Pedro Paulo Vezzá Campos Daniel Moraes Huguenin Antonio Rui Castro Junior

Guerra Das Universidades: Manual do Desenvolvedor

São Paulo - SP, Brasil 20 de dezembro de 2011

Pedro Paulo Vezzá Campos Daniel Moraes Huguenin Antonio Rui Castro Junior

Guerra Das Universidades: Manual do Desenvolvedor

Manual do desenvolvedor do projeto desenvolvido pela Equipe Knuth apresentado para avaliação na disciplina MAC0242, do curso de Bacharelado em Ciência da Computação, turma 45, da Universidade de São Paulo, ministrada pelo professor Roberto Hirata Junior.

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO INSTITUTO DE MATEMÁTICA E ESTATÍSTICA UNIVERSIDADE DE SÃO PAULO

> São Paulo - SP, Brasil 20 de dezembro de 2011

Sumário

1	Intro	odução	p. 4
2	Croi	nologia	p. 5
	2.1	Fase 0 - Especificação	p. 5
		2.1.1 Decisões Tomadas	p. 5
		2.1.2 Dificuldades Enfrentadas	p. 5
	2.2	Fase 1 - Protótipo	p. 6
		2.2.1 Decisões Tomadas	p. 6
		2.2.2 Dificuldades Enfrentadas	p. 6
	2.3	Fase 2 - Continuação	p. 7
	2.4	Fase 3 - Implementação do Modelo Lógico	p. 7
		2.4.1 Decisões Tomadas	p. 7
		2.4.2 Dificuldades Enfrentadas	p. 7
	2.5	Fase 4 - Documentação	p. 8
		2.5.1 Decisões Tomadas	p. 8
		2.5.2 Dificuldades Encontradas	p. 8
3	Orga	anização do Código	p. 9
	3.1	Modelo Lógico	_
	3.2	Visão	-
	3.3	Controle	
	3.4	Testes	•
	J.⊤	10000	ρ. 1 ⊓

	3.5 Utilitários	. p. 17
4	Documentação do Código	p. 18
Re	eferências Bibliográficas	p. 19

1 Introdução

Este relatório apresenta os desenvolvimentos e resultados dos trabalhos da Equipe Knuth - Pedro Paulo Vezzá Campos, Daniel Moraes Huguenin e Antonio Rui Castro Junior que resultaram na produção do jogo **Guerra das Universidades**.

Instruções de compilação e execução estão disponíveis em [1]. Instruções de como jogar e a especificação do projeto estão disponíveis em [2]. O blog de desenvolvimento encontra-se em http://guerradasuniversidades.wordpress.com. O repositório de código está em http://code.google.com/p/guerradasuniversidades.

2 Cronologia

2.1 Fase 0 - Especificação

Para o projeto da disciplina MAC0242 - Laboratório de Programação II foi sugerido pelo professor Hirata a implementação de um jogo com temática universitária aos moldes de jogos famosos, tais como *Civilization* ou *SimCity*. Para isso, deveriam ser utilizandos os conceitos de Programação Orientada a Objetos (POO) vistos em aula, linguagem Java e um *framework* de desenvolvimento de jogos, inicialmente com a sugestão de utilização do PlayN [3].

2.1.1 Decisões Tomadas

Na fase 0 o grupo definiu através de *mockups* e descrição textual o que viria a ser posteriormente o Guerra Das Universidades. O jogo possui uma temática semelhante a *Age of Empires*, consistindo de um ambiente de *singleplayer* 2D com oponentes implementados como Inteligências Artificiais (IA). O objetivo do jogador, que assume o papel de reitor de uma universidade, é construir estruturas essenciais ao campus, recrutar alunos e professores para atacar universidades inimigas enquanto se defende de ataques de terceiros. Ganha a universidade que permanecer funcionando por mais tempo.

Ainda, foi criado o blog de desenvolvimento onde os alunos podem apresentar seus progressos e dificuldades no decorrer do desenvolvimento. O blog da Equipe Knuth encontra-se em: http://guerradasuniversidades.wordpress.com. Por fim, os manuais de desenvolvedor e usuário tiveram seus primeiros esboços produzidos.

2.1.2 Dificuldades Enfrentadas

Conscientes do curto período de tempo que haveria para o desenvolvimento completo do projeto a equipe decidiu por não tornar muito complexa a especificação, transformando funcionalidades supérfluas tais como recursos visuais mais complexos em opcionais a serem imple-

mentados caso haja tempo suficiente para isso.

Em contrapartida, houve um comprometimento mútuo de **preservar ao máximo a quali- dade do código produzido**, o que trouxe posteriormente um aumento de legibilidade, robustez e manutenibilidade do código.

2.2 Fase 1 - Protótipo

Na fase 1 foi proposto que os alunos codificassem um protótipo do jogo para provar que o jogo é viável do ponto de vista de dificuldade de implementação e jogabilidade.

2.2.1 Decisões Tomadas

O grupo decidiu apostar no uso do PlayN como framework para o desenvolvimento do jogo por sugestão do professor e pela proposta ambiciosa dos desenvolvedores do arcabouço de prover uma ferramenta que torne o desenvolvimento de jogos agnóstico à plataforma a ser utilizada seja ela Java, HTML5, Flash ou Android.

Uma vez decidido isso, procedeu-se com a codificação quase total da interface gráfica do jogo sem nenhum tipo de modelo lógico associado. Com isso, os alunos tomaram maior familiaridade com as ferramentas utilizadas no processo e adiantaram uma parte importante do desenvolvimento do projeto.

O protótipo gerado pode ser acessado em: http://pedrovc-playn-test.appspot.com

2.2.2 Dificuldades Enfrentadas

A grande dificuldade enfrentada neste momento foi dominar o PlayN. O framework é bastante recente e possui uma documentação **extremamente** escassa. A comunidade de desenvolvedores está ainda se formando tornando fraca a oferta de material de estudo como fóruns, tutoriais ou manuais. O aprendizado concentrou-se nos exemplos fornecidos juntamente com o PlayN e bastante experimentação.

Exemplos de problemas com o PlayN (E sua biblioteca auxiliar a tripleplay): A pobreza de recursos como caixas de diálogo e disposição de elementos graficamente na tela, problemas com a translação e atualização de layers com botões clicáveis (Enfrentados na implementação do scroll de itens que podem ser comprados).

Ainda, outra dificuldade foi coordenar o trabalho remoto dos integrantes devido aos diversos conflitos de horário, entre estágios e matérias extras. Nos reunimos poucas vezes (mas foram bem produtivas). Neste momento, o uso de um repositório de código compartilhado foi crucial para permitir o progresso do trabalho.

2.3 Fase 2 - Continuação

Devido a questões de cronograma esta fase foi suprimida da agenda de entregas a ser respeitada pelas equipes.

2.4 Fase 3 - Implementação do Modelo Lógico

Nesta fase foi pedido que as equipes tenham implementado completamente seus jogos.

2.4.1 Decisões Tomadas

Uma vez que as especificações foram melhor delineadas na fase 1, procedeu-se com a implementação do modelo lógico de maneira isolada da interface gráfica. Para isso, utilizou-se de uma interface em modo texto que simula as ações permitidas pelo jogo.

Paralelamente, foram produzidos os testes unitários do modelo lógico utilizando a biblioteca JUnit para garantir seu bom funcionamento. Para a parte gráfica, os testes foram realizados visualmente e de maneira manual.

Em seguida foi realizada a junção da interface com o modelo. Neste momento foram necessários pequenos ajustes na interface para adequar-se ao funcionamento do modelo, além disso foram aperfeiçoadas as classes controle do jogo, configurando um projeto MVC [4].

2.4.2 Dificuldades Enfrentadas

A implementação do modelo em si não trouxe grandes dificuldades exceto os cuidados para manter o código dentro dos princípios da orientação a objetos. Foi necessário o uso de algumas design patterns como **Observador-Observado**, **Fachada** e **MVC** para atingir este objetivo. Isto está detalhado posteriormente na seção 4.

Outra dificuldade foi o fato do GWT não disponibilizar o uso das classes Observer e Observable do Java ao ser feito o deploy para o Google AppEngine. Isso foi resolvido com o uso

de uma implementação livre destas classes sob a GPL.

2.5 Fase 4 - Documentação

Para a fase final do projeto foi requisitado que os alunos entregassem seus projetos finalizados e documentados para avaliação.

2.5.1 Decisões Tomadas

Nesta fase a equipe decidiu documentar o código utilizando a sintaxe Javadoc o que permitiu a geração automatizada de documentação acessível via navegador web. O resultado encontra-se no diretório Documentação do arquivo compactado junto com os fontes do projeto, manuais, etc.

Ainda, foram feitos ajustes finos na interface que haviam sido deixados incompletos em fases anteriores.

Por fim, foram completos os manuais de usuário, desenvolvedor e compilação/execução.

2.5.2 Dificuldades Encontradas

A dificuldade enfrentada neste momento foi enfrentar o trabalho de documentar um projeto que totaliza aproximadamente 3000 linhas de código entre modelo e interface.

3 Organização do Código

Devido ao modelo MVC adotado no projeto serão apresentados os detalhes de implementação de cada uma das partes separadamente. O código possui 7 pacotes, sendo eles:

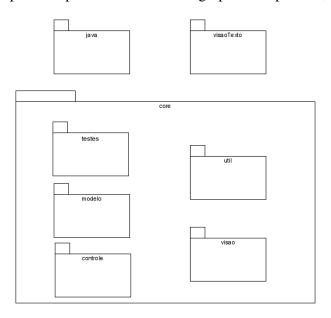


Figura 3.1: Diagrama de pacotes do projeto

com.mac242.guerradasuniversidades.java Pacote gerado pelo PlayN, contém apenas uma classe, a GuerraDasUniversidadesJava, responsável por realizar a ligação entre a implementação agnóstica de plataforma (Pacote core) e a implementação Java.

com.mac242.guerradasuniversidades.core.modelo Pacote contendo o modelo lógico do jogo com.mac242.guerradasuniversidades.core.visao Pacote contendo a interface gráfica implementada com o PlayN

com.mac242.guerradasuniversidades.core.controle Pacote contendo o controle da interface gráfica, também implementada com o PlayN

com.mac242.guerradasuniversidades.core.testes Testes unitários do modelo.

com.mac242.guerradasuniversidades.core.util Classes utilitárias não relacionadas ao projeto diretamente.

com.mac242.guerradasuniversidades.visaoTexto Pacote contendo uma interface em modo texto.

3.1 Modelo Lógico

O funcionamento do jogo é bastante centrado nas capacidades e ações de cada jogador interagindo entre si, sendo necessário apenas uma gerência dessas interações. Como consequência, o código apresenta como classes principais as classes GuerraDasUniversidades e Jogador.

A classe GuerraDasUniversidades é responsável por todos os eventos de responsabilidade do jogo como um todo, sendo uma **Fachada** [5] para o jogo em si. Suas atribuições incluem: Contagem do tempo, inicialização dos jogadores e ser o informante de eventos ocorridos no jogo através do padrão Observador-Observado.

A definição das atribuições de um jogador genérico é definida através da interface Tipo Jogador. Na implementação final duas classes implementam tal interface: Jogador e Jogador Maquina correspondendo a

A classe Jogador cuida das particularidades de um único jogador, tais como seus PE, FO, maxFO, maxPE, etc. A única exceção é a gerência das estruturas compradas ou compráveis, que fica a cargo do GerenteEstruturas. Como cada estrutura possui sua particularidade e para preservar a extensibilidade futura a equipe preferiu que cada estrutura tenha seu próprio método de compra e destruição no GerenteEstruturas.

JogadorMaquina é responsável por implementar a lógica de inteligência artifical dos jogadores oponentes no jogo, o que é feito no método realizarJogada(). Além desta atribuição ela deveria reimplementar os métodos de Jogador que não foram modificados. Para evitar a repetição de código e evitar os problemas da herança tradicional de código foi realizada uma herança através de composição, JogadorMaquina delega a uma instância de Jogador interna as tarefas de controle das informações do jogador.

Todas as classes que implementam TipoJogador e o GerenteEstruturas são internas ao jogo. Por este motivo, elas trabalham segundo o modelo de programação por contrato [6]. A API de um jogador, que implementa todos os tratamentos de dados é chamada FachadaJogador e pode ser obtida por meio do GuerraDasUniversidades.

Diversos enums foram utilizados ao longo do modelo para representar diversas informações

constantes ao longo do jogo: FocoAdministracao trata do do enfoque escolhido pelo jogador (Humanas, Biológicas, etc.). NomeUniversidade inclui o nome das universidades que o jogo reconhece. Além disso é utilizado como identificador único de um jogador. ResultadoAtaque representa os diferentes resultados de um ataque, de Sucesso a falta de FO, falta de uma sala completa etc. TipoNotificacao representa os diferentes tipos de eventos que acontecem dentro do jogo e que são informados a observadores externos. Estrutura representa todas as estruturas e acontecimentos no jogo que podem ser comprados.

Por fim, as classes de status Status Jogador e Status SalaAula são objetos simples e descartáveis que representam informações instantâneas de um jogador ou sala de aula.

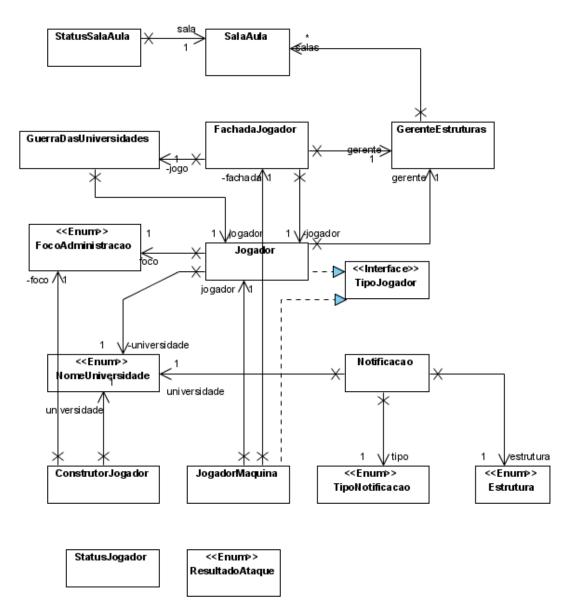


Figura 3.2: Diagrama de classes simplificado do modelo lógico

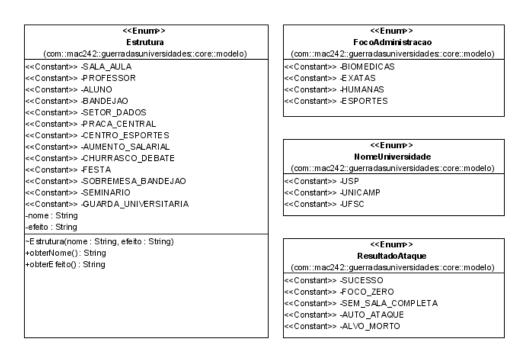


Figura 3.3: Enums do modelo lógico

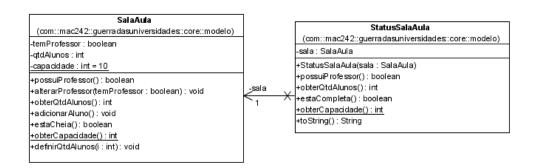


Figura 3.4: Classes representantes de uma sala de aula

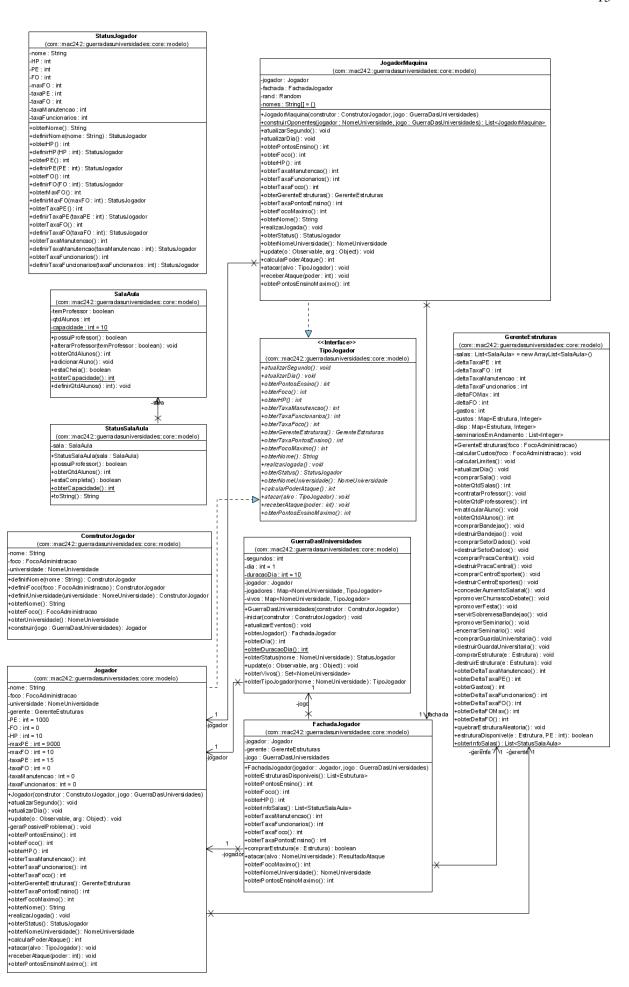


Figura 3.5: Diagrama de classes do modelo lógico

3.2 Visão

A interface gráfica do jogo é composta basicamente de telas, cada uma delas extende a classe abstrata TipoTela, que contém métodos utilitários comuns a todas as telas como rotinas de inicialização, atualização e encerramento de telas e estilos de botões.

O funcionamento de uma tela é baseado em quatro métodos:

init() Inicialização das layers da tela incluindo a base, que deve todas as outras layers da tela e que será eliminada ao final da exibição dela.

update(int delta) Atualização da lógica da tela, chamado regularmente a cada 10ms.

paint(int alpha) Renderização da tela, chamado o máximo de vezes possível pelo framework.

shutdown() Destruição da layer base e quaisquer outras estruturas alocadas pela tela.

Cada tela está instanciada apenas uma vez no jogo e essa instância pode ser acessada a qualquer momento através da classe VisaoGuerraUniversidades. Esta classe é responsável é a fachada da visão do jogo sendo responsável por trocar a tela atual e delegar eventos de cliques de mouse aos tratadores responsáveis. As telas do jogo são: Iniciar, Creditos, Ajuda, KonamiCode, Recordes, Menu, Opcoes, FimJogo.

3.3 Controle

No pacote com.mac242.guerradasuniversidades.core.controle encontram-se as classes tratadoras de cliques em diferentes tipos de itens e com diferentes propósitos. Todos eles realizam a tradução entre intenções do jogador na interface em ações interpretáveis pelo modelo. Estão implementadas as seguintes classes: TratadorBotaoBlocoEnsino e TratadorBotaoEstrutura responsáveis pela invocação de ações de compra de estruturas. TratadorTrocarTela responsável por invocar a VisaoGuerraDasUniversidades para realizar a troca de tela. TratadorAtacarOponer responsável por invocar ataques a oponentes no jogo.

3.4 Testes

Foram implementados testes unitários utilizando a biblioteca JUnit para garantir o bom funcionamento das classes do modelo. As classes de testes encontram-se em com.mac 242.guerradas univers

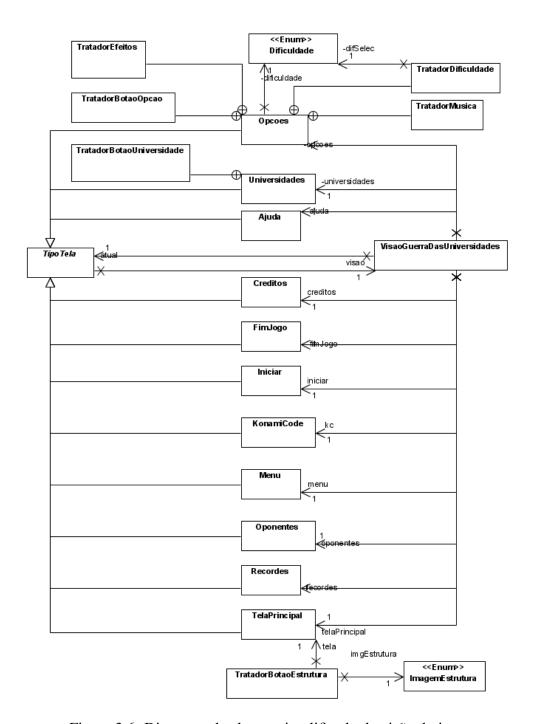


Figura 3.6: Diagrama de classes simplificado da visão do jogo

Creditos (com::mac242::guerradasuniversidades::core::visao) -iface: Interface +Creditos(jogo: VisaoGuerraDasUniversidades) +init(): void -desenharRecordes(): void -desenharVoltar(): void +update(delta: float): void +paint(alpha: float): void +shutdown(): void +toString(): String

Ajuda (com::mac242::guerradasuniversidades::core::visao) -iface: Interface +Ajuda(gog: VisaoGuerraDasUniversidades) +init(): void -desenhar Ajuda(): void -desenhar Ajuda(): void +update(delta::float): void +paint(ajpha::float): void +baint(ajpha::float): void +toString(): String

KonamiCode (com::mac242::guerradasuniversidades::core::visao) -konami::int[] = (38, 38, 40, 40, 37, 39, 37, 39, 66, 65) -estado::int = 0 +KonamiCode(jogo::VisaoGuerraDasUniversidades) -pressionarTecla(teda::int)::boolean +int()::void -desenharKonamiCode()::void +paint(alpha::float):void +update(defa::float):void +shutdown()::void

```
Recordes
(com::mac242::guerradasuniversidades::core::visao)
-iface : Interface
+Recordes()go : VisaoGuerraDasUniversidades)
+init() : void
-desenhar/Recordes() : void
-desenhar/Voltar() : void
+update(delta : float) : void
+paint(alpha : float) : void
+baint(alpha : float) : void
+to String() : String
```

```
VisaoGuerraDasUniversidades
 (com::mac242::guerradasuniversidades::core::visao)
atual : TipoTela
-menu : Menu
opcoes: Opcoes
recordes: Recordes
-kc : KonamiCode
iniciar : Iniciar
-universidades : Universidades
-telaPrincipal : TelaPrincipal
ajuda : Ajuda
-ajuda : Ajuda
-creditos: Creditos
-oponentes: Oponentes
-fimJogo: FimJogo
 contaUpdates:int
-jogo: GuerraDasUniversidades
construtor: ConstrutorJogador
-VisaoGuerraDasUniversidades()
+obterConstrutor(): ConstrutorJogador
+iniciarJogo(): void
+obterJogo(): GuerraDasUniversidades
 obterJogador():FachadaJogador
+init(): void
+rnii() : void
-tratarTeclado() : void
+exibirTela(tela : TipoTela) : void
+update(delta : float) : void
+paint(alpha:float):void
-updateRate():int
-obterMenu():Menu
+obterOpcoes(): Opcoes
+obterRecordes(): Recordes
+obterIniciar(): Iniciar
+obterUniversidades(): Universidades
-obterTelaPrincipal(): TelaPrincipal
+obterAjuda(): Ajuda
+obterCreditos(): Creditos
+obterOponentes(): Oponentes
 obterFimJogo(): FimJogo
```

```
Menu
(com::mac242::guerra dasuniversidades::core::visao)
-iface: Interface
+Menu(jogo: VisaoGuerraDasUniversidades)
+inif(): void
-desenharMenu(): void
-inidalizarInterface(): Root
-desenharLogo(): void
+update(delta::float): void
+paint(alpha::float): void
+shutdown(): void
+to String(): String
```

	TelaPrincipal
(com	::mac242::guerradasuniversidades::core::visao)
-iface : Interfac	-
-canvasFO : C	anvas
-canvasTaxas	: Canvas
-canvasCampu	us : Canvas
-canvasDia : C	anvas
-canvas Avisos	: Canvas
-canvasHP : C	anvas
-canvasSalas/	Aula : Canvas
-canvasPE : C	an vas
-estruturas : M	ap <imagemestrutura, boolean=""></imagemestrutura,>
-fontes : Map<	Float, TextFormat>
-posicaoMenu	: int = 0
-botoes : List<	Group>
-esquerda : Bu	itton
-direita : Buttor	1
-root : Root	
-avisos:List<	String> = newLinkedList <string>()</string>
+TelaPrincipal	(visao: VisaoGuerraDasUniversidades)
+init(): void	
-inicializarBarr	aSuperior(): void
-inicializarEstru	utura(img: ImagemEstrutura): Group
-inicializarEstru	utura (img: Imagem Estrutura, tratador: UnitSlot): Group
+atualizarCam	pus(): void
-atualizarTaxa:	s(): void
-atualizarFO()	: void
-atualizarPE():	void
+a dicionar Avis	o(aviso: String): void
+exibirAvisos()): void
-atualizarDia()	: void
-atualizarHP()	: void
-inicializarSala	sAula(): void
-atualizarBarra	Superior(): void
-atualizarSalas	sAula(): void
-desenharLogo	o(): void
+desenharTela	aFundo(): void
-inicializarCan	vas(I:int, h:int, x:int, y:int): Canvas
	o(texto: String, tam anho: float, canvas: Canvas): void
+update(delta	
+update(arg0:	Observable, arg1: Object): void
+paint(alpha:	
+shutdown():	
+toString(): St	
	as():Map <imagemestrutura,boolean></imagemestrutura,boolean>

	Universidades
(0	om::mac242::guerradasuniversidades::core::visao)
-ifac	ce : Interface
+to:	String(): String
+Ur	niversidades(jogo: VisaoGuerraDasUniversidades)
+ini	t(): void
-de	senharMenu(): void
-de	senharMapa(): void
+up	date(delta: float): void
+ра	int(alpha:float):void
+sh	utdown(): void

```
Oponentes
(com::mac242::guerra dasuniversidades::core::visao)
-iface: Interface
-legendas: M ap<Nom eUniversidade, Button>
+Oponentes(jogo: VisaoGuerraDasUniversidades)
+inft(): void
-exibirOponentes(): void
-gerarin foOponente(nome: Nom eUniversidade): Group-atualizarInfoOponente(nome: NomeUniversidade): void
+update(delta: float): void
+paint(alpha: float): void
+shutdown(): void
+toString(): String
```

```
(com::mac242::guerradas...niversidades::core::visao)
-iface : Interface
-dificuldade : Dificuldade = Dificuldade MEDIO
-musica : boolean = true
-efeto s: boolean = true
-efeto dificuldadeAtual : Button
-musica Atual : Button
-efeto Atual : Button
-efet
```

	FimJogo
(com∷mac2	42::guerradasuniversidades::core::visao)
-iface : Interfa	ace
-imagem : Str	ing
+FimJogo(jog	go: VisaoGuerraDasUniversidades)
+exibir∀ntonia	(): void
+exibirDemota	a(): void
+init(): void	
-desenharFim	nJogo() : void
+update(delta	a:float):void
+paint(alpha:	: float): void
+shutdown():	void
+toString(): S	String

	Tipo Tela
	(com::mac242::guerradasuniversidades::core::visao)
#v	risao: VisaoGuerraDasUniversidades
#b	pase : GroupLayer
-e	stiloBotao:Styles
-е	stiloBotaoSelecionado : Styles
+T	ipoTela(jogo: VisaoGuerraDasUniversidades)
#i1	niciarBase(): void
#c	lestruirBase(): void
#c	lesenharFundo(cor:int):void
#c	obterEstiloBotao(): Styles
#c	obterEstiloBotaoSelecionado(): Styles
+j	nit(): void
+1.	paint(alpha: float): void
+ι	ıpdate(delta : float) : void
+ 9	shutdown(): void

```
Iniciar
(com::mac242::guerra dasuniversidad es::core::visao)
-fface: Interface
-nome: String = "Lorem"
+to String(): String
+Iniciar[jog: VisaoGuerraDasUniversidades)
+init(): void
+desenhalMenu(): void
+paint(alpha::foat): void
+paint(alpha::foat): void
+shutdown(): void
```

Figura 3.7: Diagrama de classes da visão do jogo

(com::mac242::guerradasuniversidades::core::controle)	(com::mac242::guerradasuniversidades::core::controle)		
-tela : TelaPrincipal +TratadorBotaoBlocoEnsino(tela : TelaPrincipal)	-visao : VisaoGuerraDasUniversidades -tela : TipoTela		
+onEmit(): void	+TratadorTrocarTela(visao : VisaoGuerraDasUniversidades, tela : TipoTela) +onEmit() : void		
Tratador Ataca r Oponente	TratadorBotaoFoco		
(com::mac242::guerra dasuniversidad es::core::controle)	(com::mac242::querra/dasuniversidades::core::controle)		
-jogador : FachadaJogador -oponente : NomeUniversidade -visao : VisaoGuerraDasUniversidades	-foco : FocoAdministracao -visao : VisaoGuerraDasUniversidades -nome : String		
+definirJogador(jogador: FachadaJogador): TratadorAtacarOponente +definirOponente(oponente: NomeUniversidade): TratadorAtacarOponente +definirVisao(visao: VisaoGuerraDasUniversidades): TratadorAtacarOponente +onEmit(): void	+TratadorBotaoFoco(foco:FocoAdministracao, visao:VisaoGuerraDastJniversidades, nome:String) +onEmit():void		

Tratador Troca r Tela

Figura 3.8: Diagrama de classes do controle do jogo

3.5 Utilitários

-im gEstrutura: Im a gem Estrutura -tela: Tel aPrin cip al

TratadorBotaoBlocoEnsino

TratadorBotaoEstrutura (com::mac242::guerradasuniversidades::core::controle)

+TratadorBotaoEstrutura(im gEstrutura: ImagemEstrutura, tela: TelaPrincipal)

O pacote com. mac242. guerradasunivers idades. core.util contém duas classes, Observer e Observable que implementam o padrão de projeto Observador-Observado. Elas são cópias fieis de java.util. Observer e java.util. Observable obtidas de código licensiado pela GPL pela antiga Sun Microsystems, atual Oracle. Isso foi feito pois o GWT não fornece tais classes no subconjunto da API do Java que é traduzível para Javascript. Isso impossibilitaria o deploy para a web no Google AppEngine.

4 Documentação do Código

Como dito anteriormente, o código fonte foi documentado utilizando a ferramenta Javadoc, responsável por gerar automaticamente uma documentação das classes, interfaces, enums, métodos etc. Tal documentação está localizada em Documentacao no formato HTML. Para lê-la abra o arquivo index.html em algum navegador web.

Referências Bibliográficas

- [1] CAMPOS, P. P. V.; HUGUENIN, D. M.; JUNIOR, A. R. C. *Guerra das Universidades: Instruções de Compilação e Execução*. São Paulo SP, dezembro 2011.
- [2] CAMPOS, P. P. V.; HUGUENIN, D. M.; JUNIOR, A. R. C. Guerra das Universidades: Manual do Usuário. São Paulo SP, dezembro 2011.
- [3] EQUIPE DE DESENVOLVIMENTO DO PLAYN. Cross platform game library for $N \ge 4$ platforms. 2011. [Online; acessado em 19 de dezembro de 2011]. Disponível em: http://code.google.com/p/playn/.
- [4] REENSKAUG, T. *Models Views Controllers*. [S.l.], 1979. Disponível em: http://heim.ifi.uio.no/trygver/themes/mvc/mvc-index.html.
- [5] GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. ed. Addison-Wesley Professional, 1994. Hardcover. ISBN 0201633612. Disponível em: http://www.worldcat.org/isbn/0201633612.
- [6] MEYER, B. Applying "design by contract". IEEE Computer, v. 25, n. 10, p. 40–51, 1992.