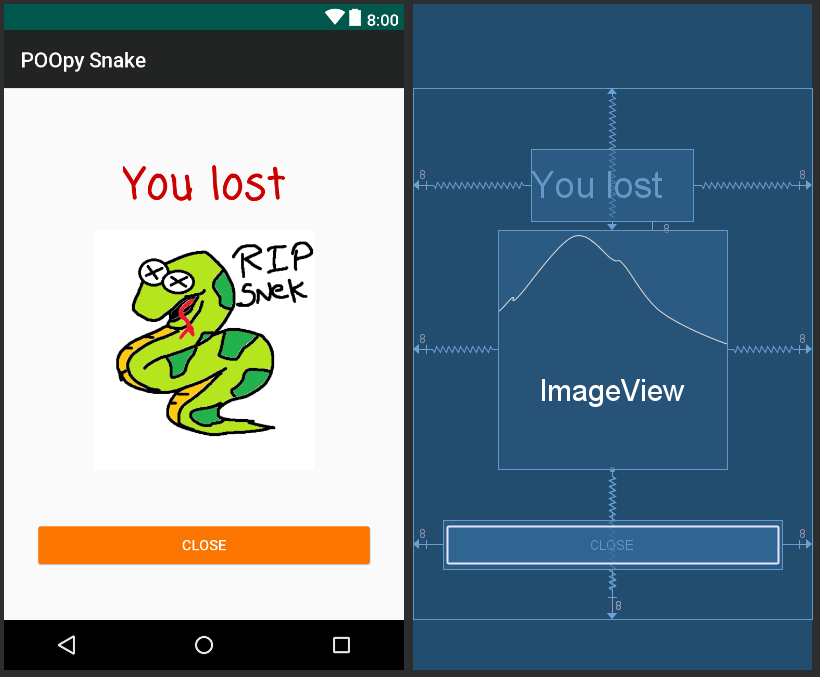
* **onCreate** – Já mencionado. Método principal do jogo em si, responsável pela orquestração dos elementos visuais e da lógica interna do jogo.
* **onStop** – Pausa o jogo.
* **onResume –** Remove a pausa e atualiza as tabelas de pontuação.
* **onSaveInstanceState –** Chamado quando a activity é destruída (rotação de ecrã, por exemplo) e permite armazenar a informação necessária à manutenção do estado atual de jogo. Tomando partido da interface *Serializable* podemos armazenar facilmente a informação do jogo e do nível atual, sendo apenas necessário o armazenamento dos mesmos no *Bundle* e de algumas variáveis primitivas.
* **onBackPressed –** Capturamos o pressionar da tecla de “voltar atrás” para evitar o encerramento imediato da aplicação, disponibilizando ao utilizador uma caixa de diálogo simples em que pode confirmar se pretende sair ou não do jogo.



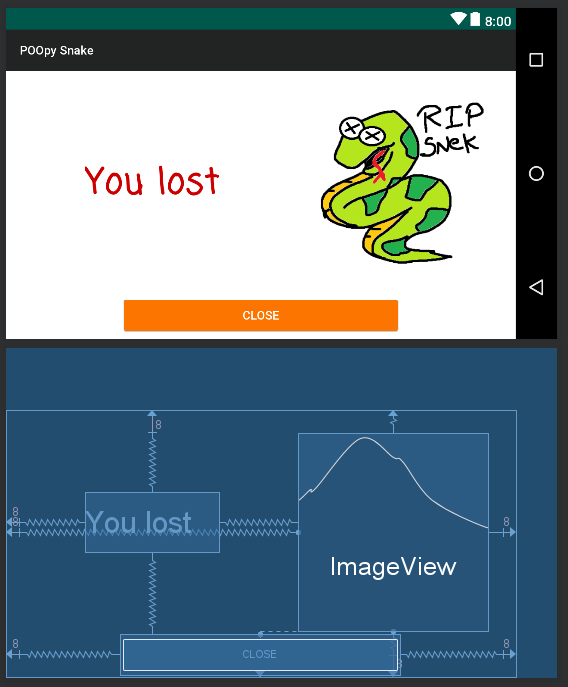
3 Diálogo de confirmação de saída

### 2.2.2 – DefeatActivity

Esta activity é apresentada quando a cobra do jogador morre e, por consequência, o jogo é perdido. É uma activity simples cuja única funcionalidade é informar o jogador da derrota e providenciar ao mesmo um botão de “fecho” que encerra a activity.



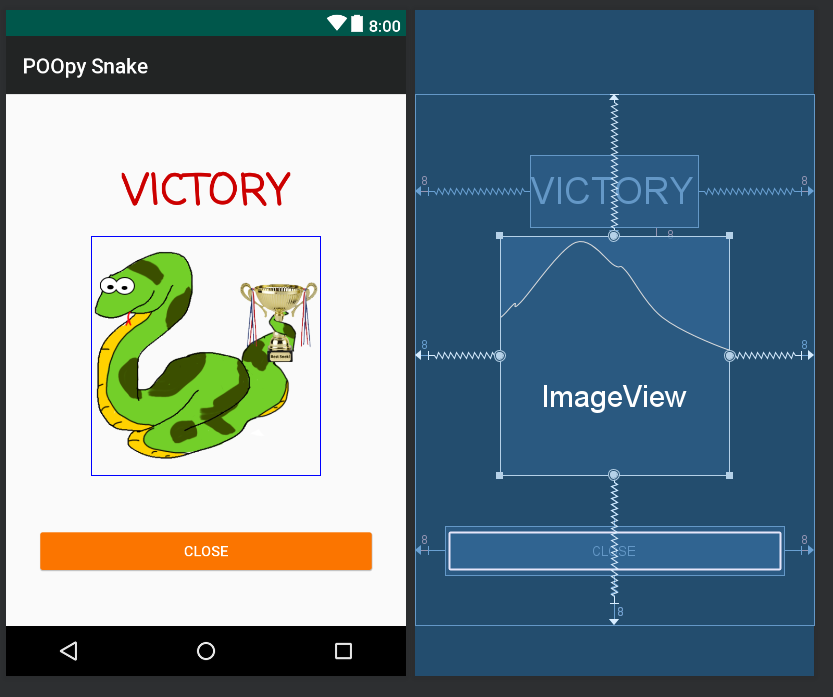
4 Ecrã de derrota – ActivityDefeat (orientação vertical)



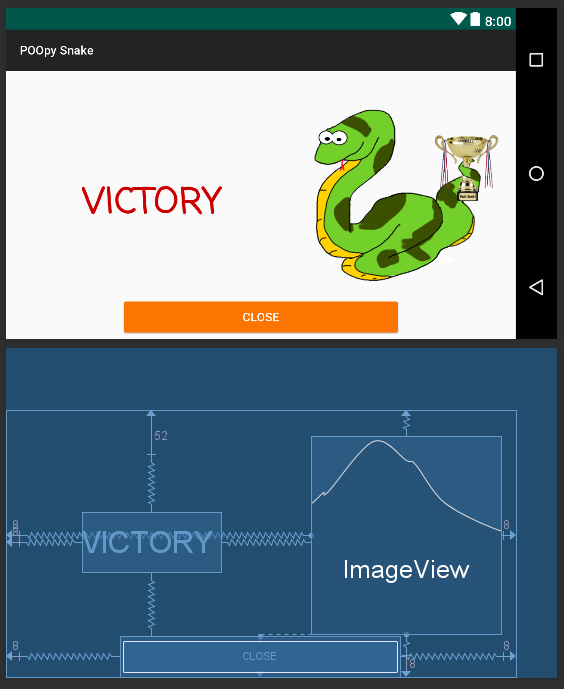
5 Ecrã de derrota – ActivityDefeat (orientação horizontal)

### 2.2.3 – VictoryActivity

Em tudo semelhante à *DefeatActivity* descrita anteriormente mas para exibição de mensagem de vitória.



6 Ecrã de vitória - Ecrã de vitória - ActivityVictory (orientação vertical)



7 Ecrã de vitória - Ecrã de vitória - ActivityVictory (orientação horizontal)

# Desenvolvimentos futuros

Em seguida deixamos uma listagem dos desenvolvimentos finais a implementar até à discussão do projeto:

* **Menu principal –** Menu de entrada no jogo onde se poderá escolher o nível a jogar e visualizar *high scores*
* **Tabela de pontuações –** Activity onde poderão ser consultados os *high scores* ordenados de forma decrescente pela sua pontuação.
* **Escolha de nível –** Activity onde se poderá escolher qual o nível a jogar. O jogador poderá escolher qualquer nível desde que tenha vencido os anteriores. Em caso de ainda não ter vencido nenhum nível irá sempre ser levado para o primeiro.
* **Optimizações do modelo –** Ainda existem alguns aspectos do modelo que poderão ser optimizados tendo em conta os conhecimentos adquiridos no decorrer do semestre, tais como tomar partido da API de instrospeção para melhoria dos padrões *Factory* ou uso de despacho dinâmico para simplificação e robustez do código das células.

# Conclusão

Todas as funcionalidades principais do projeto foram implementadas tais como algumas das opcionais, nomeadamente:

* *­Layouts* diferentes para cada orientação do ecrã
* Reposição do estado de jogo quando o ecrã é rodado
* Internacionalização de todas as mensagens para Inglês e Português
* Implementação de ratos e cobras inimigas

Em falta estão os desenvolvimentos mencionados no ponto 3.

Em suma podemos concluir que o projeto foi bastante desafiante mas ao mesmo tempo extremamente didáctico pois permitiu-nos adquirir um conhecimento muito mais aprofundado não só da linguagem Java e do sistema Android mas principalmente de alguns padrões e técnicas de programação orientada a objectos e a sua aplicação no “mundo real”. Este crescimento é-nos mais aparente principalmente quando consultamos código que escrevemos ao longo do semestre e no qual nos é possível detetar erros e falhas e, principalmente, perceber como poderíamos ter feito melhor o que, a nosso ver, é um bom indicador de aprendizagem e crescimento pessoal.