2015

Relatório de LeGame - Remake

Como está organizada a programação

Neste documento vai estar explicado como está organizado o LeGame – Remake bem como fazer algumas alterações básicas.

João Lourenço
LeGame - Remake
17-03-2015

Índice

Introdução

Para este projecto foi usado o Java 7 e OpenGL 2.1.
O objectivo do jogo é muito simples, sobreviver o maior número de níveis possível enquanto vários tipos de monstros nos tentam matar. Ao morrer voltamos para o nível 1.
O jogo usa dois sistemas de rendering, Instant Rendering e VBO Rendering.
Instant Rendering está a ser usado apenas para alguns objectos menos importantes e VBO para os mais importantes.
O jogo tem uma vista Top-To-Down em 2D e usa Tiled based mapping.
No final deste relatório existe um sector de termos caso haja dúvidas em algum termo.
O jogo foi desenvolvido com um sistema refresh-rate, ou seja, cada frame o jogo é totalmente redesenhado no ecrã.

Desenvolvimento da ideia e criação do projecto

A ideia para este projecto surgiu na aula de PSI como a criação de uma segunda e melhorada versão do jogo LeGame inicialmente criado em Pascal por mim no 10º (2013/2014).
A ideia inicial seria manter o mesmo objectivo do jogo e apenas adicionar a vertente 2D Pixel by Pixel em vez de Character by Character.
Este projecto foi criado em eclipse no entanto pode ser visto e/ou editado em qualquer editor de Java ou de texto, no entanto para ser compilado precisa de o Java Compiler.
Uma vez que o jogo usa OpenGL os DLL's e natives do mesmo são obrigatórios para o correr. Para adicionar estes ficheiros e obter o "Fat Jar" (Jar file com todos os resources, DLL's e Natives já incluídos) foi usado um programa chamado, Jarsplice.

Organização em Geral

Neste relatório vai estar apenas noções básicas e como fazer certas alterações, para explicações mais detalhadas o código está comentado linha a linha.
Uma vez que este é um grande projecto e conta com cerca de 10,000 linhas de código é fundamental um alto nível de organização. Para obter este nível de organização o projecto está dividido em grandes 3 partes, src, lib e res.

- Src ou Source Code é a pasta que contém todo o código escrito em Java.
- Lib ou Libraries é onde se encontram todas as bibliotecas extras e as suas natives e/ou Dll's.
- Res ou Resources contem todos os recursos extras como imagens ou shaders.

Dentro de cada uma destas pastas encontra-se ainda subpastas mais específicas, no caso do src as packages que vou falar mais á frente.

O jogo funciona com 2 grandes tipos de Objectos que contêm subobjectos explicados em baixo.

- Entities – Todos os Objectos que apareçam no jogo que precisem de ter movimento e/ou vida.
  - Block – Faz parte das Entities pode se mexer mas não tem vida.
  - Light - Faz parte das Entities pode se mexer mas não tem vida.
  - Mob - Faz parte das Entities pode se mexer e tem vida.
- World – Contêm todas as entities e tiles presentes no mundo em questão.
  - Tiles – Tile ou Azulejo é o que define o mundo.

Packages:

net.joaolourenco.legame	(Onde se encontram as classes principais)
net.joaolourenco.legame.entity	(Classes Modelo para as Entities)
net.joaolourenco.legame.entity.actions	(Classes que contêm acções pré-definidas para as entities se poderem por exemplo mexer.)
net.joaolourenco.legame.entity.block	(Classes para os blocos com movimento se encontram)
net.joaolourenco.legame.entity.light	(Classes Modelos para as luzes que o jogo tem)
net.joaolourenco.legame.entity.light.specific	(Classes para luzes em especifico como as Spot e Point light)
net.joaolourenco.legame.entity.mob	(Classes para todas as entities com vida)
net.joaolourenco.legame.graphics	(Classes básicas de grafismos)
net.joaolourenco.legame.graphics.font	(Classes responsáveis pelo desenho de letras)
net.joaolourenco.legame.graphics.menu	(Classes responsáveis por todos os menus)
net.joaolourenco.legame.graphics.menu.objects	(Classes responsáveis pelos objectos dos menus como botões, sliders e etc.)
net.joaolourenco.legame.items	(Classes para gestão e criação de items não físicos para o jogo)
net.joaolourenco.legame.settings	(Classes que gerem todas as definições do jogo)
net.joaolourenco.legame.utils	(Classes extras ao jogo como cálculos matemáticos)
net.joaolourenco.legame.world	(Classes que gerem o Mundo)
net.joaolourenco.legame.world.tile	(Classes que gerem os vários tipos de tiles presente no Mundo)

Explicação de Classes:

net.joaolourenco.legame

Main

Esta class contém a Main method do Java e é o que gere todo o game loop.

A Main class contém um Thread onde vai correr o Jogo, uma instância da world class e todas as funções necessárias para a interacção com as mesmas.

Funções:

- Main – Main method usada pelo java para ser iniciada a aplicação.
- Start – Method para iniciar o thread do jogo.
- Stop – Method para terminar o jogo.
- Init – Method para inicializar o jogo. Vai criar a janela, definir as projecções, fazer o load básico de texturas e abrir o menu principal.
- Run –Method apenas chamada uma vez durante o Runtime todo que contem o game loop. O game loop, definido por um while loop, chama 3 funções, render, update e tick. Quando o while loop for terminado vai ser chamada a função cleanup.
- Render – Esta função é responsável pelos renders todos do jogo, daqui partem 4 chamadas principais, uma para o world para que este faça o render do mundo e de todas as suas entidades, uma para os todos menus abertos, uma para todos os textos animados e por fim uma para todos os textos estáticos. Esta função é chamada no máximo 300 vezes por segundo pois eu coloquei um limite para poupar os recursos da máquina, este limite pode ser alterado entre 300 e 30 frames nas definições do jogo.
- Update – Esta method faz todos os principais updates no jogo. Daqui tal como no render partem chamadas para o Mundo, Menus e textos animados. Para esta função funcionar na perfeição deve ter 60 chamadas por segundo constantes.
- Tick – Esta method é chamada uma vez por segundo e serve para manter um controlo superior sobre os updates, aqui podem ser efectuadas verificações e ou alterações de logica que sejam mais pesadas e/ou não seja preciso fazer 60 vezes por segundo.
- Cleanup – esta função vai limpar todos os registos criados pelo jogo e memória utilizada e em seguida terminar a aplicação.
- Getters e Setters – Existe 1 getter e 4 setters nesta class e servem para isso mesmo, obter e definir as variáveis da class por membros externos á mesma.

Registry

A Registry class contém todos os registos de variáveis e estados que o jogo necessita.

Exemplos de registos são, a própria main class, a instância do player, textos, shaders, displaymodes.

Esta class está repleta de getters e setters e não faz mais nada se não adicionar, obter e remover dados das ArrayLists.

net.joaolourenco.legame.entity

Entity

A Entity class usa o modificador abstract e por isso é apenas uma class modelo. Este modelo vai ser usado para todo o tipo de entities quer sejam mobs ou blocks.

Esta class contém as seguintes variáveis:

- X e Y – Localização da entity.
- Width e Height – Tamanho da entity.
- Texture – Textura da entity.
- World – O mundo que a entity pertence.
- Removed – Uma flag para confirmar se esta entity deve ou não ser removida do processamento.
- Random – Contêm uma instância da class Random do java que serve para criar variáveis random.
- Collidable – Se a entity é collidable com outras entities.
- Light Collidable – Se a luz colide com a entity.
- Light Affected – Se a luz afecta a entity, ou seja, se fica mais claro ou escuro ao pé da luz.
- Shade – The shader for the entity.
- Inventory – ArrayList que contêm todos os items da entity.
- Renderable – Se é ou não possível desenhar a entity.
- Dying – Se a entity está a morrer ou não.
- GodMode – Se a entity pode ou não morrer.
- Time – Para actualizar a animação de morrer.
- Rate – Velocidade de actualização das animações.
- Life – Vida que a entity tem.
- RenderHealthBar – Flag para verificar se a barra de vida da entity vai ou não ser renderizada.
- VertexID – Os dados do VBO para o render engine.

Esta class contém duas funções abstract, a update e a tick para que ao se criar uma nova entity se possa configurar a mesma ao seu máximo.

Por outro lado a render method está definida para poupar código ao criar uma nova entity.

Esta class conta ainda com várias methods importantes para a sua versatilidade como:

- getVertices – Esta method permite-nos obter os vértices da entity para criar uma light shadow mais realista.
- Remove – Method para remover uma entity da memória do Mundo e se a entity em cause for o Player é dado como um Game Over.
- Init – Method responsável por guardar uma instância da world class para ser usada pela entity quando necessário.
- getSpeed – Esta method permite-nos obter a velocidade que a entity tem para que seja mais fácil a alteração da mesma.
- getX e getY – Estas methods retornam as coordenadas do canto superior esquerdo quando se é introduzido false como parâmetro, e o centro da entity quando se usa true como parâmetro.
- getTX e getTY - Estas methods retornam as coordenadas do canto superior esquerdo quando se é introduzido false como parâmetro, e o centro da entity quando se usa true como parâmetro. As coordenadas retornadas vão ser em tile size ou seja (1, 1) que corresponde a (64, 64) em pixel precision.
- setLocationAndCenter – Esta method coloca o player numa determinada localização e coloca o ecrã centrado nele.
- Attacked – Esta method é chamada cada vez que a entity é atacada, é ela que gere o dado induzido e pode produzir mensagens de aviso de for necessário.
- generateRandom – Esta method serve para criar uma random de ints ou floats dentro de um intervalo predefinido. Esta função encontra-se em várias classes.

Net.joaolourenco.legame.entity.actions

MovementAction

A class MovementAction é tal como a Entity class uma class modelo. Esta class gere acções de movimento de AI para as entities. Estas acções podem ser configuradas para serem usadas varia vezes por várias entities. Por exemplo pode ser criada uma nova class como extensão da MovementAction para fazer uma entity se mover seguindo um padrão random.

Para adicionar uma acção de movimento a uma entity basta fazer:
this.moveActions.add(new NovoObjectoDeMovimento(this));

ex:
this.moveActions.add(new RandomTargetedMovementAction(this));

Esta class contém as seguintes Methods:

- Update – Para efectuar os movimentos da entity.
- Finished – Flag para verificar se a animação / movimento foi terminado ou não.
- getXA e getYA – Para obter o movimento efectuado pela class.
- generateRandom – Esta method serve para criar uma random de ints ou floats dentro de um intervalo predefinido. Esta função encontra-se em várias classes.

PersistentTargetedMovementAction

Esta class, PersistentTargetedMovementAction, é uma extensão da MovementAction e consegue seguir constantemente uma outra entity.
Esta class usa um A* Seach Algorithm para encontrar o seu caminho.

Esta class apenas tem uma função update que usa para obter a localização e caminho para o seu alvo e para e se dirigir ao mesmo.

RandomTargetedMovementAction

Esta class, RandomTargetedMovementAction, é uma extensão da MovementAction e consegue criar um ponto qualquer no mapa e enviar a entity para lá, pode ser usada a função finished para saber se a entity já chegou ao seu destino ou não.
No Citizen (Umas das entity mobs do jogo) este algoritmo é usado e cada vez que chega ao seu destino é gerado um novo ponto fazendo assim uso da finished method para criar uma Persistent Random Targeted Movement Action.
Esta class usa um A* Seach Algorithm para encontrar o seu caminho tal como a PersistentTargetedMovementAction mas em vez de se focar numa outra entity foca-se num ponto estático.

Esta class apenas tem uma função update que usa para obter o caminho para o seu alvo e para e se dirigir ao mesmo.

net.joaolourenco.legame.entity.block

Door

Esta class gere as portas do jogo. Esta class contem uma enum chamada States que contem os vários estado de porta, closed, closing, open e opening.

Esta class contém as seguintes variáveis:

- state – Estado actual da porta.
- Key – Chave para a porta.
- usesKey – Se é preciso ou não uma chave para abrir a porta.
- Locked – Se a porta está traçada ou não.
- Jammed – Se a porta está encravada ou não.
- DoorGap – Gere a abertura actual da porta.
- MaxDoorGap – Abertura máxima que a porta pode ter.
- DoorRadius – Raio máximo de acção para a porta se abrir automaticamente.
- alongXAxis – Se a porta é a horizontal ou vertical.
- autoOpen e autoClose – Se a porta se abre/fecha automaticamente.
- OverrideStatus – Se a porta vai ignorar o abrir e fechar automaticamente e adoptar o estado que o jogador as deixa.

Uma vez que a porta usa dois blocos a função render têm de ser diferente da entity.
A porta pode ter ser automática ou não, pode precisar ou não de chave e para se criar uma chave para ela deve se fazer da seguinte maneira:

```
DoorKey key = new DoorKey(1, door.getKey());
```

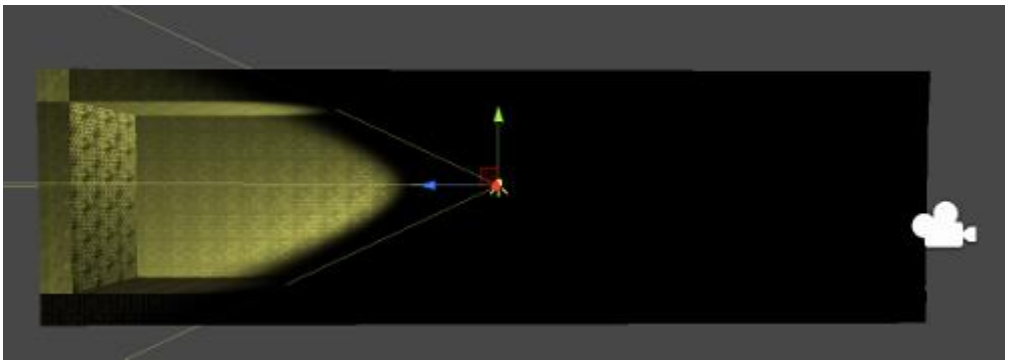
, em que door se refere á porta a que se pretende a nova chave.

net.joaolourenco.legame.entity.light

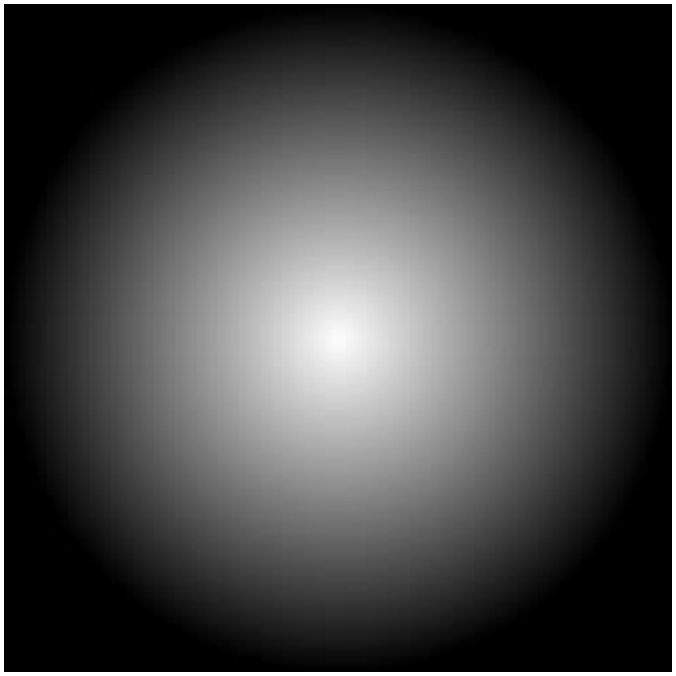
Light

A class Light é mais uma class Modelo para todas as luzes do jogo.
Esta class é a que vai criar e fazer o render das sombras da luz na method renderShadows.
Existem 2 tipos de luzes, 1 – PointLight, 2 – SpotLight.

Spot Light tem este aspecto:



Point Light tem este aspecto:



PointLight

Esta class é a implementação da PointLight, esta class não tem nada de especial apenas define no seu construtor: this.type = 1; para que o seu tipo seja uma PointLight. Esta class não é necessária, antes pelo contrário é apenas facultativa para tornar o nosso trabalho mais fácil e para organização de código.

SpotLight

Esta class é a implementação da SpotLight, esta class não tem nada de especial apenas define no seu construtor: this.type = 2; para que o seu tipo seja uma SpotLight. Esta class não é necessária, antes pelo contrário é apenas facultativa para tornar o nosso trabalho mais fácil e para organização de código.

net.joaolourenco.legame.entity.light.specific

FireLight

Esta class é a implementação de uma FireLight, esta luz usa uma PointLight com variações de cor e intensidade para simular uma luz de fogo.

net.joaolourenco.legame.entity.mob

Mob

A Mob class é responsável por definir um modelo de Entity para entities que se mexam e/ou que tenha vida.

Esta class implementa ferramentas de AI como as MovementAction, AnimatedSprites entre muitas outras coisas para os mobs.

Methods importantes nesta class:

- checkCollision – Method que verifica a colisão entre a entity e os outro tiles e com outras entities.
- updateTexture – Method que actualiza a animação da entity.
- setTextureAtlas – Method que define a animação de andar da entity.
- setDyingAtlas – Method que define a animação de morte da entity.
- setAttackingAtlas – Method que define a animação de ataque da entity.
- Died – Method que indica o comportamento da entity ao morrer.

Citizen

Citizen é um mob passivo que usa o RandomTargetedMovementAction para andar a passear no mapa.

Este mob só serve para distrair o player.

Esta class não tem nenhuma method específica nem nenhuma variável especifica, apenas usa as implementações da Mob e Entity class para criar uma nova entity nova.

Este Mob pode e deve servir como modelo para a criação de novo Mobs pela sua incrível simplicidade.

Skeleton

Skeleton é um mob agressivo que usa o PersistentTargetedMovementAction para andar a atras do player.

Este Mob tem metade da velocidade do player no entanto tem um attack range de 100 o que são quase dois tiles.

O skeleton só nos consegue seguir se tivermos uma área de 800x800 á volta dele ou seja numa área de cerca de 12x12 tiles.

Novamente este mob não usa nenhuma method especifica nem nenhuma variável especifica, apenas usa as implementações da Mob e Entity class para criar uma nova entity nova.

Spider

Spider é um mob agressivo que usa o PersistentTargetedMovementAction para andar a atras do player tal como o Skeleton.

Este Mob tem a velocidade do player no entanto tem um attack range de 50 o que não chega a ser um tile.

A spider só nos consegue seguir se tivermos uma área de 400x400 á volta dela ou seja numa área de cerca de 6x6 tiles.

Novamente este mob não usa nenhuma method especifica nem nenhuma variável especifica, apenas usa as implementações da Mob e Entity class para criar uma nova entity nova.

Player

Player é a class responsável pelo player que tal como todos os outros mobs não têm qualquer tipo de methods ou variáveis extra.

AnimatedSprite

A class AnimatedSprite serve como gestor de texturas para criar uma animação.

Esta class permite-nos definir um set ilimitado de texturas como uma só e ao efectuar os updates á mesma esta vai avançando textura a textura criando assim uma animação.

Methods importantes:

- setFrameRate – Esta method define o numero de updates que a textura sofre. Quanto maior o número, menor o número de updates que a textura sofre.
- getFrame – Obtém o frame na animação actual.
- Update – Actualiza a textura.
- resetAnimation – Volta ao frame 0 da animação.
- getTexture – Obtém a textura do frame actual.

É possível criar um novo AnimatedSprite da seguinte maneira:

```
new AnimatedSprite(Array de Texturas, comprimento da animação em frames, frame default);
```

RenderableComponent

Esta class contém todas as methods de render do jogo.

Todas as method têm o mesmo nome, render, as únicas coisas que mudam é os parâmetros e o que a method faz.

O primeiro render, usa immediate-mode e serve para fazer o render de quadrados simples e que não sejam muito requisitados.

O segundo render, usa VBO e serve para fazer o render de objectos com maior frequência,

O terceiro render, usa immediate-mode e serve apenas para o render dos menus.

O quarto render, usa immediate-mode e serve para fazer o render de entities e de alguns objectos do menu.

O quinto render, usa immediate-mode e serve para fazer o render de portas e de alguns objectos do menu, este render pode rodar objectos.

O sexto render, usa VBO e serve para fazer o render com altas frequências como os tiles.

Shader

Shader é uma class extremamente importante no projecto, a Shader class é responsável por carregar e compilar as shaders para a memória.

Methods importantes:

- recompile – Permite-nos recompilar o shader para testes sem ter de reiniciar o jogo.
- Bind – Permite-nos colocar o shader em uso.
- Release – Permite-nos libertar o shader.
- cleanUp – Permite-nos limpar a memoria para que não haja memory leaks.

Texture

Texture é a class responsável por todas as texturas do jogo, esta class consegue fazer o load de texture atlas e de texture e também as armazena.

Methods importantes:

- preload – Esta method faz o pre load de texturas que são maioritariamente as texturas dos menus.
- Load – Esta method faz o full load de todas as texturas para que possam ser usadas no jogo.
- loadTexture – Faz o load de uma só textura sem transparência.
- loadAtlas – Faz o load de Atlas e converte-os para texturas individuais. Suporta transparência.

Exemplos de load:

```
Menus = loadTexture("/textures/menus/textura.png", false);

Menus = loadAtlas("/textures/menus/textureAtlas.png", 3, 3);
```

loadAtlas requer como argumentos extra, o numero de texturas em x e em y.

net.joaolourenco.legame.graphics.font

Font

Font é a class que faz todo o processamento das letras para se fazer o render das mesmas.

Esta class contém o seu próprio shader e o seu próprio render, por estes motivos torna-a mais pesada e para evitar o excesso de instâncias está registada na Registry class.

Como registar uma font na Registry class:

```
Registry.registerFont(new Font());
```

Como obter a font já registada:

```
Registry.getFont();
```

Methods Importantes:

- getStringSize – Esta method actua como um simulador de render mas em vez de fazer o render apenas contabiliza o comprimento da String.
- drawString – Esta method é a responsável para o render das strings.
- drawString – Esta method é a responsável para o render das strings e pode ser usar cores.

AnimatedTex

AnimatedText é uma class que gere os textos animados, isto é, todos os textos que vão aparecendo gradualmente e que desaparecem passado um X de tempo.

Esta class contém um alto número de constructors para satisfazer as nossas necessidades em qualquer sítio.

Para criar um novo AnimatedText apenas é preciso fazer:

```
new AnimatedText(“A testar o texto”, x, y);
```

Isto vai criar um texto animado com todas as definições standard. Podemos ainda se desejarmos, usar os construtores para alterar estas mesmas settings.

Methods importantes:

- getRenderTiming – Quanto tempo vai durar até todas as letras estarem rendered.
- getTotalTiming – Quanto tempo vai durar ao texto até desaparecer.
- getDelayTime – Quanto tempo de delay tem antes de começar a ser feito o render.

Utilizando um excerto do código do tutorial podemos observar que ao enviar um AnimatedText a outro AnimatedText o que vai acontecer é que o segundo só vai começar quando o primeiro acabar. Assim criar uma ilusão de sequência de texto perfeita.

```
AnimatedText a = new AnimatedText("This game has a simple objective.", x, y, 25, 100, 200, -1);

new AnimatedText("Get to the end of each level. ALIVE!", x, y, 25, 100, 200, -5, a);
```

StaticText

StaticText é uma class para criar textos imediatos e sem animações, estes textos vão se manter no ecrã até serem removidos pelo programador manualmente.

Para adicionar um basta:

```
Registry.registerStaticText(new StaticText(“Texto”, x, y, tamanho));
```

Para remover:

```
Registry.clearStaticTexts();
```

net.joaolourenco.legame.graphics.menu

Menu

Menu class é mais uma abstract class que funciona como class Modelo para os menus.

Esta class contém o render, update e tick abstract para que possam ser criados para outros menus.

Para abrir um menu qualquer basta:

```
Registry.registerMenu(new MainMenu());
```

Methods importantes:

- Open – Abre o menu em questão.
- Close – Fecha o menu em questão.
- generateRandom – Esta method serve para criar uma random de ints ou floats dentro de um intervalo predefinido. Esta função encontra-se em várias classes.

Loading

Loading class usa a interface disponibilizada pela Menu class para criar um menu de loading.

Usa uma array de MenuClouds para o topo do seu menu e uma imagem rotativa para o loading.

MainMenu

MainMenu class usa a interface disponibilizada pela Menu class tal como a Loading class para criar o menu Principal.

Usa uma array de MenuClouds para o topo do seu menu e não implementa qualquer tipo de methods extras.

OptionsMenu

OptionsMenu tal como os outros Menus não se altera muito para criar o menu apenas usa as methods da Menu class para criar o seu menu e mais uma chamada RecreateWindow que serve para recriar a janela usando as novas settings em uso.

Nesta method de RecreateWindow o Display é destruído, aplicadas as novas definições de janela, criado um novo Display, recriada a Font, Projecções e em seguida todas as shaders e texturas são recarregadas.

net.joaolourenco.legame.menu.objects

ClickAction

ClickAction é apenas uma interface para um callback de quando um botão é premido.

MenuCloud

MenuCloud é o elemento responsável pelo render das nuvens nos menus. Esta class não executa qualquer tipo de callbacks nem verifica nada nenhuma acção em específico.

Esta class usa uma Função render para desenhar as nuvens e uma função update para mover as nuvens.

Esta class contém todos os getters e setters necessários ao seu funcionamento.

MenuActionReader

MenuActionReader é a class modelo responsável pela, criação e actualização dos botões para os menus.

Esta class suporta múltiplos callbacks que podem ser chamados ou quando o botão é premido, ou quando libertamos o rato em cima do botão.

Para a criação de funções devemos usar sempre a ClickCallBacks que é quando o botão do rato é deprimido.

Variáveis:

- X, y – Localização do botão.
- xOffseted – Se o botão está deslocado ou não da sua localização.
- Width e Height – Tamanha do botão.
- Spacing – Espaçamento para as fonts.
- Size – Tamanho para as fonts.
- screenHeight – Altura do ecrã para ser usado em cálculos.
- Text – Texto que vai ser desenhado no ecrã.
- menuOwner – A que menu pertence este botão.
- Font – Uma instância da font class usada para fazer o render de texto se necessário.
- Color – Cor inicial do botão.
- sColor – Cor quando estamos com o rato em cima da cor.
- pColor – Cor quando premimos o botão.
- dColor – Cor quando o botão está desactivado.
- cColor – Cor activa no momento do render.
- Selected – Se o botão está seleccionado ou não.
- Mouse – Se o botão contêm o focus do rato.
- Enabled – Se o botão está activo ou não.

Methods importantes:

- addMouseDownAction – Method para adicionar callbacks.
- addClickAction – Method que deve ser usada para adicionar callbacks para efectuar acções no botões.

MenuButton

MenuButton é uma implementação da Class Modelo MenuActionReader. Este MenuButton não implementa nenhuma method em especial, apenas usa as fornecidas pelo MenuActionReader.

No update o MenuButton verifica se temos o rato em cima do botão ou não e se clicamos nele ou não.

Se clicarmos vão ser efectuadas todos os callbacks registados.

MenuCheckBox

MenuCheckBox é uma implementação da Class Modelo MenuActionReader. Esta class tal como a MenuButton não implementa nenhuma method em especial, apenas usa as fornecidas pelo MenuActionReader. O MenuCheckBox é como uma CheckBox no VB, pode estar com um tick ou não.

Apesar de ter callbacks estes servem apenas para adicionar efeitos especiais. Para obter se está seleccionada ou não deve se fazer:

```
Checkbox.isSelected();
```

MenuOptionSelect

MenuOptionSelect é uma implementação da Class Modelo MenuActionReader para a criação de um set de botões com um display no meio para que se possa fazer a escolha de alguma coisa.

Este botão contém dois MenuButtons para efectuar o slide e ainda um font render para mostrar a opção escolhida no momento.

Este botão funciona quase como um DropDown Menu mas na horizontal.

Methods importantes:

- addOption – Adicionar opções.
- setActive – Definir um elemento como activo.
- getSelected – Optem o valor do elemento activo.

MenuSlider

MenuSlider é uma implementação da Class Modelo MenuActionReader para a criação de um slider button. Um bom exemplo de um Slider button é o volume do windows.

Esta class não usa nenhum elemento extra para a sua criação.

Methods importantes:

- getValue – Obter o valor da slider.
- setPosition –Definir uma posição default para o slider.

net.joaolourenco.legame.items

Item

Item é uma class abstract que funciona como class Modelo para todos os items.

Esta class contém as seguintes variáveis:

- Name – Nome do item.
- current_life – O numero de vezes que a chave ainda pode ser usada.
- Inicial_life – O número de vezes que a chave podia ser usada no início.
- Destructable – Se o item é destruível ou não.

Para além dos getters e dos setters esta class apenas contém quatro funções importantes:

- useItem – Method chamada quando o item é usado.
- itemBroken – Method chamada quando o item se parte.
- itemBreak – Method chamada para confirmar se o item se parte ou não.

DoorKey

DoorKey é um exemplo de um item simples que pode ser criado a partir da class Modelo Item.

Esta class só implementa uma method para verificar se é possível abrir a porta em questão, de resto apenas usa a interface fornecida pela class Item.

net.joaolourenco.legame.settings

GeneralSettings

GeneralSettings é uma class sem methods, apenas variáveis que possam ser usadas no jogo todo.

Variáveis:

- version – Versão do jogo.
- Name – Nome do jogo.
- Fullname – Junção do nome e da versão.
- TILE_SIZE – Tamanho dos tiles.
- TILE_SIZE_MASK – O bit operator correspondente ao TILE_SIZE.
- useAverageFPS – Flag para usar media de FPS ou não.
- ticksPerAverage – Quantos ticks para obter uma media.
- showLightFloat – Mostrar o nível de luz no mundo.
- defaultEntityWalking – Velocidade do player a andar.
- defaultEntityRunning – Velocidade do player a correr.
- defaultLightPointSize – Tamanho default do ponto das luzes.
- defaultLightSize – Tamanho default das luzes.
- defaultLightFacing – Rotação default das luzes.

As restantes variáveis são as localizações dos shaders para serem compilados.

Settings_Key

Settings_Key é uma class que serve como objecto para guardar as settings do jogo.

Variáveis:

- key – Nome da setting a guardar.
- Value – Valor da setting guardar.

Settings

Settings é a class responsável pela gestão e organização dos ficheiros de settings.

Methods:

- SettingsDefault – Settings default, só é usado quando não existem settings no sistema.
- SetsingsLoader – Carrega as settings do ficheiro settings.conf
- SettingWritter – Apaga se existir o ficheiro settings.conf e cria-o novamente com as novas settings.
- saveDOMSource – Method para guardar o mundo no ficheiro world.xml em formato XML.

net.joaolourenco.legame.utils

Buffers

OpenGL trabalha muito com buffers, por isso criei uma class inteira só para trabalhar com os buffers.

Methods:

- createFloatBuffer – Cria um buffer de floats.
- createByteBuffer – Cria um buffer de Bytes.
- createIntBuffer – Criar um buffer de ints.

KeyboardFilter

KeyboardFilter funciona como um filtro de teclado. Para evitar spam de teclas o KeyboardFilter só recebe input de 500 em 500 milissegundos.

Methods:

- isKeyDown – Method que verifica se a tecla está em baixo ou não sem spam.

Timer

Timer é uma class que funciona como um timer, tem o seu próprio thread para não deixar o jogo lento e de X em X de tempo e um X de vezes faz um callback para uma method registada.

Os callbacks são sempre efectuados em para TimerResult.

Para registar um novo timer podemos usar o seguinte comando:

```
new Timer("Nome”, TempoDoTimer, NumeroDeVezes, new TimerResult(this) {
        public void timerCall(String caller) {
                // Acção a executar.
        }
});
```

TimerResult

TimerResult é uma class apenas para callbacks e contêm apenas um objecto para ser usado se for preciso e a função timerCall é chamada sempre que o Timer termina.

Vector2f

Vector2f é uma class de gestão de vectores de floats. Grande parte do código da mesma vem do LWJGL.

VertexHandlers

VertexHandlers contêm os Vertex's para os VBOs.

Node

Node class é uma class usada apenas para guardar informações para o A* Search Algorithm.

Não contém qualquer tipo de methods para além do seu construtor.

World

World class é a class responsável por a maior parte dos updates, renders e ticks. Esta class é a class que guarda todas as entities e todos os tiles.

Variáveis:

- DAY_LIGHT – Luz ambiente presente no mundo.
- Entities – ArrayList para guardar todas as entities.
- worldTiles – Array para guardar todos os tiles.
- Width e Height - Tamanho no mapa.
- xOffset e yOffset – Variáveis responsáveis pelo scroll do mapa. Estas são as variáveis que fazer o ecrã ficar centrado no player.
- goingUp – Se está a ficar de dia ou de noite.
- gameOver e levelOver – Se o mundo terminou ou não.
- stopLoading – Se o mundo já está carregado ou não.
- Lost – Se o jogador perdeu ou ganhou.
- levelEnabled –Se o mundo já é jogável ou não.
- updatesReady – Se o mundo já pode fazer os seus updates normais.
- Ready – Se o Mundo já está pronto a ser jogável.
- Player – Instância da class player.
- Loading – Instancia do loading para ser aberto e fechado quando necessário.
- endMessage – Mensagem que aparece no fim do mundo.
- Level – O nível em que estamos.
- nodeSorter – Um comparador de Nodes para organizar uma array.

Quando um novo mundo é criado é feita uma sequência de chamadas para uma serie de methods seguindo esta ordem:

- Criação de uma nova loading menu.
- preLoad, onde se faz o load de todas as texturas do mapa.
- Criação de um novo player.
- Criação da worldTiles array de acordo com o tamanho definido.
- generateLevel, que vai gerar o mundo em geral.
- Após o generateLevel ter acabado, 1500 milissegundos depois, os updates são ligados e por fim 2000 milissegundos depois de ter acabado, o render é iniciado, assim o nível só é visível aos nossos olhos quando está pronto.

Esta class tem as methods de render, tick e update muito diferentes de qualquer outra class pois para alem de terem de se actualizar, têm de actualizar também todas as suas entities e tiles.

Ações do Render:

- O mapa é movido para a posição correcta com o glTranslatef().
- Usando o glColor3f() é certificado que não existe nenhuma cor especifica no render que venha a alterar as nossas cores de render.
- É calculado o canto superior esquerdo, x0, o superior direito, x1, e o inferior esquerdo, y0, e o direito, y1.
- Usando as variáveis calculadas é feito um for loop que vá de y0 a y1 e dentro deste um outro for loop que vá de x0 a x1.
- Se o tile em questão existir, este é renderizado nas coordenadas certas.
- Usando o glColor3f() novamente para certificado que não existe nenhuma cor especifica no render que venha a alterar as nossas cores de render.
- É usado outro for loop para correr a ArrayList das entities toda e é verificada se estas estão a menos de 800 pixels do player, se sim, é feito o seu render.

Ações do update:

- É verificado e o loading já acabou, se o jogo já acabou e/ou se os updates já estão ligados.
- Novamente um for loop para correr as entities todas e verificar se estão perto do player, se estiverem, actualiza-as.
- É também aqui que é feita a chamada para a função entityOver para o tile em que a entity está.
- Se o mundo não estiver terminado faz o update dos tiles e altera também a luminosidade do mundo.
- Se o mundo estiver terminado em vez de ser feito o update do mundo a luz é aumentada a uma velocidade muito superior e quando atingir valores inferiores a 0.1f, é adicionado o texto final no ecrã e o nível termina.

Ações do Tick :

- Verifica se as entities foram removidas ou não, se sim remover da memorias, se não fazer o tick das mesmas.

Methods importantes:

- levelEnd – Method chamada quando o nível é terminado mas ainda o nível ainda não foi removido.
- gameOver – Method que chama o que for para iniciar após o fim do mundo, esta é a ultima method a ser chamada antes do mundo ser destruído.
- getDistance – Esta class tem varias funções de getDistance para verificar a distancia entre dois pontos ou até mesmo entre duas entities.
- getNearByEntities – Obtêm as entities perto do player.
- setTile – Method para atribuir um Tile ao mundo.
- addEntity – Method para adicionar uma entity ao mundo.
- getTile – Method para obter um tile nas coordenadas pedidas por parâmetros.
- setSize – Method para alterar o tamanho do mundo. Esta method recria a array e por este motivo todos os tiles são perdidos.
- findPath – É a method on o A* Search Algorithm corre.
- saveWorld - É a method que guarda o mundo em formato XML.

RandomWorld

RandomWorld é a class responsável pela criação dos níveis em geral. Esta class não implementa nenhuma method extra para além das que o seu Modelo, World class, contêm.

Na Method generateLevel é feita a criação do nível, neste momento só existe um ou dois níveis para quando estiver acabada a class é suposto criar níveis random.

Tutorial

Tutorial é mais uma class que vêm da World class e é responsável por todo o tutorial.

A method principal nesta class é a changeStep que faz toda a troca de senários para o próximo passo do tutorial.

net.joaolourenco.legame.tile

FinishPoint

É uma class de apenas uma função que criar os 4 pontos de fim do mundo e é apenas para simplificação de código.

Tile

Tile pode ser comparado com os azulejos do chão da nossa casa mas para o nosso mundo. Esta class é uma class Modelo para outros Tiles e só implementa funções básicas.

SolidTile

SolidTile é a implementação do Tile, este não implementa nenhuma method extra apenas usa as já fornecidas pela class Modelo Tile.

FireTile

FireTile é um tile que pretende fazer-se passar de uma fogueira em conjunto com uma FireLight mas nunca funcionou lá muito bem por isso não está a ser usada.

FinishTile

FinishTile é o tile que marca o fim do mundo, este, tal como todos os outros, não implementa mais nenhuma method especial apenas usa as que lhe são oferecidas pela class Modelo Tile.

Esta class usa a method, entityOver, para verificar se o player está em cima de si. Se estiver vai iniciar um timer de 500 milissegundos para terminar o nível. Se não estiver nada acontece.

Resources

Shaders

Esta pasta contém todos os shaders para o jogo escritos em GLSL por mim.

A Shader class é o que as gere e as compila para a memória do jogo.

Ficheiros:

- backmenu.frag – Fragment Shader para o fundo do menu.
- block.frag – Fragment Shader para os tiles.
- default.vert – Vertex Shader para todos os Fragment Shaders no jogo.
- font.frag – Fragment Shader para as fonts.
- light.frag – Fragment Shader para as luzes todas.
- menu.frag – Fragment Shader para os menus.

Textures

Esta pasta contém todas as texturas espalhadas por várias pastas:

- Menus – Todas as texturas para os menus.
- Mobs – Todas as texturas para os Mobs.
- Testing – Todas as texturas de testes.
- Util – Todas as texturas extras como fonts.
- Directórios principais - encontram-se as texturas de tiles.

Library

Directório principal

Neste directório encontram-se os jars do lwjgl que nos permite programar em OpenGL.

LWJGL quer dizer Light Weight Java Game Library.

Esta lib necessita de Dll's ou também chamadas de natives que se encontrão na pasta windows.

Conclusão

Em suma, o projecto apesar de não estar totalmente completo, encontra se numa versão estável para ser usado como engine para outros projectos.

Consegui com este projecto o que desejava, alcancei as minhas visões e consegui escrever friendly code.

Este projecto é open source e encontra se no meu github sobre uma licença Apache 2.0.

https://github.com/joaogl/LeGame-Remake

Eu filmei a criação do jogo e existem duas versões do mesmo, uma tem cerca de 5 min, outra 11 min. Contêm o mesmo apenas um está mais acelareado que o outro.

http://joaolourenco.net/LeGame.mp4

http://joaolourenco.net/LeGameRapido.mp4

Termos – Dicionário

- Game loop – É um loop finito que é usado em jogos pois contêm temporizadores e é o que efectua as chamadas para as funções de render, update e tick. É também com o game loop que podemos observar os nossos FPS.
- Frame – Frame é o nome que se dá á imagem actualmente presente no ecrã.
- Update – Update é o nome que se dá á chamada para actualização da parte lógica do jogo.
- FPS ou Frames per second - É o numero de frames desenhados por uma aplicação por segundo.
- UPS ou Updates per second – É o número de updates lógicos que a nossa aplicação recebe por segundo, por norma este número deve ser estável.
- Delta – Delta é o tempo em milissegundos entre dois updates, o delta é usado para uniformizar o movimento das entities para que estejamos num computador lento ou rápido o movimento seja sempre o mesmo.
- Entity – Entidade que pode ou não ter vida e por ou não mexer-se, mas uma delas tem de ter.
- Mob – Mobs são as entidades que têm vida como, monstros e players.
- Shaders – Shader ou Shader program é um sub programa criado em GLSL (linguagem do OpenGL) e é uma forma de programar gráficos a um nível muito mais profundo e especifico para a placa gráfica, assim as velocidades de render são superiores e o controlo sobre efeitos gráficos superior. Um Shader program é normalmente constituído por um Vertex Shader e um Fragment Shader.
- Vertex Shader – Vertex shader é o shader responsável por controlar a posição de cada pixel.
- Fragment Shader – Fragment shader é o shader responsável pela cor de cada pixel em especifico, no fragment shader não é possível aceder ao pixel do lado, apenas ao pixel em que nos encontramos.
- DisplayMode – DisplayMode é o objecto que contem as informações como a resolução em uso, se está ou não em fullscreen, o bit rate e os bits per pixel.
- VBO – Vertex Buffer Object é uma funcionalidade do OpenGL que permite carregar os dados dos indices, vértices e etc. para a memória gráfica. Usa um non-immediate-mode rendering method pois usa a memória da gráfica o que a torna muito mais rápida que a immediate-mode rendering.
- A* Seach Algorithm – É um algoritmo de procura e processamento de caminho entre dois pontos ou nodes.
- Point Light – Uma luz que ilumina tudo á sua volta como por exemplo uma lâmpada normal.
- Spot Light – Uma luz que ilumina para onde está apontada como por exemplo uma lanterna.
- Texture Atlas – Um conjunto de várias texturas numa só textura seguindo um padrão 2D (Linhas e Colunas)
- Anti-Aliase – Anti Aliase é um sistema usado por jogos que torna os quadrados menos quadrados, assim disfarça as resoluções mais baixas ou os níveis de qualidade inferiores. Nota-se uma maior diferença nos cantos de faces á distancia.
- Display – Janela onde o jogo está a correr. Se destruído tudo é apagado, incluindo texturas e shader programs.
- Getter – Method que retorna uma variável de uma class sem executar qualquer tipo de código.
- Setter – Method que altera uma variável de uma class sem executar qualquer tipo de código.
- Buffer – São objectos do OpenL que guardam informação não formatada na GPU.
- GPU – Graphics Processing Unit ou Componente gráfica do computador.
- Callback – Quando uma função é passada como parâmetro para ser mais tarde chamada.