RELATÓRIO

Laboratório de Programação Orientada a Objetos

"Pixel Realm"

8 de Junho de 2014

Filipe Fernandes Miranda – ei12021

João Pedro Norim Marques Bandeira – ei12022

Índice

Introdução

Para este segundo projeto da disciplina de Laboratório de Programação Orientada a Objetos (LPOO), foi-nos proposta a realização de um programa em Java, dentro de uma lista de sugestões fornecidas. Visto que nenhum dos elementos do grupo estava particularmente interessado em algum dos temas propostos, e segundo sugestão do professor das aulas teórico-prácticas, decidiu-se pensar num tema alternativo.

Após alguma pesquisa, decidimos fazer um jogo baseado no “Realm of the Mad God”. Este jogo, seria desenvolvido para a plataforma Android, constituindo assim um desafio ainda maior, pois o grupo não estava familiarizado com o desenvolvimento de programas na mesma. Sendo assim, para além de ter de haver um planeamento ao nível de lógica (classes a implementar, e interações no jogo), haveria também algum trabalho de pesquisa sobre API de Android.

Descrição do jogo

Como já referido, o programa desenvolvido foi uma aplicação para Android de um jogo baseado no "Realm of the Mad God".

O utilizador do jogo controla uma personagem principal (Herói), cujo objectivo é matar os vários inimigos com que se depara, de forma a poder progredir no jogo.

Quanto à interface do jogo, é bastante simples e intuitiva. O Herói toma uma posição central no ecrã, e para o mover pelo mapa, o utilizador pode recorrer a um joystick localizado no canto inferior esquerdo do ecrã.

O Herói pode tirar vida aos seus inimigos de duas formas: a primeira é num "confronto cara-a-cara", em que o Herói vai ao encontro de um inimigo, e enquanto estiverem em contacto, vão ambos perdendo vida de acordo com o poder de ataque do outro; e a segunda é lançando projécteis (na interface do jogo representados como setas) na direcção do inimigo (bastando para isto tocar na zona do ecrã para onde se pretende lançar a seta).

Quando um inimigo morre, pode, ou não, ser gerado um objeto que pode ser apanhado pelo Herói. Este objeto pode ser de três tipos:

1. Vida – ao ser apanhado, a vida do Herói é incrementada.
2. Ataque – ao ser apanhado, o "poder de ataque" do Herói aumenta, isto é, cada vez que atacar um inimigo, este perderá mais vida.
3. "Moedas" – ao serem apanhadas moedas, é incrementada uma barra no topo do ecrã. Quando o utilizador consegue preencher toda a barra, abre um "portal" que permite o acesso ao nível seguinte.

Como referido, um "portal" dá acesso aos níveis seguintes. Um nível superior caracteriza-se por ter inimigos mais fortes (retiram mais vida ao Herói sempre que o atacam).

Sempre que o utilizador pausa o jogo, ou sai do mesmo, o estado do Herói é gravado, e quando recomeçar, estará tal e qual como foi deixado. É também guardado o nível em que o herói se encontrava no último jogo.

A instalação do jogo pode ser feita correndo o ficheiro .apk enviado, em qualquer dispositivo Android (4.0 ou superior).

Alguns “screenshots” do jogo:

Imagem 1 Ecrã inicial (horizontal)

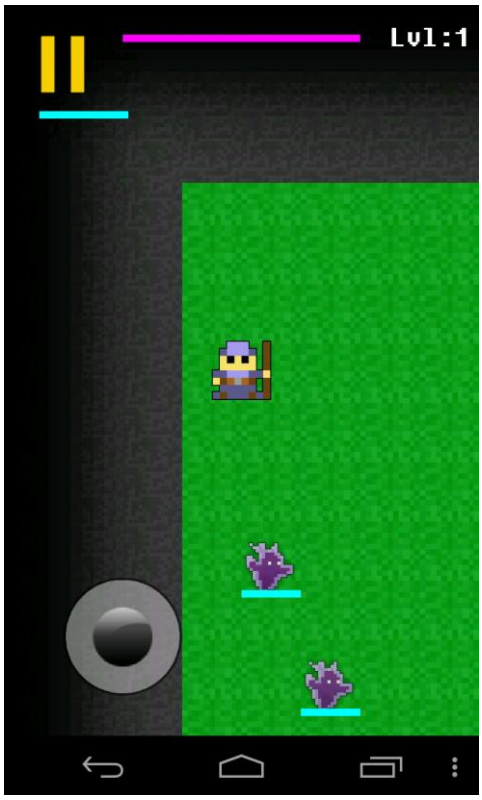


Imagem 2 Início do jogo. Herói e inimigos

Imagem 3 Herói a disparar setas contra inimigo

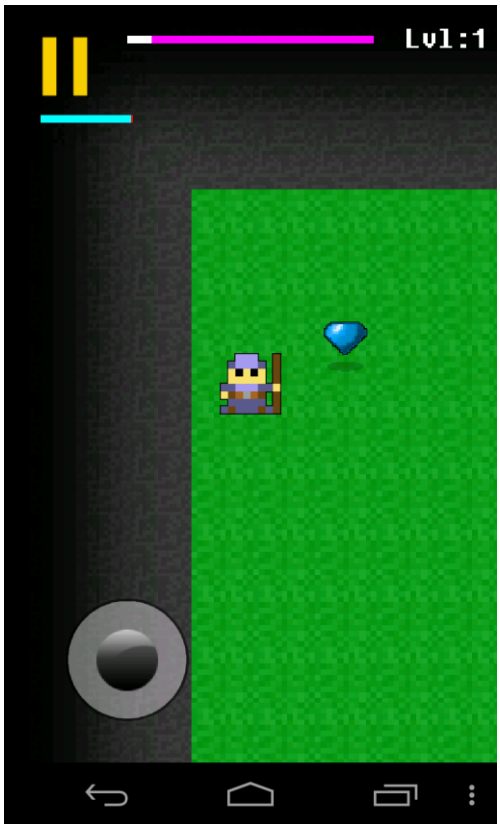


Imagem 4 Gem azul (moeda)

Imagem 5 Gem vermelha (vida)

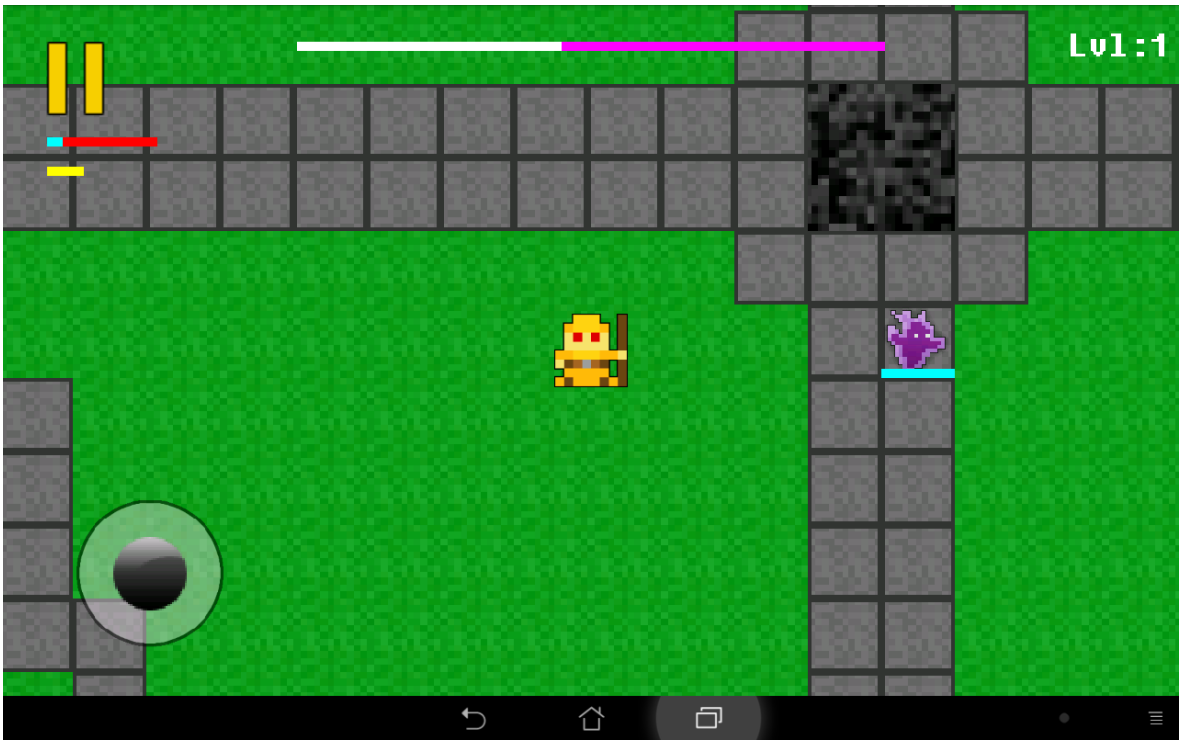


Imagem 6 Herói com maior ataque (versão do jogo em tablet)

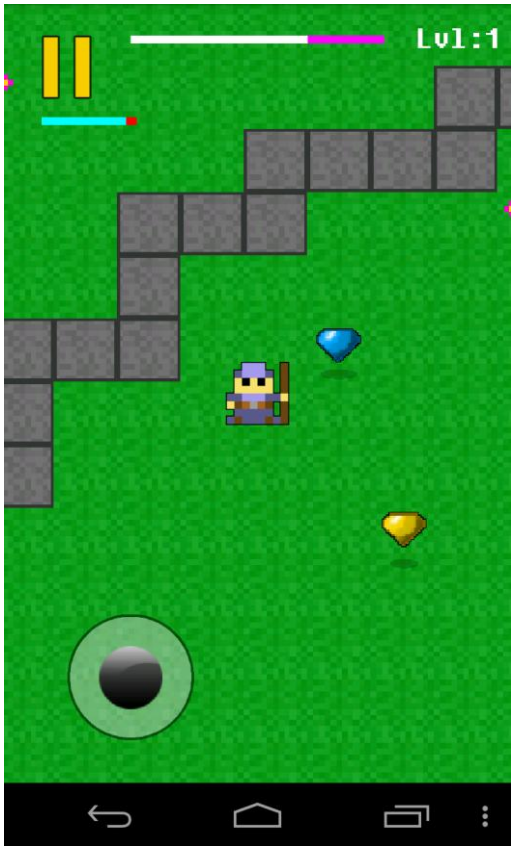


Imagem 7 Gem amarela (ataque)

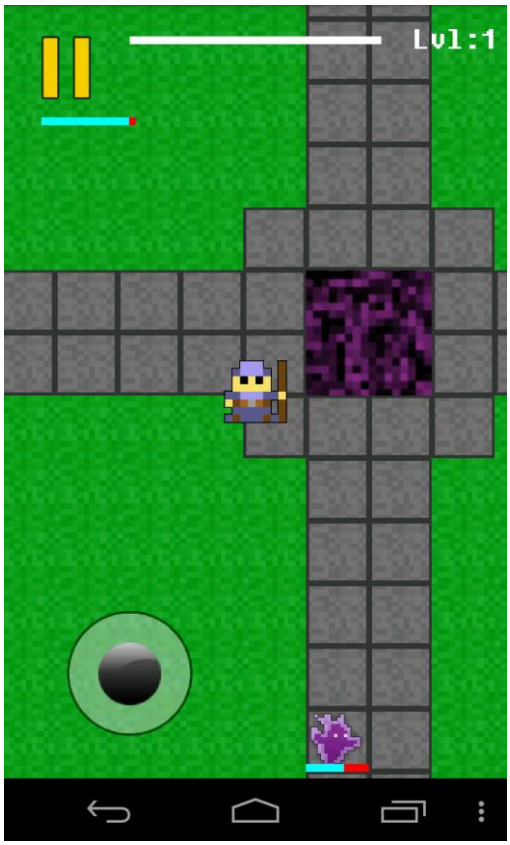


Imagem 8 Portal de passagem para nível seguinte

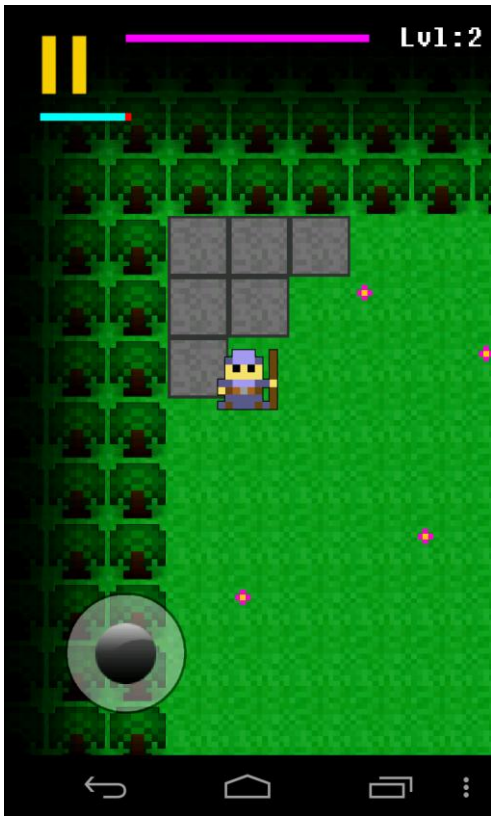


Imagem 9 Nível 2

Imagem 10 Ecrã de jogo quando jogador perde

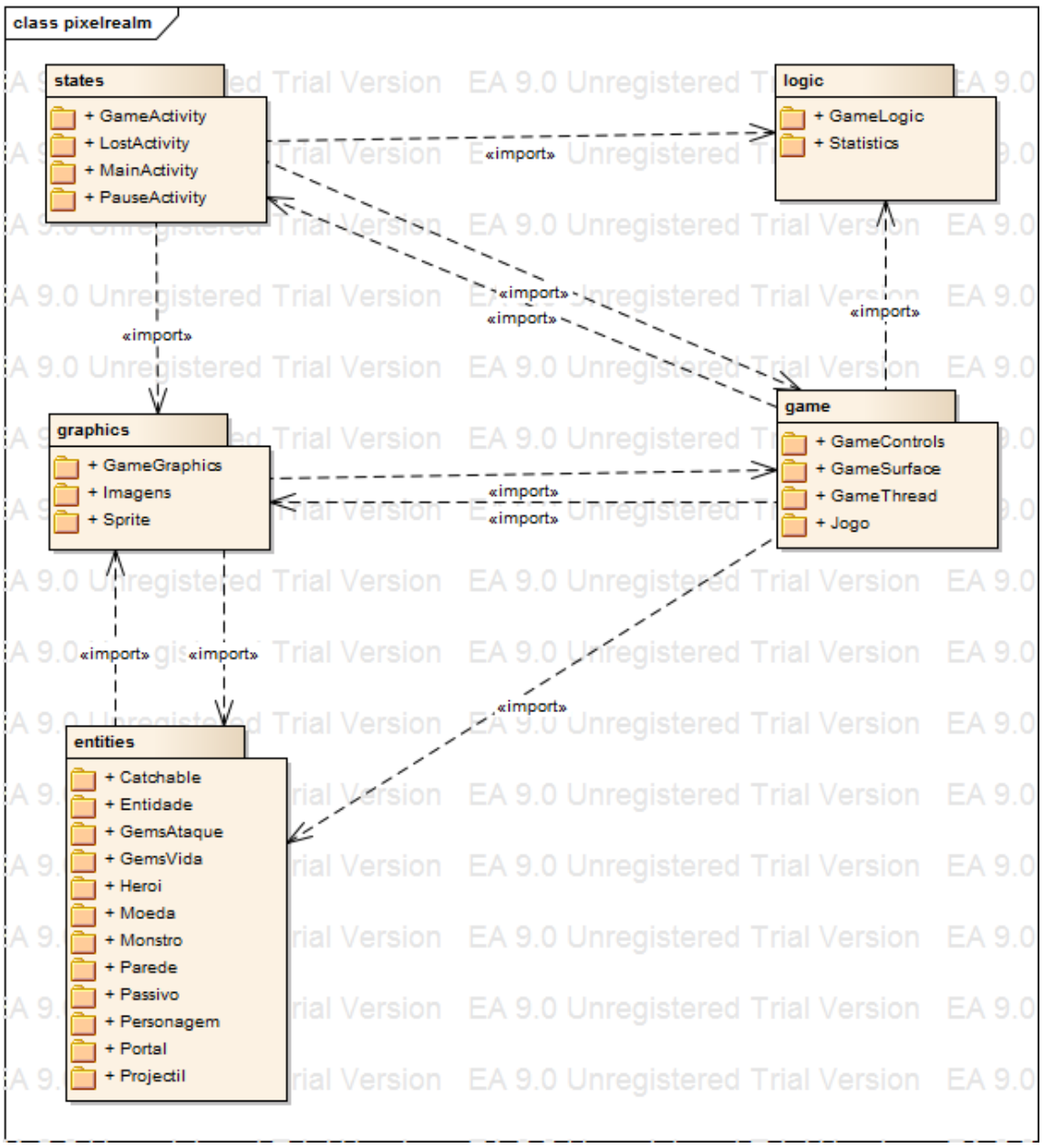
Imagem 11 Menu de pausa

UML

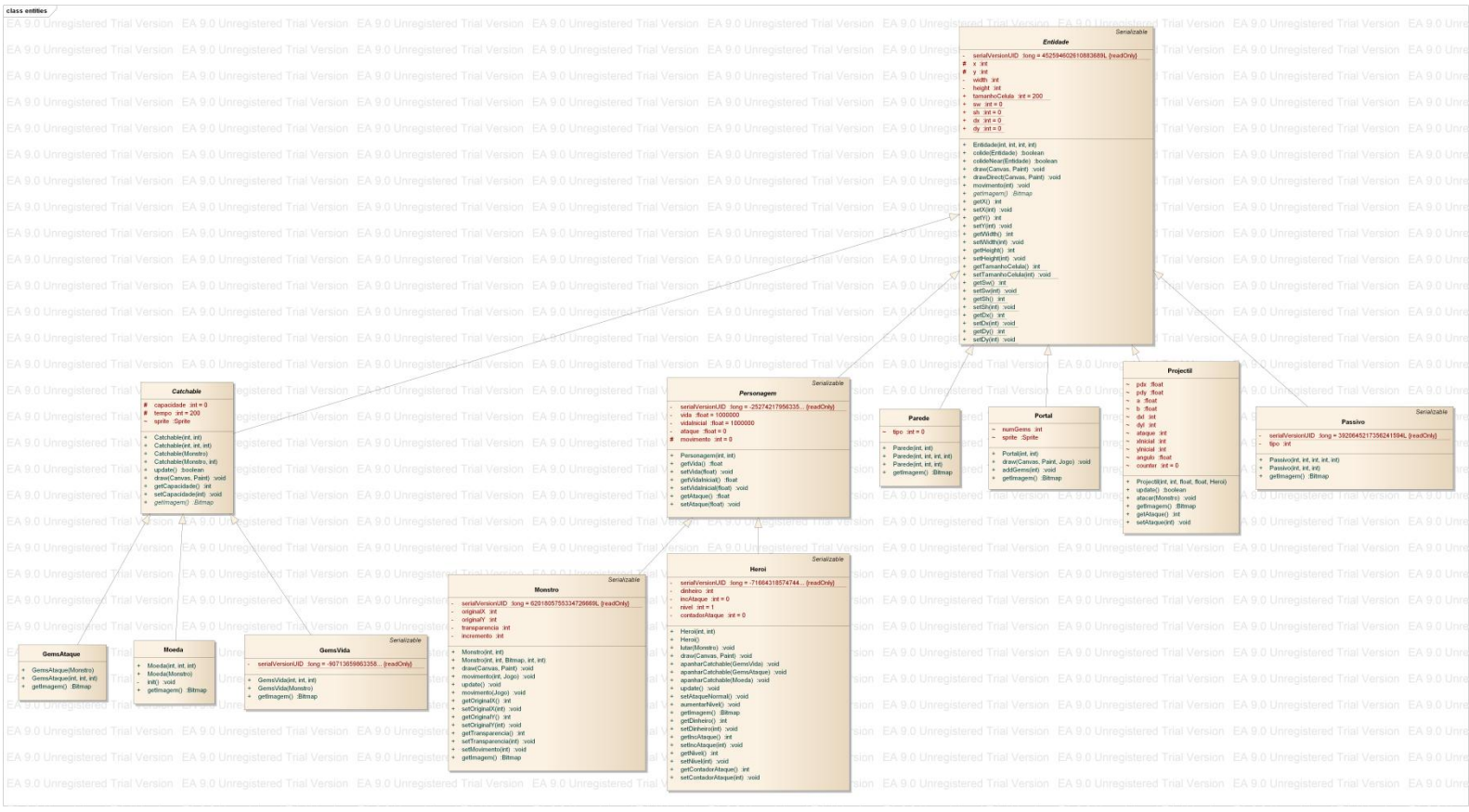
Apresentam-se abaixo os diagramas de classes relativos ao projeto, divididos por package (como o grupo não conseguiu utilizar os computadores da FEUP para gerar o UML, as imagens acabaram por ficar com marcas de água da versão de teste do Enterprise Architect).

Nota: apenas a versão PDF deste relatório permite fazer a ampliação necessária para visualizar corretamente os diagramas.

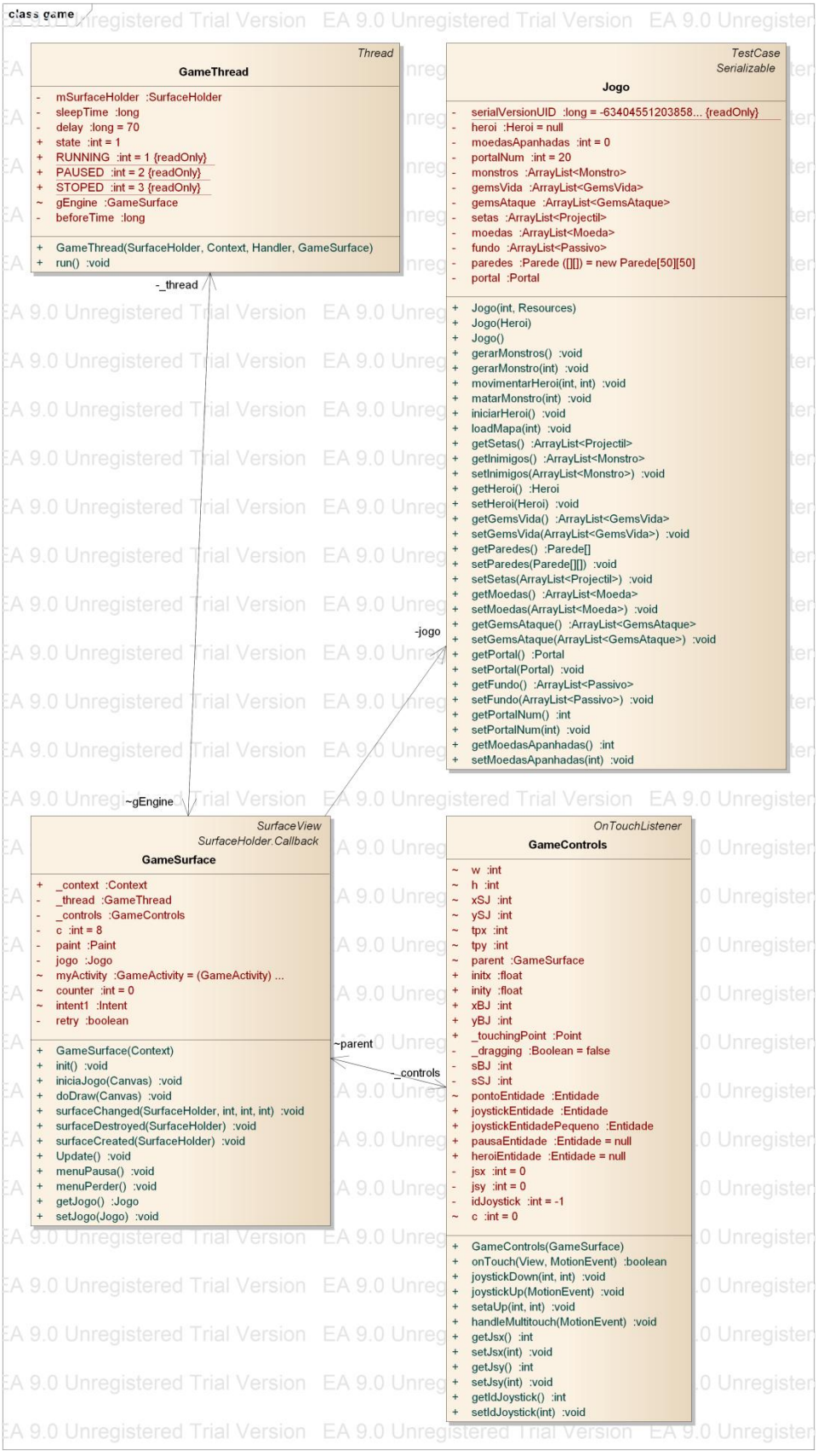
- **Packages**



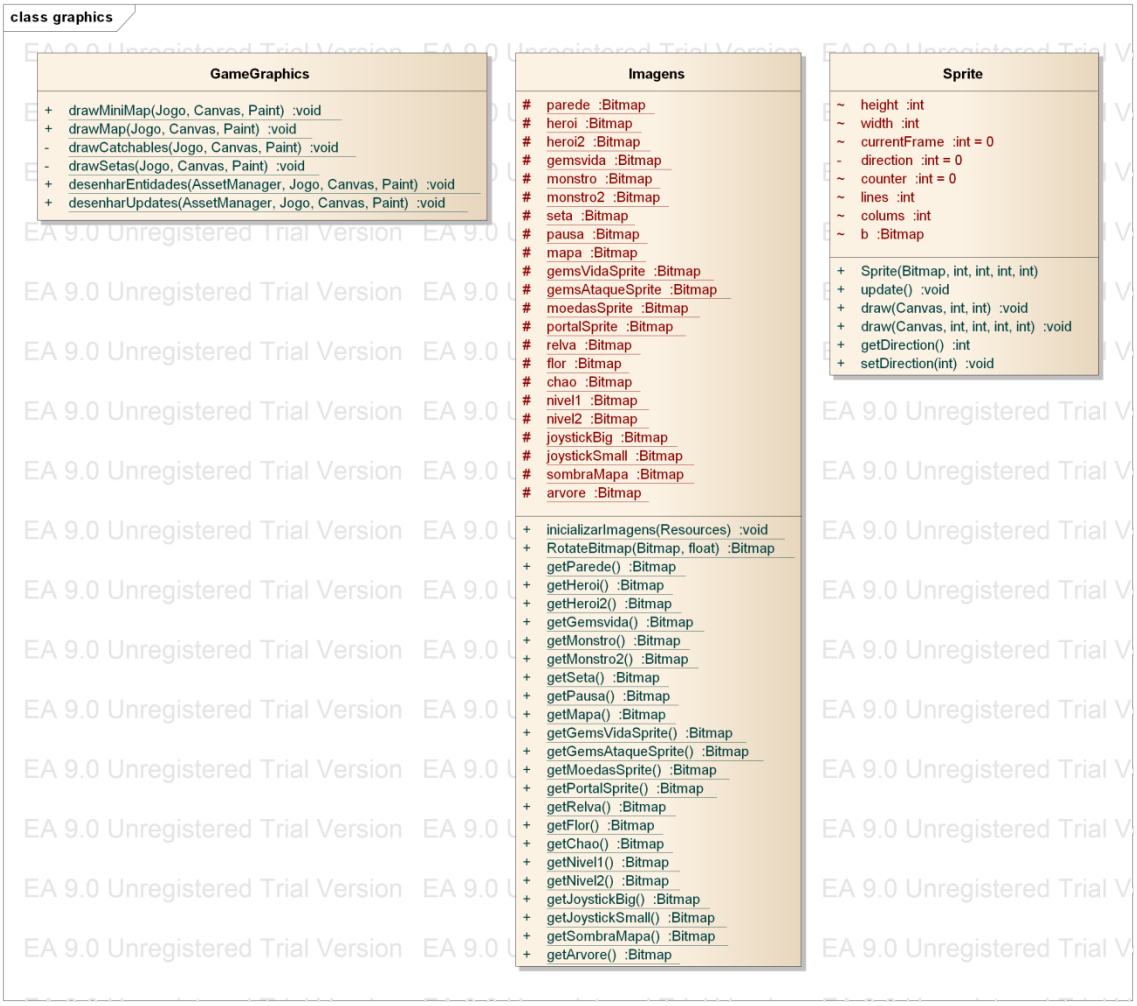
- **Entidades**


class entities
Entidade
Catchable
Personagem
Parede
Portal
Projectil
Passivo
GemsAtaque
Moeda
GemsVida
Monstro
Heroi

- **Jogo**

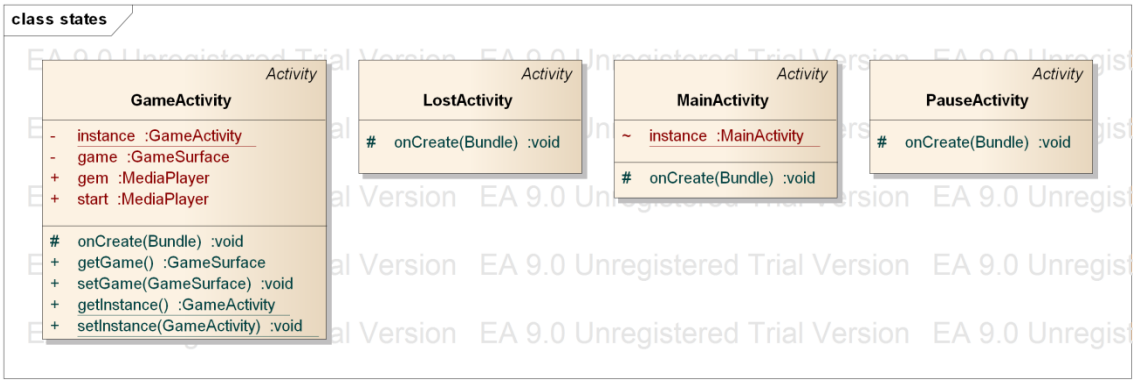

class game
Thread
GameThread
- mSurfaceHolder :SurfaceHolder
- sleepTime :long
- delay :long = 70
+ state :int = 1
+ RUNNING :int = 1 {readOnly}
+ PAUSED :int = 2 {readOnly}
+ STOPED :int = 3 {readOnly}
~ gEngine :GameSurface
- beforeTime :long
+ GameThread(SurfaceHolder, Context, Handler, GameSurface)
+ run() :void
-_thread
TestCase
Serializable
Jogo
- serialVersionUID :long = -63404551203858... {readOnly}
- heroi :Heroi = null
- moedasApanhadas :int = 0
- portalNum :int = 20
- monstros :ArrayList<Monstro>
- gemsVida :ArrayList<GemsVida>
- gemsAtaque :ArrayList<GemsAtaque>
- setas :ArrayList<Projectil>
- moedas :ArrayList<Moeda>
- fundo :ArrayList<Passivo>
- paredes :Parede ([][]) = new Parede[50][50]
- portal :Portal
+ Jogo(int, Resources)
+ Jogo(Heroi)
+ Jogo()
+ gerarMonstros() :void
+ gerarMonstro(int) :void
+ movimentarHeroi(int, int) :void
+ matarMonstro(int) :void
+ iniciarHeroi() :void
+ loadMapa(int) :void
+ getSetas() :ArrayList<Projectil>
+ getInimigos() :ArrayList<Monstro>
+ setInimigos(ArrayList<Monstro>) :void
+ getHeroi() :Heroi
+ setHeroi(Heroi) :void
+ getGemsVida() :ArrayList<GemsVida>
+ setGemsVida(ArrayList<GemsVida>) :void
+ getParedes() :Parede[]
+ setParedes(Parede[][]) :void
+ setSetas(ArrayList<Projectil>) :void
+ getMoedas() :ArrayList<Moeda>
+ setMoedas(ArrayList<Moeda>) :void
+ getGemsAtaque() :ArrayList<GemsAtaque>
+ setGemsAtaque(ArrayList<GemsAtaque>) :void
+ getPortal() :Portal
+ setPortal(Portal) :void
+ getFundo() :ArrayList<Passivo>
+ setFundo(ArrayList<Passivo>) :void
+ getPortalNum() :int
+ setPortalNum(int) :void
+ getMoedasApanhadas() :int
+ setMoedasApanhadas(int) :void
-jogo
~gEngine
SurfaceView
SurfaceHolder.Callback
GameSurface
+ _context :Context
- _thread :GameThread
- _controls :GameControls
- c :int = 8
- paint :Paint
- jogo :Jogo
~ myActivity :GameActivity = (GameActivity) ...
~ counter :int = 0
~ intent1 :Intent
- retry :boolean
+ GameSurface(Context)
+ init() :void
+ iniciaJogo(Canvas) :void
+ doDraw(Canvas) :void
+ surfaceChanged(SurfaceHolder, int, int, int) :void
+ surfaceDestroyed(SurfaceHolder) :void
+ surfaceCreated(SurfaceHolder) :void
+ Update() :void
+ menuPausa() :void
+ menuPerder() :void
+ getJogo() :Jogo
+ setJogo(Jogo) :void
~parent
-_controls
OnTouchListener
GameControls
~ w :int
~ h :int
~ xSJ :int
~ ySJ :int
~ tpx :int
~ tpy :int
~ parent :GameSurface
+ initx :float
+ inity :float
+ xBJ :int
+ yBJ :int
+ _touchingPoint :Point
- _dragging :Boolean = false
- sBJ :int
- sSJ :int
~ pontoEntidade :Entidade
+ joystickEntidade :Entidade
+ joystickEntidadePequeno :Entidade
+ pausaEntidade :Entidade = null
+ heroiEntidade :Entidade = null
- jsx :int = 0
- jsy :int = 0
- idJoystick :int = -1
~ c :int = 0
+ GameControls(GameSurface)
+ onTouch(View, MotionEvent) :boolean
+ joystickDown(int, int) :void
+ joystickUp(MotionEvent) :void
+ setaUp(int, int) :void
+ handleMultitouch(MotionEvent) :void
+ getJsx() :int
+ setJsx(int) :void
+ getJsy() :int
+ setJsy(int) :void
+ getIdJoystick() :int
+ setIdJoystick(int) :void

- **Gráficos**

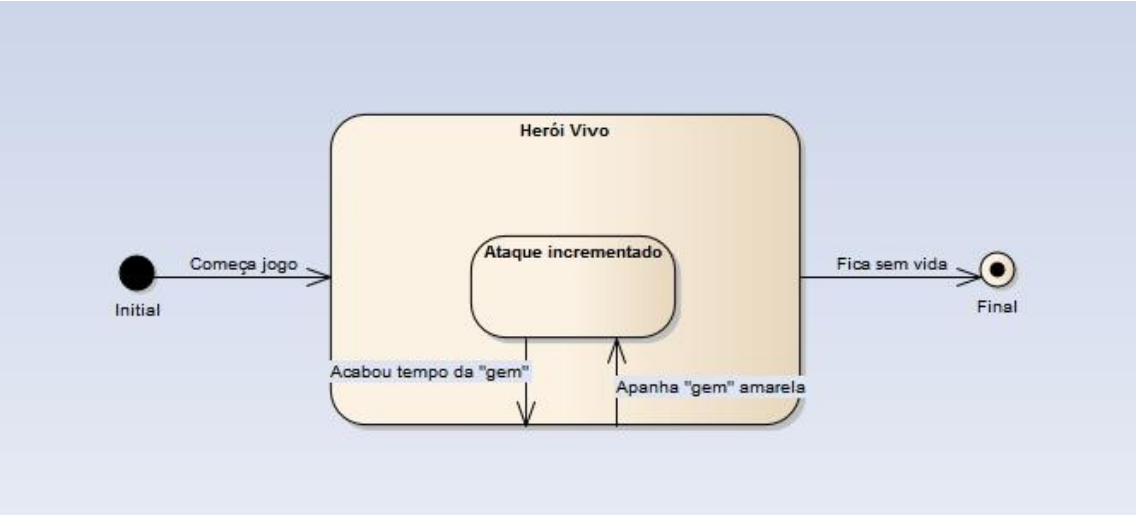

class graphics
GameGraphics
+ drawMiniMap(Jogo, Canvas, Paint) :void
+ drawMap(Jogo, Canvas, Paint) :void
- drawCatchables(Jogo, Canvas, Paint) :void
- drawSetas(Jogo, Canvas, Paint) :void
+ desenharEntidades(AssetManager, Jogo, Canvas, Paint) :void
+ desenharUpdates(AssetManager, Jogo, Canvas, Paint) :void
Imagens
parede :Bitmap
heroi :Bitmap
heroi2 :Bitmap
gemsvida :Bitmap
monstro :Bitmap
monstro2 :Bitmap
seta :Bitmap
pausa :Bitmap
mapa :Bitmap
gemsVidaSprite :Bitmap
gemsAtaqueSprite :Bitmap
moedasSprite :Bitmap
portalSprite :Bitmap
relva :Bitmap
flor :Bitmap
chao :Bitmap
nivel1 :Bitmap
nivel2 :Bitmap
joystickBig :Bitmap
joystickSmall :Bitmap
sombraMapa :Bitmap
arvore :Bitmap
+ inicializarImagens(Resources) :void
+ RotateBitmap(Bitmap, float) :Bitmap
+ getParede() :Bitmap
+ getHeroi() :Bitmap
+ getHeroi2() :Bitmap
+ getGemsvida() :Bitmap
+ getMonstro() :Bitmap
+ getMonstro2() :Bitmap
+ getSeta() :Bitmap
+ getPausa() :Bitmap
+ getMapa() :Bitmap
+ getGemsVidaSprite() :Bitmap
+ getGemsAtaqueSprite() :Bitmap
+ getMoedasSprite() :Bitmap
+ getPortalSprite() :Bitmap
+ getRelva() :Bitmap
+ getFlor() :Bitmap
+ getChao() :Bitmap
+ getNivel1() :Bitmap
+ getNivel2() :Bitmap
+ getJoystickBig() :Bitmap
+ getJoystickSmall() :Bitmap
+ getSombraMapa() :Bitmap
+ getArvore() :Bitmap
Sprite
~ height :int
~ width :int
~ currentFrame :int = 0
- direction :int = 0
~ counter :int = 0
~ lines :int
~ colums :int
~ b :Bitmap
+ Sprite(Bitmap, int, int, int, int)
+ update() :void
+ draw(Canvas, int, int) :void
+ draw(Canvas, int, int, int, int) :void
+ getDirection() :int
+ setDirection(int) :void

- **Lógica**

- **"Activities"**



- **Diagrama de estados do Herói**



Design Patterns

Neste projeto foram utilizados diferentes “design patterns”:

- Singleton: este padrão foi utilizado na implementação da “GameActivity”. É um padrão que é utilizado para que não exista mais do que uma instância de um objeto. A “GameActivity” é um desses objetos. Também na classe “Imagens” se utilizaram unicamente objetos estáticos, eliminando a necessidade de repetidamente se instanciarem imagens, o que ocuparia uma quantidade vastamente superior de memória.
- “Template pattern”: a utilização de classes abstratas, e subclasses que implementam os seus métodos, conforme o comportamento de cada uma, foram uma das principais técnicas utilizadas no projeto, e características deste padrão de desenho de software.

Detalhes do desenvolvimento e algumas dificuldades

Como já referido na Introdução, um dos maiores desafios deste projeto foi a necessidade de algum trabalho de pesquisa relativo ao desenvolvimento de aplicação para o sistema operativo Android, ligeiramente diferente do que tinha sido feito no primeiro projeto (Projeto Guiado).

Segue-se uma breve explicação dos principais focos de desenvolvimento:

- **Desenvolvimento do Joystick:** o jogo desenvolvido baseia-se numa jogabilidade com recorrência ao Joystick, pelo que tivemos de investigar e recolher informações de várias fontes para percebermos qual seria a maneira mais fácil e correcta de implementar esse joystick. Foi sempre prioridade tornar a interação do jogador com o herói bastante intuitiva, rápida, "responsive" e encapsulada, para que com apenas a modificação de algumas variáveis fosse possível alterar a sua posição, tamanho e impulso/movimento que da ao herói.

- **Implementação de multitouch:** para além do joystick, o jogador pode também clicar em várias partes do ecrã para disparar “projéteis”. Para tornar mais intuitiva a interação do utilizador com o jogo, decidimos implementar um handler de Multitouch, desenvolvido por nós, que conseguiria registar dois ou mais toques ao mesmo tempo, de modo a que fosse possível movimentar o “herói” e disparar ao mesmo tempo.

- **Alocações de memória para Imagens:** na aplicação de gráficos ao jogo, para ultrapassar o problema de memória com a instanciação de um Bitmap para cada objeto, decidimos criar uma classe com todas as imagens, inicializada no início da aplicação. Deste modo, cada Entidade não teria uma Bitmap, mas sim uma referência para um determinado Bitmap instanciado na classe Imagens.

- Para a aplicação de animações no jogo, o grupo decidiu aplicar **Sprites**. Criámos uma função que tratasse de fazer update à imagem que iria ser mostrada, sendo que o Bitmap de source estaria já instanciado na classe Imagens, diminuindo o custo de memória do jogo.

- **Centered Player Camera:** um dos grandes problemas que nos propusemos a resolver com este jogo foi o do "Centered Camera Player", isto é, o jogador encontra-se sempre no centro do ecrã e tudo o resto é que se movimenta. Para tal, decidimos abordar os movimentos do player como um incremento ou decremento de duas variáveis: deslocamento na horizontal (dx) e deslocamento na vertical (dy). Deste modo, não tivemos de modificar todas as posições, de todos os objetos no tabuleiro a cada movimento do jogador, mas sim trabalhar sempre com a posição das Entidades e os deslocamentos no jogo.

- **Resizable Gameplay:** sendo que decidimos desenvolver um jogo para Android, seria mais do que natural começar desde inicio a pensar como iria ser feito o "scale" para telemóveis de maiores e menores dimensões, bem como tablets. Para tal, todo o desenho do jogo no ecrã é calculado recorrendo à largura e comprimento do telemóvel. O algoritmo implementado permite-se adaptar a telemóveis mais largos ou mais estreitos, em "landscape e portrait" e até em tablets, nunca distorcendo as imagens.

- Foram ainda aplicados **vários layouts, dependendo da resolução**, e orientação do ecrã, para tirar mais partido do "screen estate".

Conclusão

Fazendo um balanço geral do trabalho, pode dizer-se que foi positivo. O grupo conseguiu alcançar todos os objetivos que se propôs, adquirindo ainda novos conhecimentos, possivelmente úteis no futuro.

De referir também que, apesar do conhecimento da existência de bibliotecas que simplificam o desenvolvimento de aplicações para Android, o grupo decidiu não as utilizar, limitando-se à utilização de classes e métodos característicos do mesmo.

Relativamente a melhorias que podem ser feitas ao jogo, essencialmente poderiam ser desenhados novos níveis, com mais diversidade de cenários e inimigos, o que seria uma mais-valia para os utilizadores.

O grupo terminou o projeto com o sucesso pretendido, e considera que foi sem dúvida uma boa forma de aprofundar conhecimentos quer na linguagem Java, como na plataforma Android, podendo ter sido um impulso para o desenvolvimento de mais aplicações nesta plataforma no futuro como programadores.

Referências

- http://www.realmofthemadgod.com/ - jogo de base para o projeto desenvolvido;
- http://www.realmeye.com/wiki/realm-of-the-mad-god - fonte de algumas imagens utilizadas no jogo.