UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

BIBLIOTECA DE COMPONENTES DE INTERFACE PARA O IPHONE OS

IZABEL CRISTINA DA SILVA CARMO

BLUMENAU
2009

2009/2-11

IZABEL CRISTINA DA SILVA CARMO

BIBLIOTECA DE COMPONENTES DE INTERFACE PARA O IPHONE OS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, M.Sc. - Orientador

BLUMENAU
2009

2009/2-11

BIBLIOTECA DE COMPONENTES DE INTERFACE PARA O IPHONE OS

Por

IZABEL CRISTINA DA SILVA CARMO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
Membro:	Prof. Francisco Adell Péricas, M.Sc. – FURB

Blumenau, 15 de dezembro de 2009

Dedico este trabalho aos meus pais e a todos os meus amigos, que foram compreensivos e perdoaram meus lapsos sociais durante a realização deste.

AGRADECIMENTOS

À meus pais pelo apoio e amor incondicional.

Aos meus amigos, pela paciência, bom humor e apoio.

Ao meu orientador, Dalton Solano dos Reis, por ter me ajudado em todas as necessidades, pelo comprometimento e atenção.

E a todos que direta ou indiretamente me apoiaram e ajudaram para que este trabalho fosse possível.

Livros são os mais silenciosos e constantes amigos; os mais acessíveis e sábios conselheiros; e os mais pacientes professores.

Charles W. Elliot

RESUMO

Este trabalho apresenta a implementação de uma biblioteca de componentes de interface para o iPhone OS, desenvolvida usando o framework UIKit, disponibilizado pela Apple. Esta biblioteca trabalha com a implementação de componentes de interface do tipo joystick, Barra de Vida, Barra de Tempo e Pedal de Aceleração, fornecendo para o usuário-desenvolvedor uma forma rápida e simples de criar e usar os componentes aqui disponibilizados.

Palavras-chave: IHC. iPhone. Dispositivos móveis. Biblioteca. UIKit.

ABSTRACT

This paper describes the implementation of a component library interface for the iPhone OS. This library was developed using the framework UIKit available by Apple and it works with components' interface implementation of a joystick, a Life Bar, a Time Bar and a Gas Pedal, providing to the user-developer a quick and simple way to create and use the components available here.

Key-words: HCI. iPhone. Mobile devices. Library. UIKit.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

CSS - *Cascading Style Sheets*

DOD - *Department Of Defense*

GCC - *Gnu Compiler Collection*

GPS - *Global Positioning System*

HTML - *HyperText Markup Language*

IB - *Interface Builder*

IHC - Interação Humano-Computador

OOPC - *Object-Oriented Programming in C*

OpenGL - *Open Graphics Library*

OSI - *Open Source Iniciative*

PDA - *Personal Digital Assistant*

RAM - *Random Access Memory*

SDK - *Software Development Kit*

SO - Sistema Operacional

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis, como os *smartphones,* surgiram da idéia de misturar as funções de um celular com as de computadores e de aplicativos de um *Personal Digital Assistant* (PDA). As pessoas aprovaram a idéia desde o início, visto que em geral os *smartphones* possuem telas maiores e poder computacional melhor que os simples celulares. Para facilitar e atrair mais pessoas, os Sistemas Operacionais[1] (SOs) dos *smartphones* geralmente são abertos, ou seja, qualquer pessoa pode desenvolver aplicações para funcionar neste tipo de celular (MARIMOTO, 2009).

Em 2008 a Apple Inc lançou um aparelho chamado iPhone 3G, que trouxe para o mercado novas funcionalidades, maior usabilidade, *design* e uma grande quantidade de aplicações disponíveis para este aparelho (APPLE, 2009h).

A Apple também criou a App Store, uma loja virtual que permite baixar aplicativos para o iPhone 3G e iPod *touch*, que podem ser disponibilizados por qualquer desenvolvedor. Atualmente, segundo a Apple (2009b), a App Store americana já conta com mais de 25.000 aplicações disponíveis.

O que se pode observar é que até o presente momento, são poucas as aplicações (principalmente de jogos) que são capazes de criar uma boa interação entre os recursos disponíveis e os usuários. Isso se deve parcialmente ao fato de que muitas das aplicações disponibilizadas na App Store são produzidas por desenvolvedores independentes, que não dispõem do tempo necessário para criar melhores formas de interação. Já as poucas aplicações que conseguem uma boa interação, não disponibilizam para os outros desenvolvedores as bibliotecas usadas.

Diante do exposto, este trabalho propõe o desenvolvimento de uma biblioteca *open source*[2] que permita explorar as formas de interação disponíveis pelo iPhone 3G e possam ser utilizadas de forma mais fácil pelo desenvolvedor.

---

[1] "Sistema operacional é o programa que administra o funcionamento de todos os componentes de hardware (equipamentos) e gerencia o trabalho dos softwares (programas)" (CASTILLO; SURIANI, 2001).

[2] A Open Source Iniciative (OSI), determina que uma aplicação para ser considerada *open source* deve garantir sua distribuição livre, acesso ao código fonte, permitir trabalhos derivados e distribuição da licença (OPEN SOURCE INICIATIVE, 2006).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar uma biblioteca que permita através dos recursos de software explorar os recursos de hardware disponíveis no iPhone 3G, facilitando assim a interação desenvolvedor versus aparelho.

Os objetivos específicos do trabalho são:

a) disponibilizar um componente que simule um *joystick* para iPhone 3G;
b) disponibilizar um componente do tipo barra de tempo;
c) disponibilizar um componente do tipo barra, como aquelas encontradas em jogos para simular a quantidade de vidas ainda disponíveis;
d) disponibilizar um componente do tipo pedal de aceleração.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos, sendo que o segundo capítulo contém a fundamentação teórica necessária para o entendimento deste trabalho.

O terceiro capítulo apresenta sobre o desenvolvimento da biblioteca de componentes de interface, onde são explanados os principais requisitos do problema trabalhado, a especificação contendo diagramas classe e de caso de uso. Ainda no terceiro capítulo é apresentado às ferramentas utilizadas, a implementação, o desenvolvimento e por fim são comentados os resultados e discussão.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são apresentados os conceitos fundamentais da interação/interface humano-computador. Na seção 2.2 comenta-se sobre dispositivos móveis, em particular, o iPhone 3G e iPhone 3GS da Apple. Na seção 2.3 é apresentado o *iPhone Human Interface Guidelines* da Apple para desenvolvimento de interfaces para o dispositivo iPhone 3G. A seção 2.4 apresenta a linguagem de programação Objective-C e na seção 2.5 são apresentadas as ferramentas utilizada neste trabalho. A seção 2.6 apresenta alguns frameworks do SO para iPhone.

Por fim na seção 2.8 são apresentados alguns trabalhos correlatos ao trabalho proposto.

2.1 INTERAÇÃO/INTERFACE HUMANO-COMPUTADOR

Nas primeiras décadas da computação os principais usuários de computadores eram especialistas na área, ou seja, os programas eram feitos por especialistas para especialistas (GENTIL, 2008, p. 50).

Com a diminuição no preço dos computadores tornou-se possível para uma maior variedade de pessoas a compra e utilização deste equipamento. Como conseqüência, atualmente a maior parte destes usuários não são especialistas na área de computação. Para estas pessoas o computador é somente o meio mais fácil de obter a informação desejada (ORTH, 2005, p. 1).

Esta mudança de realidade requer o desenvolvimento de aplicações que sejam naturalmente intuitivas e amigáveis, o que segundo Orth (2005, p. 1) "trouxe para a área de interfaces a necessidade de inúmeros novos conhecimentos de ordem humanística, psicológica, ergonômica, lingüística, semiótica, de design, de comunicação em geral e também de engenharia cognitiva."

Moran (1981, p. 1) conceitua interface como "um sistema de comunicação que possui dois componentes, um componente físico, composto por hardware e software, no qual o usuário percebe e manipula, e outro conceitual, onde ele interpreta, processa e raciocina."

Este processo ação/interpretação é chamado de Interação Humano-Computador (IHC).

Para Hewett et al. (1992, p. 5, tradução nossa), "IHC é uma disciplina preocupada com o *design*, avaliação e implementação de sistemas computacionais interativos para uso humano e com o estudo dos principais fenômenos ao seu redor."

Ou seja, IHC aborda o *design* de sistemas computacionais visando auxiliar as pessoas para que seja possível executar suas tarefas de forma produtiva e com segurança (ROCHA; BARANAUKAS, 2003, p. 15).

> IHC tem, portanto, papel no desenvolvimento de todo tipo de sistema, variando dos sistemas de controle de tráfego aéreo onde segurança é extremamente importante, até sistemas de escritório onde produtividade e satisfação são os parâmetros mais relevantes, até jogos, onde o envolvimento dos usuários é o requisito básico. (ROCHA; BARANAUKAS, 2003, p. 15).

Beltran (1999 apud ORTH, 2005, p. 2) concluiu através de pesquisas que uma interface ruim é a causa pela qual aproximadamente 86% dos usuários de uma aplicação param de usá-la. Orth (2005) afirma que os fatores que causam interfaces ruins são:

a) falta ao projetista o conhecimento dos usuários;
b) as interfaces acabam sendo feitas de projetistas para projetistas, pois os mesmos esquecem-se das necessidades dos usuários;
c) foca-se muita na funcionalidade e acaba-se esquecendo da usabilidade da aplicação;
d) a falta de conhecimento nos fatores humanos.

Cybis, Betiol e Faust (2007 apud SHNEIDERMAN; PLAISANT, 2004, p. 24) sugerem 8 regras para a criação de interfaces humano-computador sendo elas:

a) perseguir a consistência;
b) fornecer atalhos;
c) fornecer *feedback* informativo;
d) marcar o final dos diálogos;
e) fornecer prevenção e manipulação simples de erros;
f) permitir o cancelamento das ações;
g) fornecer controle e iniciativa ao usuário;
h) reduzir a carga de memória de trabalho.

Para que um sistema tenha uma boa aceitação os fatores utilidade, confiabilidade, usabilidade e eficiência precisam ser satisfeitos (ORTH, 2005, p. 3).

2.2 DISPOSITIVOS MÓVEIS

Para Cybis, Betiol e Faust (2007, p. 217), o uso cada vez mais comum de equipamentos portáteis que fazem uso da tecnologia de comunicação sem fio, como os celulares, PDAs, está afetando a forma com que as pessoas interatuam com informações que antes só se tinham acesso por meio de computadores. Esta mudança permite que novos equipamentos, serviços e aplicações possam ser criados para atender estas novas necessidades deste tipo de usuário.

"A definição de dispositivo móvel é qualquer equipamento ou periférico que possa ser transportado com conteúdo e esteja acessível em qualquer lugar" (FIEMG, 2007, p. 1).

2.2.1 iPhone 3G

Segundo a Apple (2009l), o iPhone 3G e iPod *touch* são dispositivos que combinam a interface multi-toque[3] com recursos sofisticados, como *e-mail*, e mensagem instantâneas e no caso do iPhone 3G também um celular.

Sendo o iPhone 3G um dispositivo móvel, o mesmo tem as suas limitações:

a) tamanho da tela: o iPhone 3G está disponível com uma tela de 3,5 polegadas com 480x320 pixel e com uma resolução de 163 ppi[4] (APPLE, 2009s);
b) processamento e memória: o poder de processamento do iPhone 3G não é muito alto, embora seja maior que o dos outros *smartphones*. Ainda de acordo com a Apple (2009l, p. 16), o iPhone 3G não permite utilizar partes da memória de armazenamento para aumentar o tamanho da memória RAM[5], que é de aproximadamente 128MB;
c) teclado virtual: o iPhone 3G fornece ao usuários um teclado virtual, que aparece quando é necessário e desaparece quando o mesmo não for mais preciso (Apple, 2009m). O problema é que o teclado virtual prejudica a usabilidade, pois o usuário demora mais para digitar textos longos do que em um teclado real e ainda existe

[3] Multi-toque é também conhecido pelo seu original do inglês *multi-touch*.

[4] *Pixels per inch.*

[5] *Random Access Memory*: em português a tradução fica como Memória de Acesso Aleatório.

uma probabilidade maior de acabar acertando uma tecla errada, já que se o iPhone 3G estiver sendo segurado com as duas mãos, os únicos dedos que sobram para digitar são os polegares. Os polegares são grandes e acabam encobrindo a tecla desejada, o que pode ocasionar que as teclas que estão ao redor possam ser acionadas por engano;

d) armazenamento: o iPhone 3G pode ser encontrado com a capacidade de armazenamento de 8GB ou de 16GB. Este dispositivo móvel não aceita cartões para a expansão da memória de armazenamento (APPLE, 2009l). Embora a capacidade de armazenamento do iPhone seja muito maior que o de outros dispositivos móveis, essa mesma capacidade é pequena em comparação com os computadores.

Assim como apresenta limitações, este dispositivo também apresenta diversas vantagens. Entre elas pode-se citar:

a) mobilidade: que segundo Ferreira (1986, p. 347), é: "1. Qualidade ou propriedade do que é móvel ou obedece às leis do movimento. 2. Facilidade de mover-se ou de ser movido";

b) tela multi-toque: é capaz de reconhecer mais de um toque ao mesmo tempo em diferentes pontos da tela. Permite também rolar listas com o deslizar do dedo, aumentar ou reduzir imagens usando o polegar e o indicador, simulando o movimento de uma pinça (APPLE, 2009o);

c) acelerômetro:

> O iPhone responde ao movimento graças à um acelerômetro interno. Quando você gira o iPhone da posição retrato (vertical) para a posição paisagem (horizontal), o acelerômetro detecta automaticamente o movimento e adapta o visor, para que você possa visualizar imediatamente uma página web em toda a sua extensão, uma foto na sua devida proporção ou controlar um jogo com seus movimentos [...]. (APPLE, 2009a, p. 1);

d) *Global Positioning System*[6] (GPS): está sob a responsabilidade do U.S. Department of Defense (DoD) que é responsável pelo desenvolvimento do mesmo (U.S. DEPARTMENT OF DEFENSE, 1996), permitindo que aplicações civis façam uso do seu GPS. Conforme Apple (2009f) o iPhone 3G "utiliza informações de satélites em órbita para encontrar os locais. Um receptor estima a distância dos satélites GPS com base no tempo que demora para o sinal chegar até ele e usa estas informações para compor o itinerário."

[6] Em português Sistema de Posicionamento Global.

2.2.2 iPhone 3GS

Segundo a Apple (2009g), no ano de 2009, foi disponibilizado no mercado o novo iPhone, essa nova versão é chamada de iPhone 3GS. O iPhone 3GS possui todas as funcionalidades do seu antecessor, mas teve algumas funcionalidades antigas melhoradas e também novas funcionalidades foram disponibilizadas como:

a) possibilidade de gravar vídeos e editá-los;
b) controle de voz;
c) aumento na capacidade de vida da bateria.

Já as seguintes funcionalidades foram melhoradas:

a) a câmera passou a ser de 3 *megapixels*;
b) capacidade de armazenamento de até 32GB.

2.3 IPHONE HUMAN INTERFACE GUIDELINES

A Apple disponibiliza o *iPhone Human Interface Guidelines* para que os desenvolvedores de aplicações para iPhone 3G e iPod *touch* possam criar aplicativos para dispositivos móveis que usem da melhor forma todos os recursos oferecidos por estes dispositivos (Apple, 2009l).

A primeira recomendação é ter em mente que uma vez que a tela é menor que uma tela de computador, será necessário que a interface com o usuário tenha somente o essencial, mesmo porque interfaces carregadas transformam uma aplicação em algo sem atrativos e difícil de usar (APPLE, 2009l).

Também deve-se lembrar que a memória é limitada e portanto deve-se evitar alocar mais do que está disponível no aparelho e deve ser limpada de forma regular. Ainda segundo a Apple (2009l), deve-se eliminar de sua aplicação vazamentos de memória, deixar os arquivos de recursos o menor possível e retardar o máximo possível o carregamento de recursos.

Outra recomendação é que no iPhone 3G, os usuários podem ter diferentes telas, mas somente podem acessar e ver uma tela por vez, ou seja, nunca simultaneamente (APPLE, 2009l). Assim sendo, somente uma aplicação pode rodar por vez. Já aplicações *third-part*y

nunca rodam em segundo plano (APPLE, 2009l).

Segundo a Apple (2009l) como o iPhone 3G possui recursos escassos em comparação com um computador, não deve-se gastar muito espaço para guardar ou mostrar informações de ajuda, até mesmo porque usuários de aparelhos móveis não dispõem de tempo para ler um monte de informações de ajuda, como acontece nos computadores. Sendo assim é possível fazer algumas coisas para evitar isso como:

a) usar controles padrões, pois os usuários já estão familiarizados com os mesmos, o que diminui a necessidade de ajuda;
b) certificar-se que as informações de trajetória informadas sejam lógicas e fáceis para o usuário predizer, também prover botões de retorno para permitir ao usuário refazer seus passos.

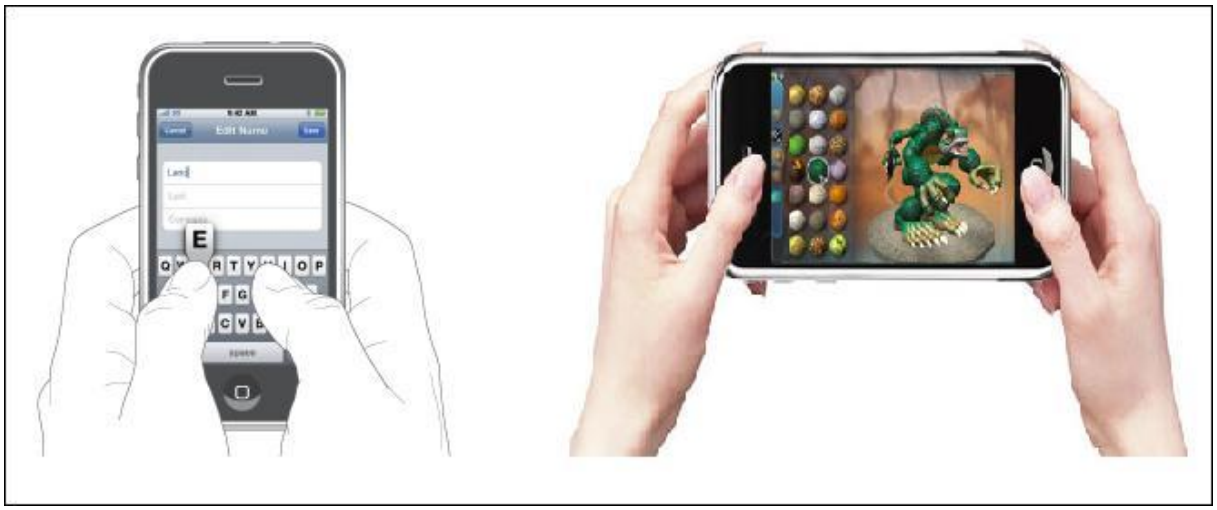
Outra recomendação importante é apresentar escolhas ou opções em forma de lista para o usuário, pois isto permite que o mesmo possa de maneira fácil examinar e escolher algo da lista que lhe está sendo mostrada, economizando assim tempo e dando usabilidade para a aplicação (APPLE, 2009l).

É importante segundo a Apple (2009l), que a aplicação ofereça para cada ação do usuário uma reposta visual, pois isto ajuda a melhorar a experiência do usuário com a aplicação, já as informações que se deseja mostrar ao usuário ficam melhor onde elas são mais visíveis e acessíveis, ou seja, na parte superior da tela. De acordo com a Apple (2009l, p. 37, tradução nossa), “as pessoas podem usar os dedos ou polegares para tocar a tela de um iPhone. Quando as pessoas usam um dedo, o mesmo tende a segurar o dispositivo na mão não dominante (ou deixam em cima de uma superfície) e tocam a tela com um dedo da mão dominante.”, conforme pode ser observado na Figura 1.

Fonte: Meu PDA (2009).

Figura 1 – Uma das possíveis formas de se usar o iPhone

Ainda de acordo com a Apple (2009l, p. 37, tradução nossa), "quando são usados os polegares, as pessoas ou seguram o dispositivo numa mão e tocam a tela com o polegar da mesma mão, ou seguram o dispositivo entre suas mãos e usam os dois polegares para tocarem a tela.", segundo pode ser observado na Figura 2. "Qualquer dos métodos que seja utilizado, a parte superior da tela é a mais visível para as pessoas." (APPLE, 2009l, p. 37, tradução nossa).



Fonte: Apple (2009j).

Figura 2 - iPhone sendo utilizado com ambos os polegares

Para evitar que o usuário perca tempo, despenda atenção e se irrite sendo cuidadoso na hora de tocar a tela para evitar tocar no elemento errado, deve-se colocar espaço suficiente entre os elementos. A Apple sugere um espaço de 44 x 44 pixels, para evitar que isso ocorra (APPLE, 2009l).

A Apple (2009l) identifica três estilos de aplicações, baseado nas características visuais, comportamentais e no modelo de dados, sendo estas aplicações de produtividade, aplicações de utilidade e aplicações imersivas.

As aplicações de produtividade geralmente têm seus dados organizados de forma hierárquica, usam múltiplas visões e geralmente mostram um nível de hierarquia por visão, permitindo que as pessoas possam achar as informações de forma progressiva até que cheguem ao nível de detalhes que estão querendo. Pode-se ser usado como exemplo de uma aplicação de produtividade o álbum de fotos e o e-mail (APPLE, 2009l).

"Uma aplicação de produtividade habilita tarefas que são baseadas na organização e manipulação de informações detalhadas. As pessoas usam aplicações de produtividade para realizar tarefas importantes." (APPLE, 2009l, p. 19, tradução nossa). Na Figura 3 é possível observar um exemplo de uma aplicação de produtividade;

Fonte: Techbook (2009), GSMFans (2007).

Figura 3 – Múltiplas visões de uma mesma aplicação de e-mail

Aplicações que realizam tarefas simples, rápidas e que exigem pouca interferência do usuário são chamadas de aplicações de utilidade pela Apple (2009l). "Uma aplicação de utilidade tende a organizar informações em uma lista de itens compacta, já que geralmente os usuários desse tipo de aplicação não precisam de muitos detalhes." (APPLE, 2009l, p. 21, tradução nossa). Segundo a Apple uma aplicação para saber informações básicas sobre o tempo, é considerada uma aplicação de utilidade, tal exemplo pode-se ser observado na Figura 4.

Fonte: GSMFans (2007).

Figura 4 – Aplicação que apresenta informações sobre o tempo

O terceiro e último estilo de aplicação são as aplicações imersivas.

> Uma aplicação imersiva oferece uma tela cheia, ambiente visualmente rico que foca no conteúdo e na experiência do usuário com aquele conteúdo. As pessoas geralmente usam esse tipo de aplicação para se divertirem, jogar um jogo, visualizar conteúdos de mídia ou desempenhar uma tarefa simples. (APPLE, 2009l, p. 23, tradução nossa).

Ainda conforme a Apple (APPLE, 2009l, p. 23, tradução nossa) as "tarefas que apresentam um ambiente único, não exibem grandes quantidades de informações com base em textos e recompensa o usuário por sua atenção são bons candidatos para a abordagem imersiva.".

2.4 OBJECTIVE-C

A linguagem Objective-C foi criada no inicio da década de 1980 pela empresa Stepstone, cujos proprietários eram Brad Cox e Tom Love, sendo estes em sua maioria os responsáveis pela criação da linguagem Objective-C. A idéia do desenvolvimento da linguagem Objective-C, inicialmente chamada de Object-Oriented Programming in C (OOPC), surgiu da necessidade de dividir os programas em partes menores e da reutilização de código, já que embora fosse possível fazer isto utilizando programação estrutura, era uma técnica que acabava se tornando complexa e que permitia pouco reaproveitamento de código.

Uma solução encontrada para este problema foi utilizar uma linguagem orientada a objetos como a linguagem de programação Smalltalk[7], mas esta usava uma máquina virtual, que exigia muita memória e executava muito lentamente. Assim sendo, Brad Cox e Tom Love começaram a trabalhar no que hoje é a linguagem Objective-C.

Em 1988, a empresa NeXT de Steve Jobs em uma transação legal, licenciou o Objective-C e mais tarde fazendo uso desta linguagem a empresa NeXT criou o sistema OpenStep, que viria a ser a base para os sistemas operacionais dos computadores Macs e dos iPhones, após a compra da NeXT pela Apple em 1996 (APPLE, 2009t).

Objective-C é uma linguagem orientada a objetos, sendo uma extensão do padrão ANSI C e fortemente inspirada na linguagem Smalltalk. Já que a linguagem Objective-C incorpora a linguagem C é possível obter todos os benefícios do C usando Objective-C, ou seja, o desenvolvedor decide quando deseja usar técnicas de programação procedural ou se o mesmo deseja usar orientação a objetos.

Objective-C é uma linguagem simples, pois sua sintaxe é pequena, não ambígua e fácil de aprender. Em comparação com outras linguagens orientadas a objetos baseadas em C, Objective-C é bastante dinâmica. O compilador preserva uma grande quantidade de informações sobre os objetos para uso em *runtime*. Esta linguagen permite postergar para quando o programa estiver rodando, decisões que possam ser feitas em momento de compilação, isto faz com que Objective-C seja flexível e poderosa (APPLE, 2009t).

"Ao contrário de algumas outras linguagens, Objective-C não possui uma sintaxe para informar que uma classe é abstrata, nem previne que alguém crie uma instancia de uma classe que é conceitualmente abstrata." (APPLE, 2009t, p. 27, tradução nossa).

Outra característica desta linguagem segundo a Apple (2009t), é que não existe uma diretiva para indicar que um método é privado. Para se declarar que um método é privado, é necessário fazer uma extensão da classe, ou seja, na classe de implementação é preciso declarar antes da diretiva que indica o início da implementação o conteúdo do Quadro 1.

```
@interface NomeClasse()
    - os métodos privados devem vir aqui;
@end
```

Quadro 1 – Declaração de métodos privados em Objective-C

[7] "Uma das primeiras linguagens de programação orientada a objetos." (APPLE, 2009t, p. 9, tradução nossa).

2.5 FERRAMENTAS

Abaixo encontram-se a descrição de algumas possíveis ferramentas para se trabalhar com Objective-C, que auxiliam no desenvolvimento de projetos para o SO do iPhone.

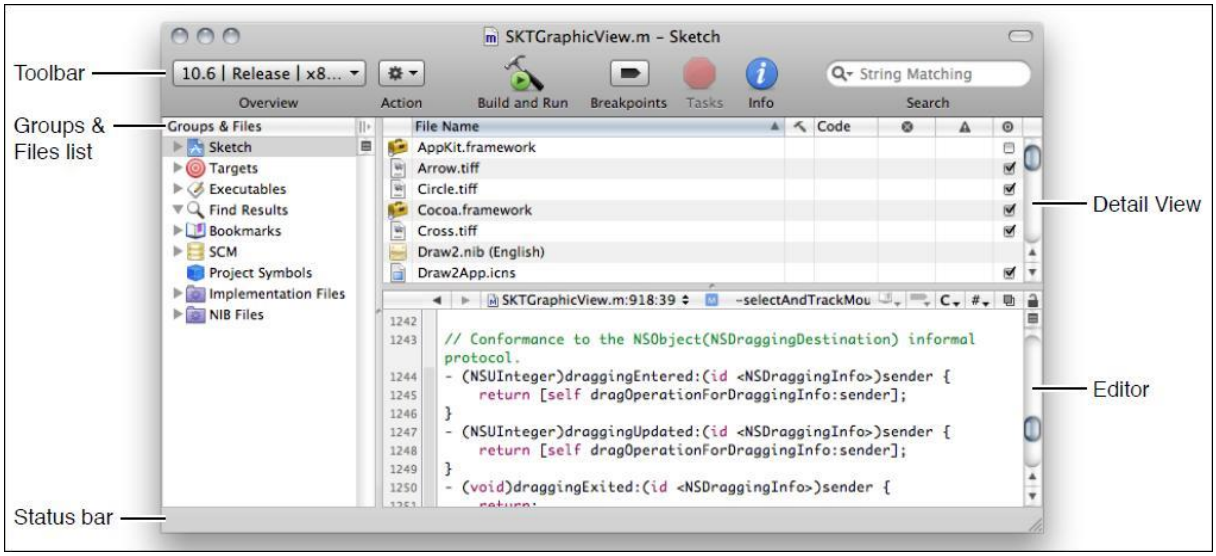
2.5.1 Xcode

Xcode é um conjunto de ferramentas da empresa Apple para desenvolvimento de software no SO Mac X. Segundo a Apple (2009c, p. 7, tradução nossa), o "Xcode inclui compiladores e aplicativos, juntamente com um extenso conjunto de bibliotecas de programação e interfaces." A principal aplicação deste conjunto de ferramentas é um ambiente gráfico também chamado de Xcode, que oferece uma poderosa interface para que o usuário possa criar e gerenciar seus projetos.

Ainda de acordo com a Apple (2009c, p. 9, tradução nossa), o "Xcode é um ambiente integrado de desenvolvimento altamente personalizável".

O Xcode possui integrado um editor de código, um depurador e o poder do compilador GNU Compiler Collection[8] (GCC). Outras ferramentas que são distribuídas como parte do Xcode são: Interface Builder, Dashcode e Instruments, além da completa documentação de referência dos mesmos (APPLE, 2009u). Na Figura 5, pode ser observada a principal tela do aplicativo Xcode.

[8] Segundo o GNU Project (2007), GCC é um conjunto de compiladores de linguagens de programação.



Fonte: Apple (2009c).

Figura 5 – Principal tela do ambiente de desenvolvimento Xcode

Conforme Apple (2009n), o Xcode está na versão 3.2 e conta com as seguintes novidades:

a) foco no código: mostra a estrutura do bloco de código sob demanda;
b) maior desempenho para o editor de código do Xcode, como por exemplo, pesquisa no *workspace* mais rápida, indentação melhorada;
c) erros de compilação, avisos e *breakpoints* são agora visualizados no código fonte e não mais apenas como um ícone na lateral;
d) suporte a linguagem Objective-C 2.0 com coletor de lixo;
e) snapshots: uma versão básica de controle de código;
f) refatoração de código da linguagem Objective-C e arquivos .nib;
g) assistente de pesquisa sensível ao contexto documentação;
h) instrumentos de análise de desempenho;
i) atualizado o assistente para auxiliar na criação de novos projetos;
j) suporte para o SO do iPhone Software Development Kit (SDK).

2.5.2 Interface Builder

Interface Builder (IB) é um editor gráfico desenvolvido pela Apple, cujo objetivo é facilitar a criação, edição e o *design* de componentes de interfaces para aplicações do SO Mac e do SO iPhone (APPLE, 2008a).

Segundo Apple (2008b, p. 11), o IB está na versão 3.1 e fornece janelas, botões, menus e outros elementos a partir de uma biblioteca de objetos, sendo possível reposicionar estes itens pela tela, estabelecer conexões entre eles, e mudar seus atributos entre outros recursos, permitindo aos desenvolvedores gerenciarem cada aspecto da criação de uma interface, conforme pode ser observado na Figura 6.

Isto pode ser feito "[...] arrastando elementos a partir de uma das paletas de controles pré-definidos e soltando os elementos na janela ou visão na qual eles estão configurados" (APPLE, 2008a, tradução nossa).

Fonte: Apple (2008b).

Figura 6 – Visão da aplicação Interface Builder

Informações como as configurações e *layout* dos objetos das interfaces criadas são salvas pelo IB em um arquivo com extensão .nib, para que em tempo de execução possam ser recreados (APPLE, 2008b).

2.5.3 Dashcode

Dashcode é um editor gráfico que está na versão 2.0 e é desenvolvido pela Apple Inc. O Dashcode permite ao desenvolvedor criar de forma rápida e fácil *widgets*[9] e aplicações web para o SO Mac e o SO do iPhone (Apple, 2007).

Segundo a Apple (2009e), o ambiente integrado Dashcode conta com ferramentas para o *design*, editor de código além de permitir testar a aplicação web ou *widget* sem precisar usar

[9] Aplicativos pequenos e leves que executam uma única tarefa e fornecem ao utilizador funcionalidades específicas como previsão do tempo, cotação de moedas, relógio, entre outros (APPLE, 2009e).

nenhuma outra aplicação, como pode ser observado na Figura 7.

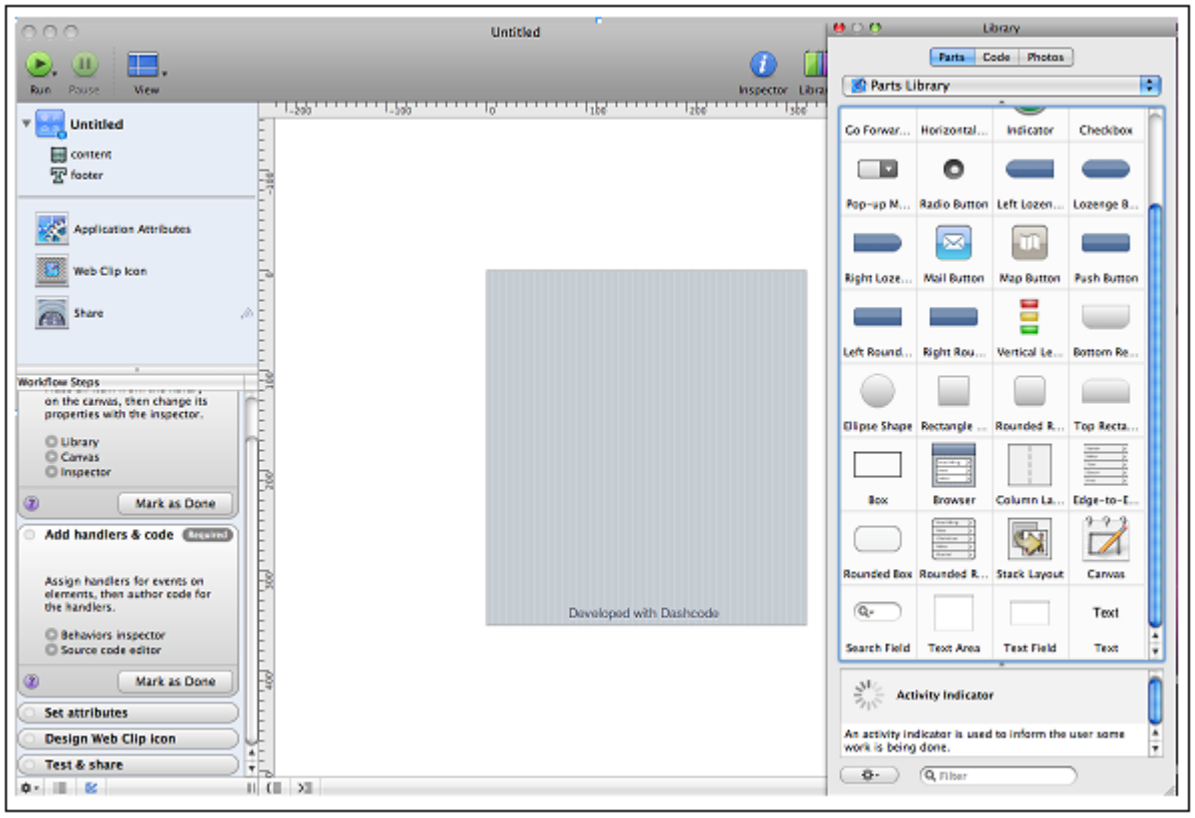


Figura 7 – Visão da aplicação Dashcode

O Dashcode de acordo com a Apple (2007), também fornece ao desenvolvedor uma biblioteca de elementos gráficos que conta com vários tipos de botões, menus, campos de textos dentre outros, que podem ser arrastados para a janela ou visão na qual eles estão configurados. Cada elemento gráfico possui diversas configurações que podem ser editadas.

Arquivos de código do tipo Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript são suportados pelo editor de código que também fornece a capacidade de auto-completar (APPLE, 2007).

2.6 FRAMEWORKS

Segundo Fayad (1999, p. 3, tradução nossa), “um *framework* é um projeto reutilizável de todo ou de parte de um sistema que é representado por um conjunto de classes abstratas e a forma como suas instâncias interagem.” Ainda de acordo com Fayad (1999, p. 3, tradução nossa), outra definição para framework pode ser, “um *framework* é o esqueleto de um aplicativo que pode ser customizado por um desenvolvedor de aplicativos.” Fayad ainda

afirma que embora ambas as definições sejam diferentes, ambas estão corretas já que a primeira descreve a estrutura de um *framework*, enquanto a segunda descreve a sua finalidade.

Já de acordo com Apple (2009v), um *framework* possui a mesma finalidade que as bibliotecas, pois fornecem rotinas que podem ser chamadas por algum aplicativo para que uma tarefa específica seja executada. Os *frameworks* (APPLE, 2006):

a) agrupam recursos relacionados, mas independentes;
b) podem incluir uma maior variedade de tipos e recursos;
c) somente uma cópia de um *framework* reside fisicamente na memória, independente de quantos processos estão usando estes recursos.

2.6.1 UIKit

O *framework* UIKit é escrito na linguagem Objective-C e fornece os objetos fundamentais para construir e gerenciar interfaces para as aplicações do usuário para a plataforma iPhone e iPod touch. Com este framework é possível suportar manipulação de eventos, desenhos, janelas e controles todos projetados para a interface multi-toque (APPLE, 2009v).

De acordo com a Apple (2009d), é possível usar os objetos do *framework* UIKit de três formas:

a) usar a aplicação Interface Builder para arrastar-soltar janelas, botões e outros objetos;
b) criar, posicionar e configurar objetos programaticamente;
c) implementar interfaces customizadas que herdam da classe UIView ou de classes que já estendam da mesma.

As classes, métodos, tipos de dados e constantes do UIKit tem seu nome precedidos pelo prefixo “UI”. As classes do *framework* UIKit herdam da classe NSObject que é a classe principal e também a classe raiz do *framework*, que de acordo com a Apple (2009p, p. 7, tradução nossa), “através da classe NSObject, os objetos herdam uma interface básica para o sistema de execução e a capacidade de comportar-se como objetos do tipo Objective-C”.

Na base está uma classe de reposta chamada de UIResponder, que é responsável por definir a interface e o comportamento padrão, se houver, para o tratamento de eventos, como por exemplo, quando o usuário toca na tela.

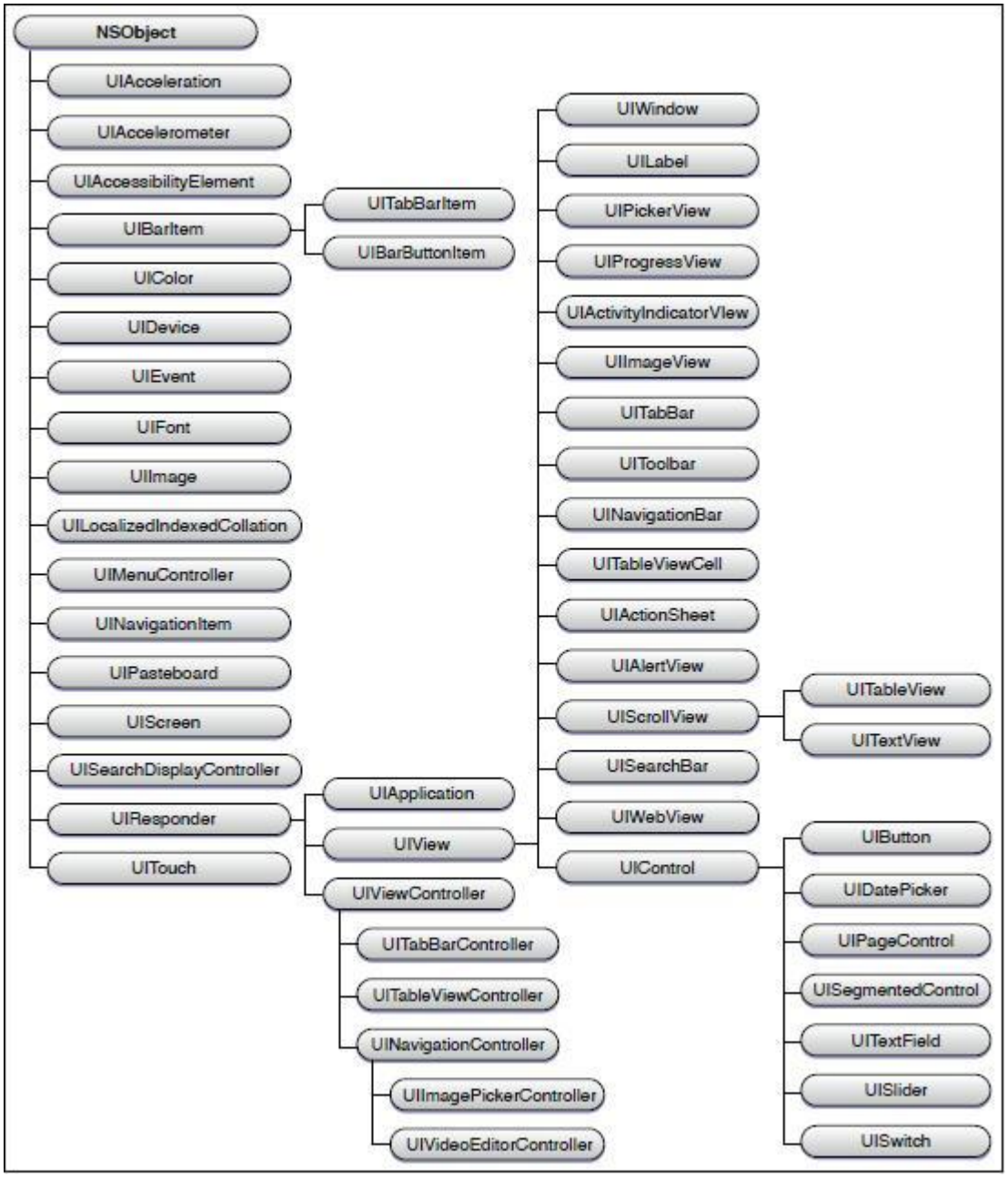
A classe UIImage é usada para representar e encapsular imagens, sendo que os objetos

criados a partir de uma classe UIImage são imutáveis, ou seja, não é possível alterar as suas propriedades após a criação. Sendo assim, também não é possível acessar diretamente os dados subjacentes da imagem (APPLE, 2009d).

Segundo a Apple (2009k, p. 49, tradução nossa), a classe UIView "define as propriedades básicas de uma visão, mas não a sua representação visual. Em vez disso, usa subclasses para definir a aparência e comportamento para elementos de sistema padrão, tais como campos de texto, botões e barra de ferramentas."

A classe UIImageView fornece uma visão que permite exibir uma única imagem ou animar uma série de imagens. Está classe fornece controles para definir a duração e freqüência da animação, assim como permite iniciar e parar a animação conforme necessidade. Por padrão novas imagens vem configuradas para ignorar eventos do usuário, assim sendo, deve-se mudar o valor da propriedade *userInteractionEnable* para *Yes*, para que a imagem pare de ignorar eventos do usuário (APPLE, 2009v).

Na Figura 8 é apresentado a hierarquia das classes do *framework* UIKit.



Fonte: Apple (2009d).

Figura 8 – Hierarquia de classes do *framework* UIKit

2.6.2 Open Graphics Library (OpenGL)

Segundo Manssour (2003), OpenGL é uma biblioteca de rotinas gráficas aberta para criação de aplicações gráficas bidimensionais (2D) e tridimensionais (3D), que possui portabilidade e rapidez.

> Seu funcionamento é semelhante ao de uma biblioteca C, uma vez que fornece uma série de funcionalidades. Normalmente se diz que um programa é baseado em OpenGL ou é uma aplicação OpenGL, o que significa que ele é escrito em alguma

> linguagem de programação que faz chamadas a uma ou mais bibliotecas OpenGL. (MANSSOUR, 2003).

OpenGL foi desenvolvida pela Silicon Graphics Inc e liberada pela primeira vez em 1992, atualmente existem implementações de OpenGL para Mac OS, Windows, Linux, Irix, Solaris, iPhone OS, dentro outros (APPLE, 2009r).

Segundo a Apple (2009r), OpenGL é excelente para se desenvolver gráficos no Mac OS, pois oferece:

a) implementação segura;
b) aceitação industrial;
c) performance;
d) controle de evolução, já que as extensões do OpenGL permitem que os desenvolvedores utilizem as melhorias de hardware quando estas ficam disponíveis;
e) independência de plataforma.

2.6.2.1 OpenGL for Embedded Systems (OpenGL ES)

De acordo com a Apple (2009q), o OpenGL ES é uma versão derivada do OpenGL, mas que foi desenvolvida para uso em dispositivos móveis. O OpenGL ES, permiti aproveitar melhor as vantagens gráficas dos hardwares modernos, também possui uma interface mais simples que facilita o aprendizado e a implementação em dispositivos móveis.

2.7 TRABALHOS CORRELATOS

Existem algumas aplicações para iPhone 3G e iPod *touch* que exploram os novos recursos de hardware oferecidos por estes dispositivos através de recursos de software. Dentre eles, foram escolhidos os jogos Prey, Pac-Man Remix, Brothers in Arms: Hour of Heroes, Minigore, Assassin's Creed e Asphalt 4: Elite Racing.

2.7.1 Prey

Prey é um jogo adaptado para o dispositivo iPhone 3G da Apple, que apresenta uma nova forma de controlar um personagem simulando um *joystick* usando o sistema multi-toque.

Segundo Kichalowsky (2009), a MachineWorks Northwest desenvolvedora do jogo "criou um esquema composto de quatro barras para controlar a direção e acionar as funções do jogo com os polegares." Isso pode ser observado na Figura 9.

Fonte: Buchanan (2009b).

Figura 9 - Jogo Prey com o controle simulado do tipo *joystick*

Próximo as barras existem alguns botões para executar algumas ações diversas, e assim deixar o *joystick* virtual ainda mais utilizável (BUCHANAN, 2009b). O jogo Prey também oferece a possibilidade de se controlar a direção, usando controles circulares, como pode ser observado na Figura 10.

Fonte: Buchanan (2009b).

Figura 10 - Jogo Prey mostrando uma forma diferente de controle

2.7.2 Pac-Man Remix

Pac-Man Remix é a versão remixada feita para o iPhone 3G do clássico *arcade* Pac-Man, onde ao contrário do original nesta versão há seis mundos com cinco mapas cada, além de ter um *layout* muito mais criativo que permite ao jogador passar por portas, pontes movediças, elevadores para múltiplos níveis, entre outros (PALLEY, 2009).

Segundo Ellis (2009c), é possível mover o Pac-Man usando gestos na tela ou usando os botão de controle, mas quando se opta por usar os botões estes não respondem como deveriam, sendo que os botões de navegação ficam mal localizados na parte inferior da tela, conforme apresentado na Figura 11.

Fonte: Ellis (2009c).

Figura 11 - Jogo Pac-Man Remix com seus botões controle

2.7.3 Brothers in Arms: Hour of Heroes

Segundo a Gameloft (2008a), o enredo do jogo se passa durante a Segunda Guerra Mundial onde "você é Jason Becker, um soldado da 101ª divisão da Força Aérea [...]. Ação digna dos grandes filmes que fará você reviver as batalhas mais intensas da 2ª Guerra Mundial na Normandia, Ardenas e Tunísia."

No jogo o recurso acelerômetro disponível no iPhone é usado para lançar granadas e o *touch screen* para atirar nos inimigos.

Na opção padrão o controle do jogo é feito com toques na tela. "Para mover-se, é só

clicar no 'controle' direcional do canto esquerdo do *display* e, tocando em qualquer outro lugar da tela, é possível mudar a visão. Para atirar, é só clicar no 'botão' no canto inferior direito." afirma Alvarenga (2008), conforme pode ser visto na Figura 12.

Fonte: Gameloft (2008a).

Figura 12 - Jogo Brothers in Arms, com seus botões de controle

2.7.4 Minigore

Minigore é um jogo de tiro para iPhone 3G com jogabilidade simples e visual ao estilo cartoon, onde o objetivo é segundo Ellis (2009b), "sobreviver a ataques constantes de monstros enquanto se movimenta e atira com a sua arma para todos os lados". Dois *joysticks* um em cada lateral da tela do iPhone controlam o jogo, sendo o da direita usado para atirar e o da esquerda para se mover, como pode ser observado na Figura 13 (HODAPP, 2009).

Fonte: Ellis (2009b).

Figura 13 - *Joysticks* de controle do jogo Minigore

2.7.5 Assassin's Creed

Assassin's Creed é um jogo que está presente em diversas plataformas como Xbox 360, PS3, PC, Nintendo DS e também para o iPhone 3G, onde na versão Altair's Chronicles para o iPhone o jogo se passa durante a Terceira Cruzada em Jerusalém no ano de 1191 (ELLIS, 2009a).

O personagem tem a sua disposição armas como uma espada, um arco, uma adaga e duas bombas, o controle dessas armas são feitos através de controles virtuais na tela no formato de botões, onde para cada arma haverá um botão, já o controle do movimento é feito através de um *joystick* na tela na lateral esquerda, conforme Figura 14 (BUCHANAN, 2009a).

Fonte: Ellis (2009a).

Figura 14 - Botões de controle de movimento e armas do jogo Assassin's Creed

2.7.6 Asphalt 4: Elite Racing

Asphalt 4: Elite Racing é um jogo de corrida para iPhone 3G com ambientes 3D, que permite ao usuário escolher entre motos e carros de marcas famosas (GAMELOFT, 2008b).

Segundo Ellis (2008), é possível controlar as motos ou carros no jogo usando o acelerômetro do iPhone como volante ou pilotar com um volante virtual na tela, ou ainda fazer gestos laterais para direcionar o automóvel, conforme Figura 15. Também é possível nesta versão jogar multi-player atráves de uma conexão Wi-Fi.

Fonte: Gameloft (2008b).

Figura 15 - Jogo Asphalt 4: Elite Racing controlado por um volante virtual

"No modo com acelerômetro, ao dar um impulso você aciona o Nitro Boost do seu carro e se fizer isto três vezes seguida, vai se tornar praticamente imbatível na estrada. Você também pode acionar o turbo em um botão na tela do iPhone [...]" (ELLIS, 2009).

3 DESENVOLVIMENTO DA BIBLIOTECA

Neste capítulo são abordadas as etapas do desenvolvimento do projeto. São ilustrados os principais requisitos, a especificação, a implementação e por fim são listados os resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados abaixo se encontram classificados em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF).

O sistema deverá:

a) ser desenvolvido em Objective-C (RNF);
b) utilizar o ambiente de programação Xcode (RNF);
c) executar na plataforma iPhone (RF);
d) permitir ao usuário decidir onde posicionara o componente *joystick* na tela (RF);
e) permitir que imagens sejam adicionadas para o componente *joystick* (RF);
f) permitir que o componente barra de vida tenha seu valor de taxa de atualização modificada a qualquer momento (RF);
g) permitir modificar o valor para a quantidade restante de vida do componente barra de vida a qualquer momento (RF);
h) permitir que imagens sejam adicionadas e que a posição na tela onde o componente barra de vida será incluso seja decisão do usuário (RF);
i) permitir que o componente barra de tempo tenha o tempo restante modificado a qualquer momento (RF);
j) permitir ao componente pedal de aceleração ter sua taxa de aceleração modificada quando o usuário decidir (RF);
k) permitir ao usuário escolher se o mesmo deseja usar a desaceleração com o componente pedal de aceleração (RF);
l) possuir documentação (RNF).

3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida utilizando alguns dos diagramas da Unified Modeling Language (UML) em conjunto com a ferramenta Enterprise Architect 6.5.802 para a elaboração dos casos de uso e dos diagramas de classes.

3.2.1 Diagrama de casos de uso

Como este trabalho é uma biblioteca de componentes de interface, o desenvolvedor dos sistemas deverá ser considerado como o usuário. Nas próximas seções são descritos os casos de uso para os componentes do tipo *joystick*, barra de vida, barra de tempo e pedal de aceleração.

3.2.1.1 Diagrama de caso de uso do componente *joystick*

A Figura 16 apresenta o diagrama de casos de uso do componente *joystick* com as principais interações do usuário com o sistema.

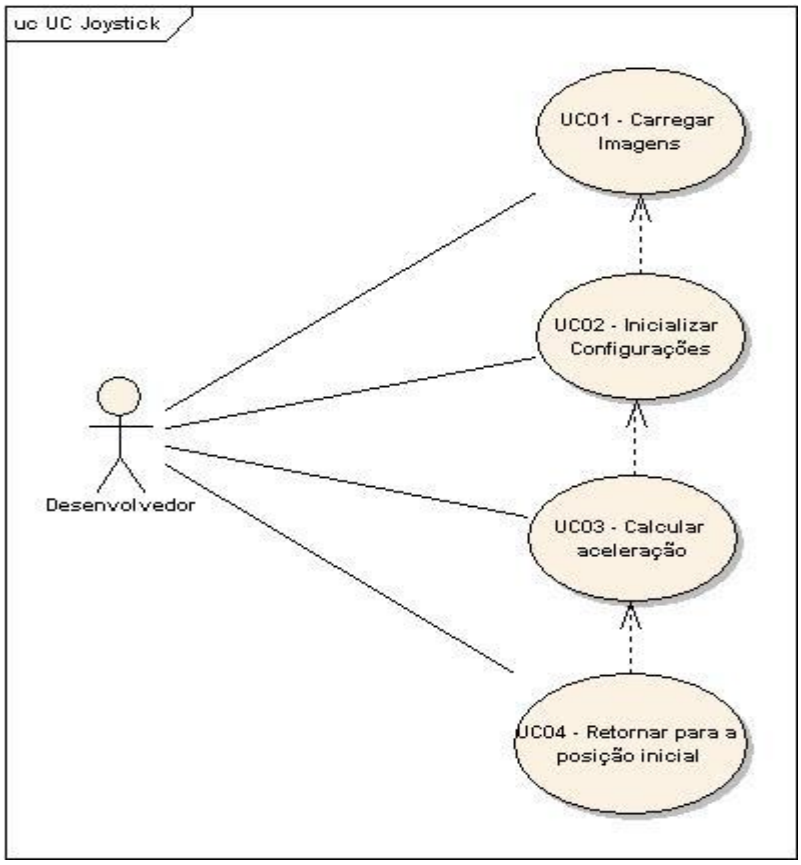


Figura 16 - Diagrama de casos de uso do componente *joystick*

3.2.1.1.1 UC01 - Carregar imagens

O caso de uso `UC01 - Carregar Imagens` (Quadro 2) descreve como o usuário-desenvolvedor adiciona as imagens do componente *joystick* a sua aplicação. Este caso de uso é composto por um cenário principal e um de exceção.

UC01 – Carregar imagens: possibilita ao usuário-desenvolvedor adicionar a imagem de fundo e a imagem superior do componente *joystick*.	
Pré-condição	O desenvolvedor possuir uma imagem em formato PNG ou JPG.
Cenário principal	1. O desenvolvedor informa a imagem de fundo do *joystick* através do método `loadImagejoystickBackground` da classe `ICJoystickCompound`, passando as informações (x, y, largura, altura e caminho) referentes a imagem; 2. O desenvolvedor armazena o valor de retorno do passo 1; 3. O desenvolvedor informa a imagem superior do componente *joystick* usando o método `loadImagejoystickTop` da classe `ICJoystickCompound`, os parâmetros são os mesmos do passo 1, mas agora referentes a imagem superior; 4. O desenvolvedor armazena o valor de retorno do passo 3;
Exceção	No passo 1 e 3, caso o desenvolvedor informe o caminho errado da imagem é retornado um erro em tempo de execução informando que a imagem não foi carregada.
Pós-condição	As imagens são carregadas com sucesso.

Quadro 2 - Caso de uso UC01

3.2.1.1.2 UC02 - Inicializar configurações

O segundo caso de uso `UC02 – Inicializar configurações` (Quadro 3) descreve como o usuário-desenvolvedor prepara o componente para uso. Este caso de uso possui um cenário principal e um de exceção.

UC02 – Inicializar configurações: possibilita ao usuário-desenvolvedor preparar o componente para uso.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC01.
Cenário principal	1. O desenvolvedor inicializa as configurações do componente *joystick* usando o método `initialSettings` da classe `ICJoystickCompound`.
Exceção	No passo 1, caso a imagem de fundo e a imagem superior do *joystick* não tenham sido previamente carregadas uma mensagem de erro é mostrada.
Pós-condição	As configurações do componente são inicializadas com sucesso.

Quadro 3 - Caso de uso UC02

3.2.1.1.3 UC03 - Calcular aceleração

O caso de uso `UC03 – Calcular aceleração` (Quadro 4) explana como o usuário-desenvolvedor pode obter a aceleração do componente. Este caso de uso é composto somente por um cenário principal.

UC03 – Calcular aceleração: permite ao usuário-desenvolvedor obter a aceleração do componente.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC02.
Cenário principal	1. O desenvolvedor obtém a aceleração atual do componente através do método `getAcceleration` da classe `ICJoystickCompound`.
Pós-condição	É retornado um valor de aceleração entre 0 e 100.

Quadro 4 - Caso de uso UC03

3.2.1.1.4 UC04 - Retornar para a posição inicial

O quarto e último caso de uso `UC04 – Retornar para a posição` (Quadro 5), referente ao componente *joystick* descreve como retornar a imagem superior a sua configuração inicial.

UC04 - Retornar para a posição inicial: possibilita ao usuário-desenvolvedor retornar a imagem superior para sua posição inicial.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC03.
Cenário principal	1. O desenvolvedor utiliza o método `touchesEndedjoystick` da classe `ICJoystickCompound`, para retornar o componente *joystick* ao seu estado inicial.
Pós-condição	A imagem superior do *joystick* retorna a posição inicial ao ser encerrada o toque em tela neste componente.

Quadro 5 - Caso de uso UC04

3.2.1.2 Diagrama de caso de uso do componente barra de vida

Este componente possui três casos de uso, conforme pode ser observado na Figura 17 que apresenta o diagrama deste componente com as principais interações do usuário com o sistema.

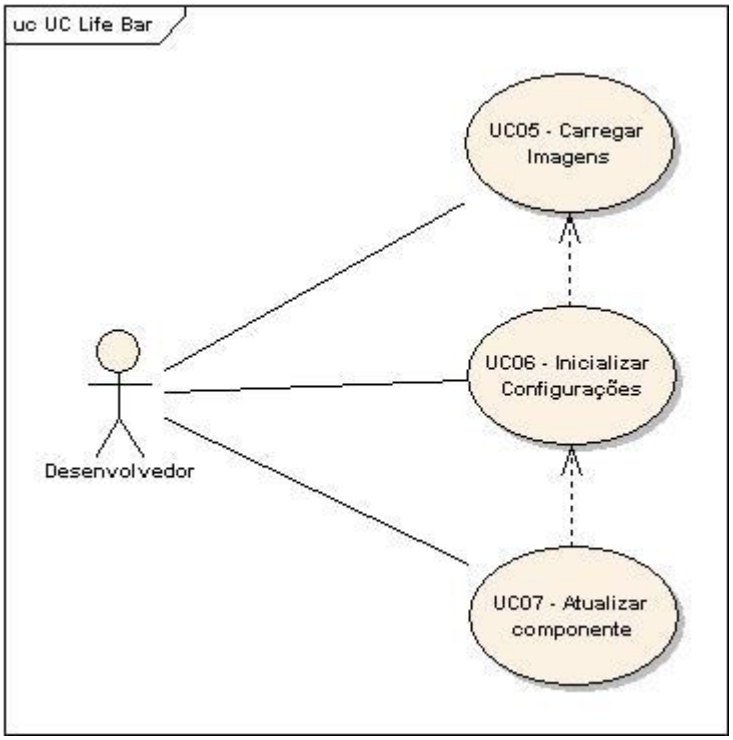


Figura 17 - Diagrama de casos de uso do componente barra de vida

3.2.1.2.1 UC05 – Carregar imagens

O caso de uso UC05 - Carregar Imagens (Quadro 6) é composto por um cenário principal e um de exceção e descreve como o usuário-desenvolvedor adiciona as imagens do componente barra de vida a sua aplicação.

UC05 – Carregar imagens: possibilita ao usuário-desenvolvedor adicionar a imagem de fundo e a imagem superior do componente barra de vida.	
Pré-condição	O desenvolvedor possuir uma imagem em formato PNG ou JPG.
Cenário principal	1. O desenvolvedor informa a imagem de fundo da barra de vida através do método `loadImageBarBackground` da classe `ICLifeBar`, passando as informações de x, y, largura, altura e caminho completo da imagem; 2. O desenvolvedor armazena o valor de retorno do passo 1; 3. O desenvolvedor informa a imagem superior do componente barra de vida usando o método `loadImageBarTop` da classe `ICLifeBar`, informando o caminho completo da imagem; 4. O desenvolvedor armazena o valor de retorno do passo 3;
Exceção	No passo 1 e 3, caso o desenvolvedor informe o caminho errado da imagem é retornado um erro em tempo de execução informando que a imagem não foi carregada.
Pós-condição	As imagens são carregadas com sucesso.

Quadro 6 - Caso de uso UC06

3.2.1.2.2 UC06 - Inicializar configurações

O caso de uso `UC06 – Inicializar configurações` (Quadro 7), propõe como o usuário-desenvolvedor prepara o componente barra de vida para uso.

UC06 – Inicializar configurações: possibilita ao usuário-desenvolvedor preparar o componente para uso.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC05.
Cenário principal	1. O desenvolvedor inicializa as configurações do componente barra de vida usando o método `initialSettings` da classe `ICLifeBar`, passando as informações referentes ao valor máximo de vida e por fim o valor da taxa de atualização da vida.
Exceção	No passo 1, caso o valor referente a taxa de atualização não seja válido, um valor padrão é adicionado no lugar do valor inválido, e um aviso é mostrado.
Pós-condição	As configurações do componente são inicializadas com sucesso.

Quadro 7 - Caso de uso UC06

3.2.1.2.3 UC07 – Atualizar Componente

O caso de uso `UC07 - Atualizar Componente` (Quadro 8), descreve como o usuário-desenvolvedor atualiza o componente barra de vida para uso.

UC07 – Atualizar Componente: possibilita ao usuário-desenvolvedor atualizar o componente.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC06.
Cenário principal	1. O desenvolvedor utiliza o método `updateBar` da classe `ICLifeBar`, para que a barra seja atualizada.
Pós-condição	O componente barra de vida é atualizado com sucesso.

Quadro 8 - Caso de uso UC07

3.2.1.3 Diagrama de caso de uso do componente barra de tempo

A Figura 18 apresenta o diagrama de casos de uso do componente barra de tempo com as principais interações do usuário com o sistema.

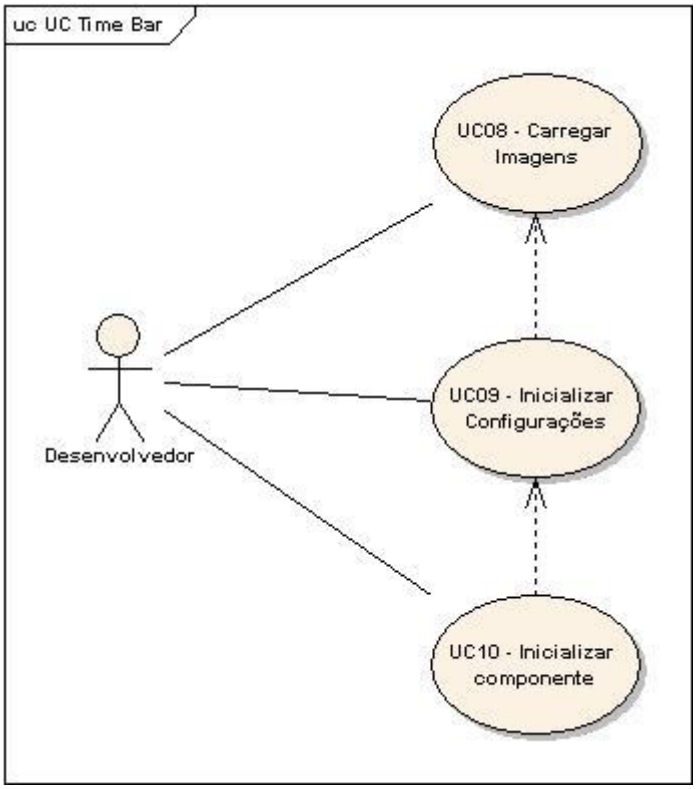


Figura 18 - Diagrama de casos de uso do componente barra de tempo

3.2.1.3.1 UC08 - Carregar imagens

O caso de uso `UC08 - Carregar Imagens` (Quadro 9) é composto por um cenário principal e um de exceção e descreve como o usuário-desenvolvedor adiciona as imagens do componente barra de tempo a sua aplicação.

UC08 – Carregar imagens: possibilita ao usuário-desenvolvedor adicionar a imagem de fundo e a imagem superior do componente barra de tempo.	
Pré-condição	O desenvolvedor possuir uma imagem em formato PNG ou JPG.
Cenário principal	1. O desenvolvedor informa a imagem de fundo da barra de tempo através do método `loadImageBarBackground` da classe `ICTimeBar`, passando as informações de x, y, largura, altura e caminho da imagem; 2. O desenvolvedor armazena o valor de retorno do passo 1; 3. O desenvolvedor informa a imagem superior do componente `time bar` usando o método `loadImageBarTop` da classe `ICTimeBar`, informando o caminho completo da imagem; 4. O desenvolvedor armazena o valor de retorno do passo 3;
Exceção	No passo 1 e 3, caso o desenvolvedor informe o caminho errado da imagem é retornado um erro em tempo de execução informando que a imagem não foi carregada.
Pós-condição	As imagens são carregadas com sucesso.

Quadro 9 - Caso de uso UC08

3.2.1.3.2 UC09 - Inicializar configurações

O caso de uso `UC09 - Inicializar configurações` (Quadro 10) propõe como o usuário-desenvolvedor prepara o componente barra de tempo para uso.

UC09 – Inicializar configurações: possibilita ao usuário-desenvolvedor preparar o componente para uso.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC08.
Cenário principal	1. O desenvolvedor inicializa as configurações do componente barra de tempo usando o método `initialSettings` da classe `ICTimeBar`, passando as informações referentes ao tempo máximo e o intervalo de atualização.
Pós-condição	As configurações do componente são inicializadas com sucesso.

Quadro 10 - Caso de uso UC09

3.2.1.3.3 UC10 - Inicializar Componente

O caso de uso `UC10 – Inicializar Componente` (Quadro 11) descreve como o usuário-desenvolvedor inicia e por conseqüência atualiza o componente barra de tempo.

UC10 – Iniciar Componente: possibilita ao usuário-desenvolvedor iniciar o componente.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC09.
Cenário principal	1. O desenvolvedor faz uso do método `inicializeBar` da classe `ICTimeBar`, para que a barra seja iniciada e em decorrência atualizada.
Pós-condição	O componente barra de tempo é iniciado com sucesso.

Quadro 11 - Caso de uso UC10

3.2.1.4 Diagrama de caso de uso do componente pedal de aceleração

A Figura 19 apresenta o diagrama de casos de uso do componente pedal de aceleração com as principais interações do usuário com o sistema.

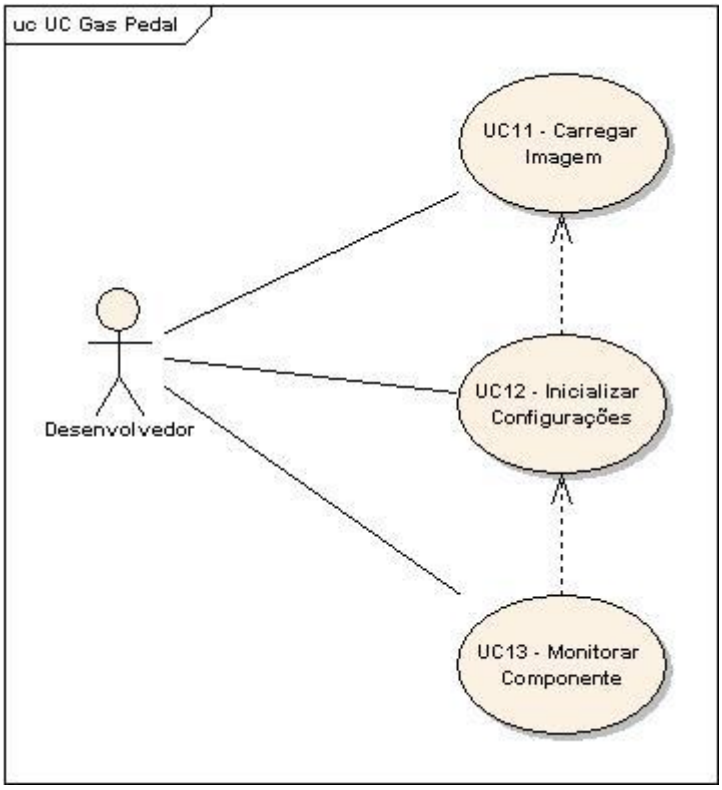


Figura 19 - Diagrama de casos de uso do componente pedal de aceleração

3.2.1.4.1 UC11 - Carregar imagem

O caso de uso `UC11 - Carregar Imagem` (Quadro 12) descreve como o usuário-desenvolvedor adiciona a imagem referente ao componente pedal de aceleração a sua aplicação.

UC11 – Carregar imagem: possibilita ao usuário adicionar a imagem do componente pedal de aceleração.	
Pré-condição	O desenvolvedor possuir uma imagem em formato PNG ou JPG.
Cenário principal	1. O desenvolvedor informa a imagem do componente pedal de aceleração através do método `loadImageGasPedal` da classe `ICGasPedal`, passando as informações de x, y, largura, altura e caminho da imagem; 2. O desenvolvedor armazena o valor de retorno do passo 1;
Exceção	No passo 1, caso o desenvolvedor informe o caminho errado da imagem é retornado um erro em tempo de execução informando que a imagem não foi carregada.
Pós-condição	A imagem é carregada com sucesso.

Quadro 12 - Caso de uso UC11

3.2.1.4.2 UC12 - Inicializar configurações

O caso de uso `UC12 – Inicializar configurações` (Quadro 13) propõe como o usuário-desenvolvedor prepara o componente pedal de aceleração para uso.

UC12 – Inicializar configurações: possibilita ao usuário-desenvolvedor preparar o componente para uso.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC11.
Cenário principal	1. O desenvolvedor inicializa as configurações do componente pedal de aceleração usando o método `initialSettings` da classe `ICGasPedal`, passando as informações referentes a taxa atualização, intervalo de atualização e se a opção de desaceleração deve ou não ser usada.
Pós-condição	As configurações do componente são inicializadas com sucesso.

Quadro 13 - Caso de uso UC12

3.2.1.4.3 UC13 - Monitorar Componente

O caso de uso `UC13 – Monitorar Componente` (Quadro 14) descreve como o usuário-desenvolvedor monitora o componente pedal de aceleração.

UC13 – Monitorar Componente: possibilita ao usuário-desenvolvedor monitorar este componente.	
Pré-condição	O desenvolvedor deve ter executado o caso de uso UC12.
Cenário principal	1. O desenvolvedor utiliza o método `touchGasPedal` para informar que o componente está sendo pressionado; 2. O desenvolvedor utiliza o método `touchEndedGasPedal` para informar quando o componente deixou de ser pressionado; 3. O desenvolvedor usa o método `getAcceleration` para receber o valor da aceleração deste componente; Todos os métodos são da classe `ICGasPedalBar`.
Pós-condição	O componente pedal de aceleração é monitorado com sucesso.

Quadro 14 - Caso de uso UC13

3.2.2 Diagrama de classes

Na Figura 20 é apresentado o diagrama de classes, sendo que este exibe as principais classes utilizadas neste projeto. Para melhor visualização, o diagrama de classes referente a cada componente será tratado em separado nas seções seguintes.

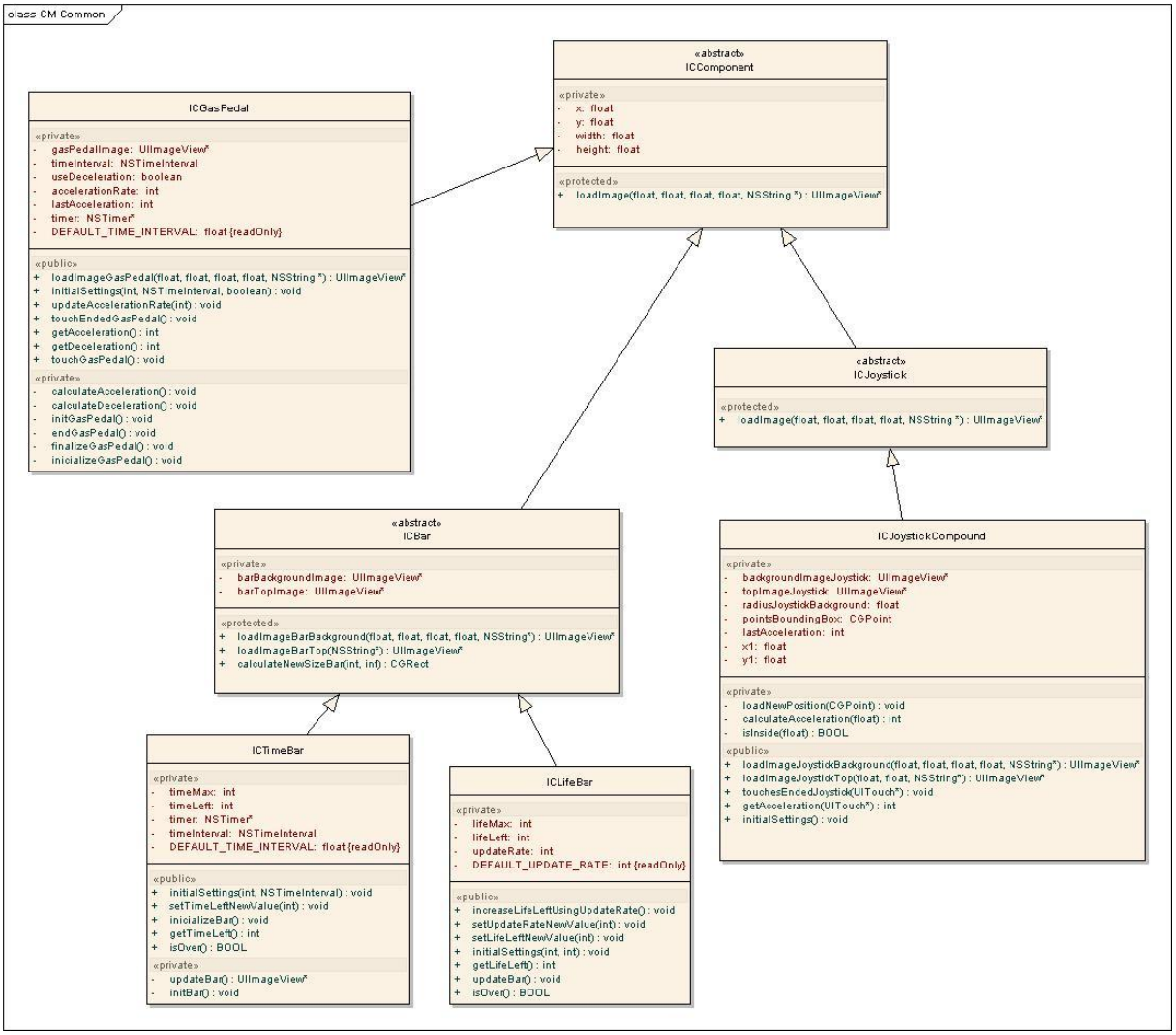


Figura 20 - Diagrama de classes da biblioteca de componentes

3.2.2.1 Classe ICComponent

A classe `ICComponent` é a classe raiz deste projeto, ela é uma classe conceitualmente abstrata que possui todos os métodos e atributos comuns a todos componentes. Está classe herda da classe UIView, pertencente ao *framework* UIKit. Como pode ser observada na Figura 21, a mesma é responsável por carregar as imagens e guardar as principais informações referentes à imagem.

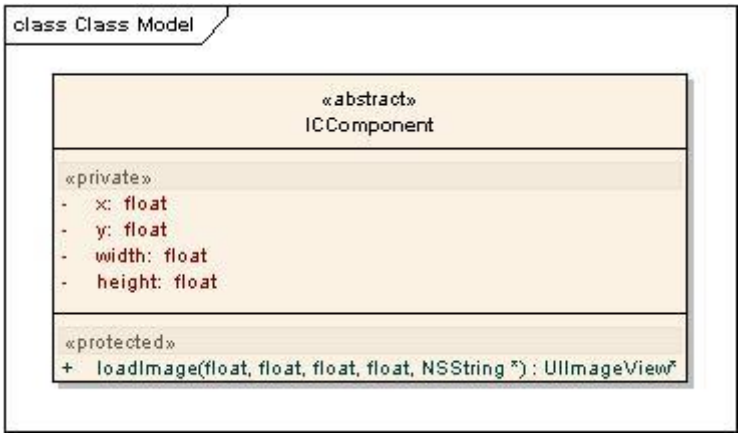


Figura 21 - Classe `ICComponent`

3.2.2.2 Diagrama de classe do componente barra de tempo

O componente barra de tempo é composto pelas classes `ICBar` e `ICTimeBar`. A Figura 22, mostra o diagrama de classe referente ao componente barra de tempo.

A classe `ICBar` é uma classe conceitualmente abstrata, que possui todos os métodos e atributos que são de comum uso para os componentes do tipo barra. Neste projeto esta classe tem por responsabilidade, trabalhar com a parte mais específica das imagens que representaram as barras. A mesma herda da classe abstrata `ICComponent`.

Já a classe `ICTimeBar` é a classe que tem todos os métodos necessários para a configuração e utilização do componente barra de tempo pelo usuário-desenvolvedor. Esta classe herda da classe abstrata `ICBar`. O método `getTimeLeft` retorna um inteiro com o tempo restante para a barra de tempo finalizar.

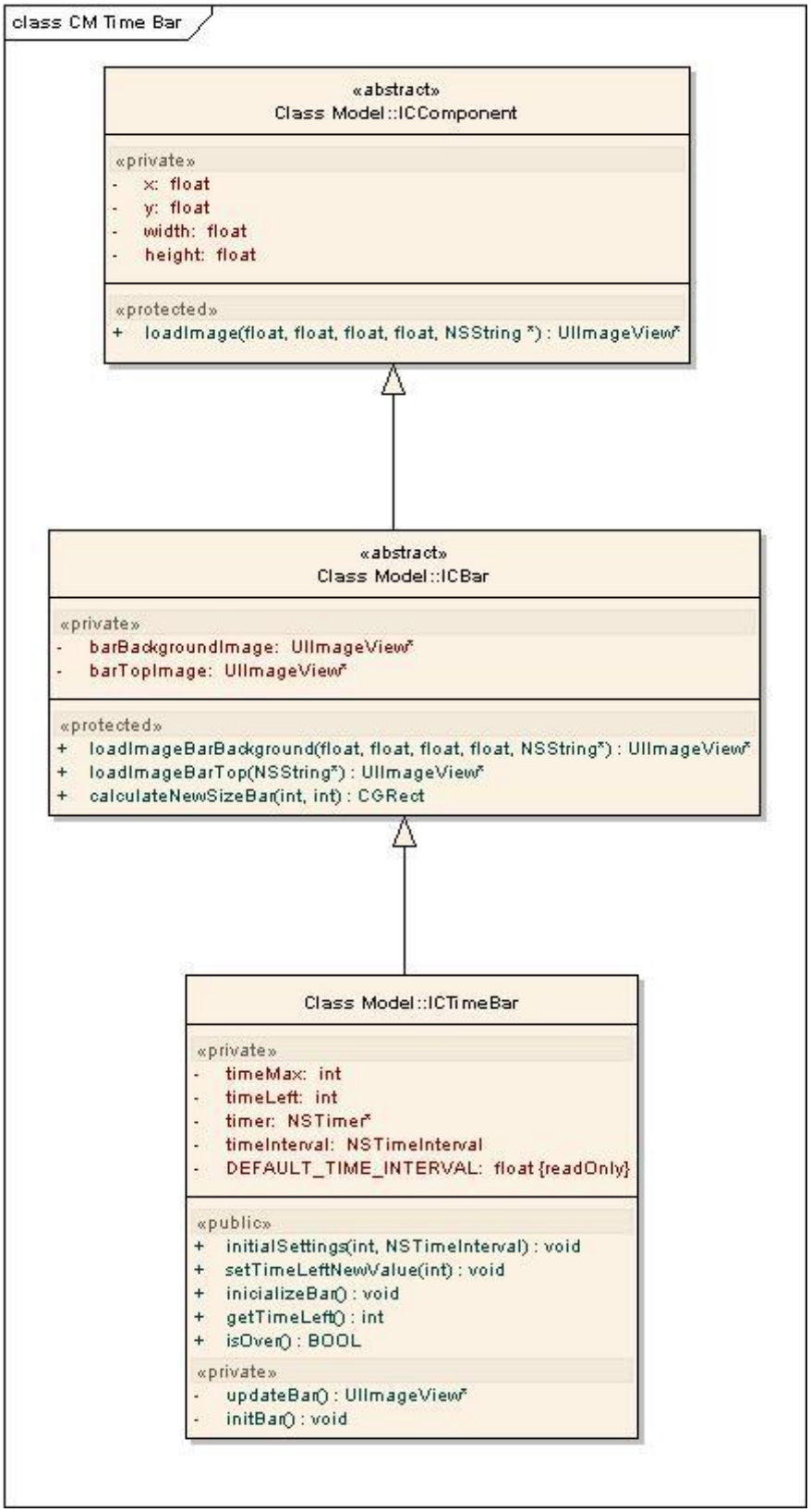


Figura 22 - Diagrama de classes referente ao componente barra de tempo

3.2.2.3 Diagrama de classe do componente barra de vida

A Figura 23, mostra o diagrama de classe referente ao componente barra de vida. O componente barra de vida é composto pelas classes `ICBar` e `ICLifeBar`. A classe `ICBar` foi descrita na seção 3.2.2.2.

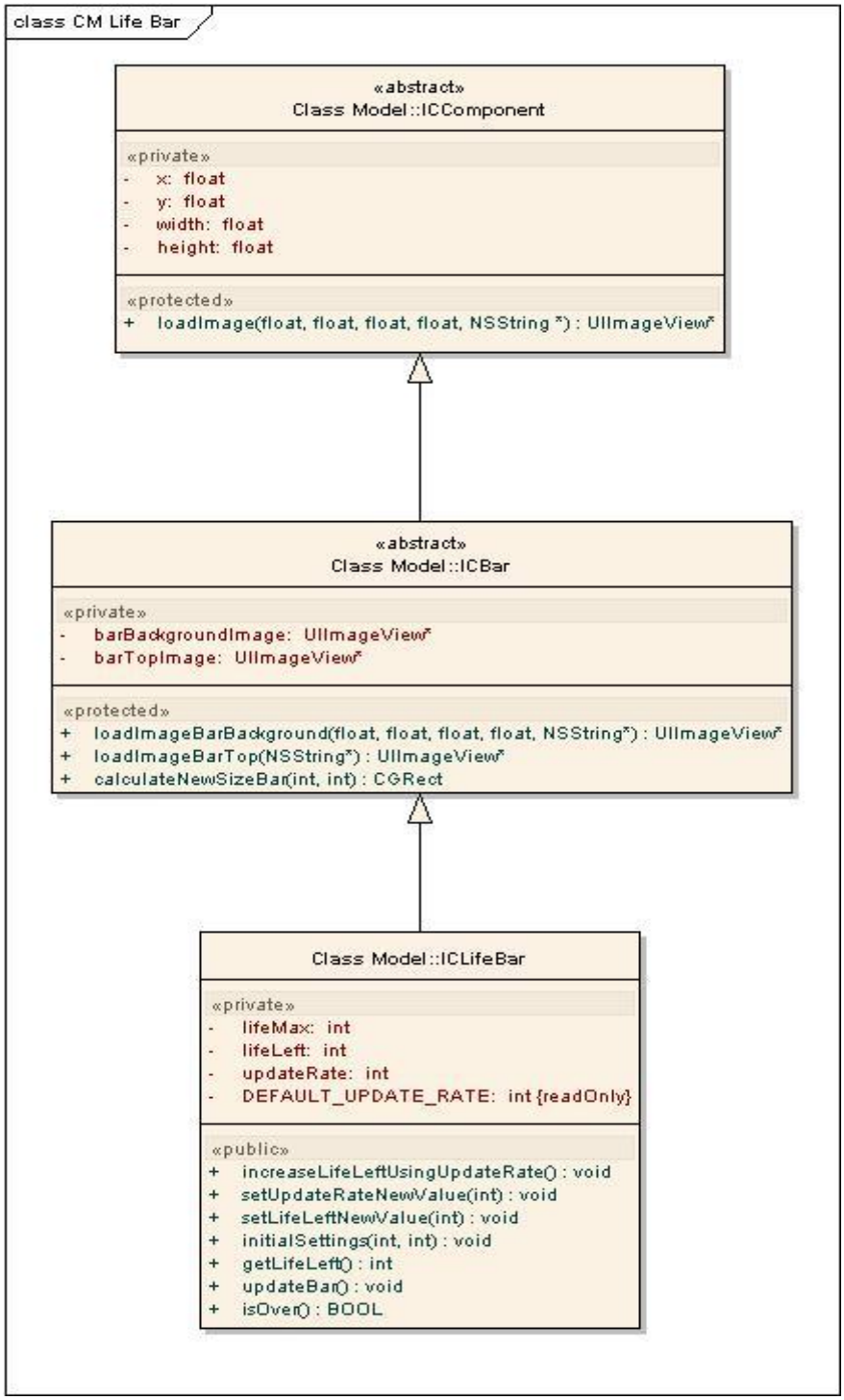


Figura 23 - Diagrama de classes referente ao componente barra de vida

A classe `ICLifeBar` possui todos os métodos necessários para a configuração e utilização do componente barra de vida pelo usuário-desenvolvedor. Através dos métodos `(int)getLifeLeft` e `(BOOL)isOver` o usuário consegue respectivamente controlar quanto falta para a barra de vida chegar ao fim ou se está já chegou ao final. Já o método `updateRate:(int)` permite que a taxa de atualização da perda de vida seja mudada a qualquer hora que o usuário assim desejar.

3.2.2.4 Diagrama de classe do componente pedal de aceleração

O diagrama de classe referente ao componente pedal de aceleração pode ser visto na Figura 24. A classe raiz `ICComponent`, foi anteriormente descrita na seção 3.2.2.1, sendo está *superclass* da classe `ICGasPedal`.

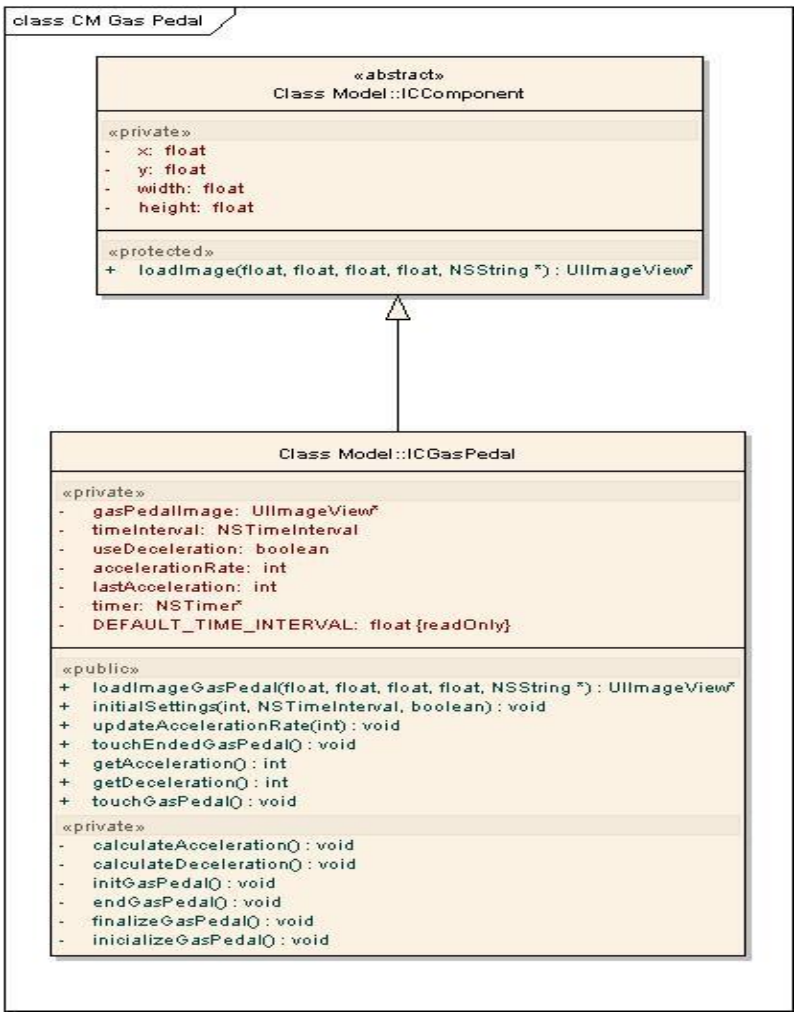


Figura 24 - Diagrama de classes referente ao componente pedal de aceleração

A classe `ICGasPedal` permite a criação de componente que representa um pedal de aceleração, permitindo ao usuário através do método `getAcceleration` obter um valor do tipo `float` referente a aceleração atual. Caso o usuário queira também é possível obter a desaceleração deste componente usando o método `getDeceleration` que retorna um valor do tipo `float`, até que a aceleração retorne a 0. Se a opção de desaceleração não for selecionada a aceleração é imediatamente zerada quando este componente deixa de ser usado.

3.2.2.5 Diagrama de classe do componente *joystick*

Pode ser observado na Figura 25, o diagrama de classe referente ao componente *joystick*.

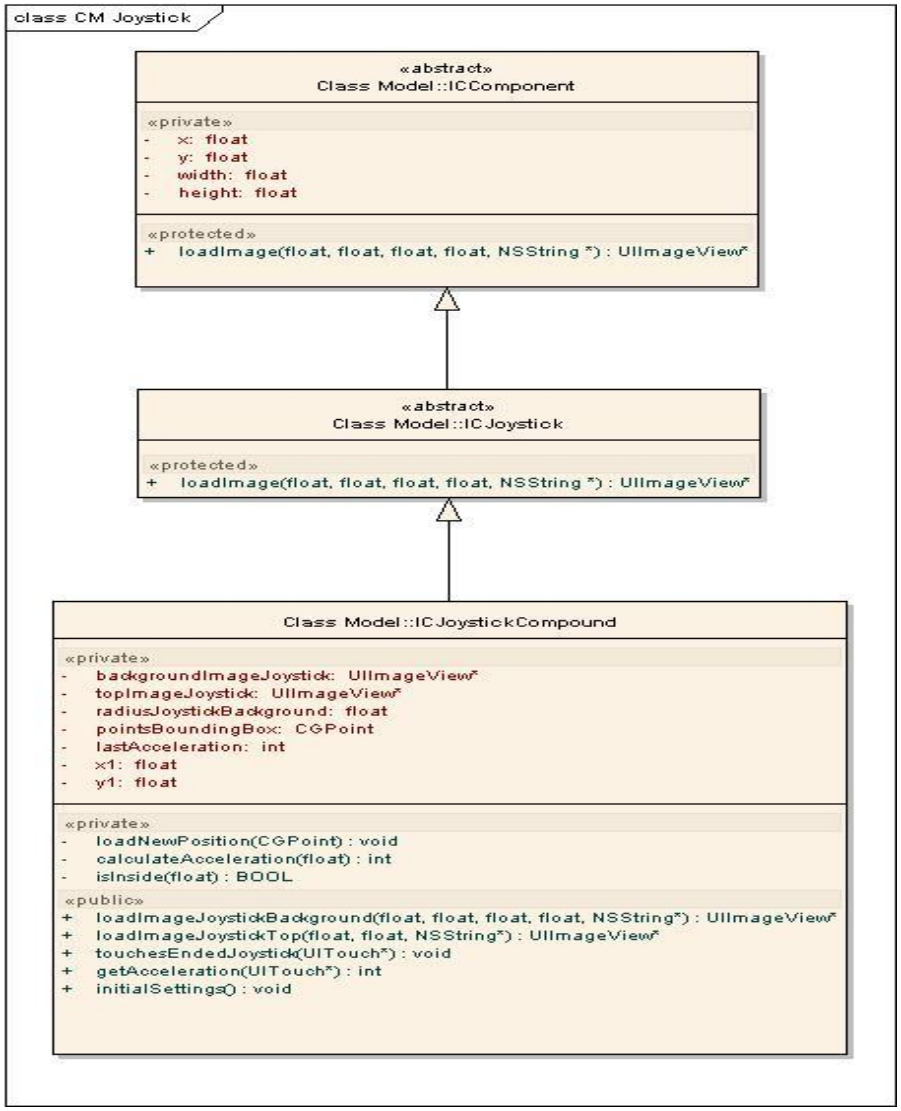


Figura 25 - Diagrama de classes referente ao componente *joystick*

A classe `ICjoystick` é uma classe conceitualmente abstrata, que possui todos os métodos e atributos comuns aos componentes do tipo *joystick*. Está classe herda da classe `ICComponent`.

A classe `ICJoystickCompound` é a classe responsável pela criação do componente do tipo *joystick*. Esta classe possui os métodos que o usuário-desenvolvedor precisará para criar e utilizar todas as funcionalidades deste componente. Os principais métodos públicos desta classe são o `initialSettings`, cuja finalidade é iniciar com valores específicos uma série de atributos da qual todo os outros métodos contidos nesta classe necessitam, o método `getAcceleration` que retorna a aceleração atual e calcula a nova posição do *joystick* superior.

Já os principais métodos privados são o `calculateAcceleration:(float)` cujo objetivo é calcular a aceleração dado diversos valores obtidos pelos outros métodos, o método `isInside:(float)` que é responsável por verificar se a imagem superior do componente *joystick* continua dentro da imagem externa do componente *joystick*.

3.3 IMPLEMENTAÇÃO

As técnicas e ferramentas utilizadas, bem como a operacionalidade da biblioteca desenvolvida neste trabalho, são apresentadas a seguir.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação deste trabalho foi utilizada a linguagem de programação Objective-C em conjunto com o ambiente de programação Xcode, ambos já descritos nas seções 2.4 e 2.5.1 respectivamente. Para a criação da documentação foi utilizada a ferramenta Doxygen, que é uma ferramenta para geração de documentação de código-fonte. Esta ferramenta é multi-plataforma e multi-linguagem, suportando atualmente as plataformas Windows, Mac e Linux e as linguagens C, C++, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP e C#.

A documentação é extraída diretamente do código-fonte comentado e também é

possível extrair a estrutura do código de códigos-fontes não documentados, sendo possível gerar diversos gráficos UML automaticamente, usando o relacionamento das classes envolvidas. A saída da documentação gerada é suportada nos formatos HTML, Latex, RTF, PostScript e PDF (HEESCH, 2009). A documentação deste trabalho encontra-se em Carmo (2009) e a documentação do componente *joystick* encontra-se no Anexo A.

Devido a impossibilidade do autor deste trabalho em obter um iPhone habilitado para receber novas aplicações, optou-se pela utilização do simulador oficial oferecido pela empresa Apple. Também será apresentado neste tópico às técnicas utilizadas e o código implementado.

3.3.1.1 iPhone Simulator

De acordo com Apple (2009i), o simulador para iPhone permite que as aplicações construídas no computador possam ser executadas, simulando como ficarão quando estiverem no dispositivo físico, isso permite que os problemas sejam previamente encontrados e que a interface do usuário possa ser testada antes que a aplicação seja liberada para o aparelho final. Ainda de acordo com a Apple (2099yi), quando o simulador está em execução o mesmo é apresentado em uma janela no formato de um iPhone, permitindo assim simular, por exemplo, o toque, o multi-toque, o movimento de arrastar e problemas com baixa memória.

3.3.1.2 Carregar a imagem

O Quadro 15 apresenta o código do método `loadImage` que é responsável por carregar as imagens. Este código encontra-se na classe abstrata `ICComponent` e é usado por todos os demais componentes.

O método `loadImage` na linha 18 recebe como parâmetro cinco valores sendo estes:

a) `x`: a posição em x onde a imagem será posicionada;
b) `y`: a posição em y onde a imagem será posicionada;
c) `width`: a largura que a imagem deverá ter;
d) `height`: a altura que a imagem deverá ter;
e) `imagePath`: o caminho completo de onde se encontra a imagem.

Nas linhas 19 a 22 do Quadro 14 os valores recebidos por parâmetro são armazenados

em atributos de mesmo nome. Na linha 26 é criado e inicializado uma nova visão. Os valores utilizados para a inicialização são os valores que foram armazenados pelas linhas 19 a 22.

Na linha 27 é utilizado o caminho completo da imagem passado por parâmetro, para retornar um objeto do tipo imagem, que é acrescentado na visão anteriormente criada. Nas linhas 29 a 31 é lançada uma exceção para assegurar que caso a imagem não possa ser carregada, o usuário-desenvolvedor seja obrigado a corrigir o problema, pois várias informações são extraídas das imagens por todos os componentes, sendo assim, elas devem ser carregadas corretamente.

```
18  -(UIImageView *)loadImage: (float)x : (float)y : (float)width : (
    float)height : (NSString *)imagePath  {
19      [self setX: x];
20      [self setY: y];
21      [self setWidth: width];
22      [self setHeight: height];
23
24
25      //Make a frame - The first two coordinates are X, Y and the next
          are width, height (width, height affect the image).
26      UIImageView *imageView = [[UIImageView alloc] initWithFrame:
    CGRectMake(self.x, self.y, self.width, self.height)];
27      imageView.image = [UIImage imageWithContentsOfFile: imagePath];
28
29      if(imageView.image == nil){
30          [NSException raise:@"Problem while loading image." format:@
    "Image %@ could not be loaded.", imagePath];
31      }
32
33      return imageView;
34  }
```

Quadro 15 - Código fonte do método `loadImage`

3.3.1.3 Calcular a distância entre as imagens do componente *joystick*

O método `getAcceleration` descrito no Quadro 16, apresenta como é calculada a distância entre a imagem de fundo e a imagem superior do componente *joystick*.

```
187 -(int)getAcceleration : (UITouch *)touch {
188     CGPoint location = [touch locationInView:self];
189 
190     [self topImageJoystick].userInteractionEnabled = YES;
191 
192     // Find the center point (X,Y) from top image joystick
193     // Same as:  xcenter = ((xmax - xmin) / 2) + xmin where xmax is equals
    to: xmin+largura
194     float x2 = [self topImageJoystick].center.x;
195     float y2 = [self topImageJoystick].center.y;
196 
197 
198     //Same as: (x2-x1)^2 + (y2-y1)^2
199     float distance = ((x2 - x1) * (x2 - x1)) + ((y2 - y1) * (y2 - y1));
200 
201 
202     //Test using only the distance
203     if([self isInside: distance]){
204         [self loadNewPosition: location];
205         [self setLastAcceleration: [self calculateAcceleration: distance]];
206         return [self lastAcceleration];
207 
208     }else { // If its outside of the big image, keep the last acceleration
209         return [self lastAcceleration];
210     }
```

Quadro 16 - Código fonte do método `getAcceleration` da classe `ICJoystickCompound`

Um exemplo de imagem superior e inferior que formam o componente *joystick* pode ser observado na Figura 26.

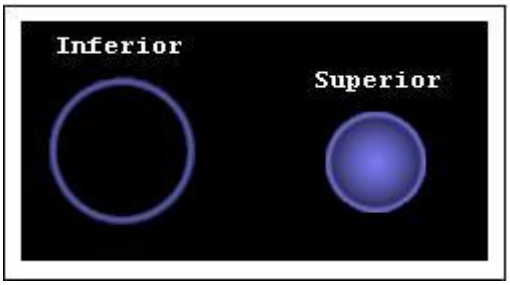


Figura 26 - Imagem de fundo e a imagem superior para componente joystick

Nas linhas 194 e 195 é encontrado o x, y do centro da imagem superior deste componente, esses valores encontrados serão usados na linha 199 onde em conjunto com o x, y do centro da imagem de fundo a distância é calculada.

Entre as linhas 203 a 210 é verificado se a imagem superior encontra-se dentro da imagem de fundo, retornando assim a aceleração.

3.3.1.4 Calcular a aceleração do componente *joystick*

A aceleração é encontrada usando o método privado `calculateAcceleration` recebendo como parâmetro o valor referente a distância que é encontrada no método `getAcceleration` (ver seção 3.3.1.3). Na linha 337 do Quadro 17, o cálculo da aceleração continua e o valor é dividido pelo raio elevado ao quadrado e na próxima linha o valor final da aceleração é retornado.

```
335 -(int)calculateAcceleration: (float)distance{
336     float acceleration = distance * 100;
337     acceleration = acceleration / [self radiusDoubleJoystickBackground];
338     acceleration = round(acceleration);
339     return (int)acceleration;
340 }
```

Quadro 17 - Código fonte do método `calculateAcceleration` da classe `ICJoystickCompound`

3.3.1.5 Validar a taxa de atualização do componente barra de vida

No Quadro 18, na linha 81 o método `setUpdateRateNewValue` recebe um valor referente a taxa de atualização, que é o valor referente à quantidade de vida que será decrementada da quantidade total de vidas informada. Na linha 82 é validado se esta taxa é maior que 0 e menor ou igual ao valor máximo de vidas informado pelo usuário-desenvolvedor. Se essa validação for verdadeira, na linha 83 é calculado o módulo entre a quantidade máxima de vida e o novo valor informado, já na linha 84 verifica se é múltiplo, se for a taxa de atualização é armazenada, para futuro uso. Caso não passe na validação da linha 82 um valor pré-definido é armazenado no lugar do valor inválido e uma mensagem informando o que aconteceu é mostrada ao usuário. Uma mensagem informando que a taxa de atualização não foi modificada será mostrada ao usuário caso a validação da linha 84 falhe, conforme pode ser observado na linha 86.

```
81  -(void)setUpdateRateNewValue: (int) updateRate {
82      if((updateRate > 0) && (updateRate <= [self lifeMax])){
83          int mod = [self lifeMax] % updateRate;
84              if(mod == 0){
85                  [self setUpdateRate: updateRate];
86              }else{
87                  NSLog(@"The new value %d is not allowed for update rate. The value must be
    multiple of %d (Max Life).  The update rate value was not changed.", updateRate, [self
    lifeMax]);
88          }
89      }else{
90          [self setUpdateRate: DEFAULT_UPDATE_RATE];
91          NSLog(@"The update rate %d is higher than allowed. The default update rate %d
    is used.", updateRate, DEFAULT_UPDATE_RATE);
92      }
93  }
```

Quadro 18 - Código fonte do método setUpdateRateNewValue da classe ICLifeBar

3.3.1.6 Calcular o tamanho da barra do componente barra de vida e barra de tempo

Os componentes barra de vida e barra de tempo possuem em comum o método `calculateNewSizeBar` que conforme pode ser visto no Quadro 19 na linha 31 recebe como parâmetro um valor para a quantidade de vidas restantes e um valor para a quantidade máxima de vidas. Esses valores serão empregados na linha 32, onde será obtida a nova largura da barra através do tamanho original da barra multiplicado pelo valor de vida restante e dividido pelo valor máximo de vidas. Por fim nas linhas 33 e 34 será criada uma nova barra com a nova largura e essa nova barra é retornada.

```
31  -(CGRect)calculateNewSizeBar: (int)leftValue : (int)maxValue {
32      float newWidth = ([self barBackgroundImage].frame.size.width * leftValue) /
    maxValue;
33      CGRect newSize = CGRectMake([super x], [super y], newWidth, [super height]);
34      return newSize;
35  }
```

Quadro 19 - Código fonte do método `calculateNewSizeBar` da classe `ICBar`

3.3.1.7 Inicializa o componente barra de tempo

Na linha 90 do Quadro 20, o método público `inicializeBar` da classe `ICTimerBar`, designa um *timer* que irá monitorar o método `initBar` no intervalo de tempo definido pelo atributo de nome `timeInterval`. O atributo `timeInterval` representa o valor em segundos pelo qual se deseja que o *timer* verifique o método especificado, neste caso o método

initBar. Por exemplo, se o valor do timeInterval for de 0.05 segundo, o método initBar será chamado duas vezes por segundo. O valor do atributo timeInterval é especificado pelo usuário-desenvolvedor em outro método.

```
89  -(void)inicializeBar {
90      timer = [NSTimer scheduledTimerWithTimeInterval:timeInterval target:self
    selector:@selector(initBar) userInfo:nil repeats:YES];
91
92  }
93
```

Quadro 20 - Código fonte do método inicializeBar

3.3.1.8 Valida a inicialização do componente barra de tempo

No método privado initBar especificado no Quadro 21, é verificado se a quantidade de vidas restantes se esgotou, em caso afirmativo o timer que foi inicializado no método inicializeBar (ver seção 3.3.1.7) é invalidado, ou seja, é desligado e para de monitorar este método, isto pode ser observado na linha 96. Caso ainda reste vidas o método updateBar é acionado e se responsabiliza por fazer o que for necessário para obter os valores necessários para que a barra de tempo seja atualizada.

```
94  -(void)initBar {
95      if([self timeLeft] == 0){
96          [timer invalidate];
97      }else{
98          [self updateBar];
99          [self refresh];
100     }
101
102 }
```

Quadro 21 - Código fonte do método initBar da classe ICTimerBar

3.3.1.9 Inicializa as principais configurações do componente pedal de aceleração

A classe ICGasPedal possui o método público initialSettings que recebe como parâmetro um valor para a taxa de aceleração, um valor em segundos para o intervalo de tempo e o último parâmetro é se a desaceleração do componente estará ligada. A taxa de aceleração é o valor que será somado à velocidade toda vez que for necessário, por exemplo, a

velocidade inicia em 0 e o usuário-desenvolvedor deseja que ela seja incrementada de cinco em cinco, então o mesmo terá que informar o valor 5 para este parâmetro.

O parâmetro para o intervalo de tempo é em segundos e se refere a quanto em quanto tempo a aceleração deve mudar, por exemplo, se o valor passado for 0.05 então a cada segundo a aceleração será incrementada por duas vezes com o valor informado no parâmetro anterior. Por fim o usuário-desenvolvedor informa se deseja que a desaceleração esteja ligada, caso opte por ligar a desaceleração isso significa que quando o componente pedal de aceleração deixar de ser usado, a aceleração irá decair gradativamente usando o valor do primeiro parâmetro e também o intervalo de tempo informado no segundo parâmetro.

Na linha 108 do Quadro 22 se o valor referente ao intervalo de tempo informado pelo usuário for igual a 0, este será invalidado e o valor padrão para um intervalo será armazenado no lugar do valor inválido e o usuário receberá uma mensagem avisando do ocorrido.

```
101 -(void)initialSettings:(float)accelerationRate : (NSTimeInterval)timeInterval :
    (BOOL) useDeceleration {
102     if([self gasPedalImage].image == nil){
103         [NSException raise:@"Image not loaded." format:@"Gas Pedal's image
    image not loaded."];
104     }else {
105         [self setLastAcceleration: 0];
106         [self setAccelerationRate: accelerationRate];
107         [self setUseDeceleration: useDeceleration];
108         if(timeInterval == 0){
109             [self setTimeInterval: DEFAULT_TIME_INTERVAL];
110         }else{
111             [self setTimeInterval: timeInterval];
112         }
113     }
114 }
```

Quadro 22 - Código fonte do método `initialSettings`

3.3.1.10 Atualiza o componente pedal de aceleração

Toda vez que o componente pedal de aceleração for acionado o método `touchGasPedal` será acionado e se responsabilizará por invalidar o *timer* anterior e logo após acionara o método `inicializeGasPedal` que irá disparar o *timer* novamente e o monitoramento deste componente reiniciara e esse comportamento pode ser observado entre as linhas 83 e 87 do Quadro 23.

```
82  -(void)touchGasPedal {
83      if(aux == TRUE) {
84          [timer invalidate];
85      }
86      [self setMove: TRUE];
87      [self inicializeGasPedal];
88  }
```

Quadro 23 - Código fonte do método `touchGasPedal` da classe `ICGasPedal`

3.3.1.11 Calcula a desaceleração do componente pedal de aceleração

Na linha 155 do método `calculateDeceleration` descrito no Quadro 24, o valor da aceleração é o valor da última aceleração menos o valor da taxa de aceleração que já foi explanada na seção 3.3.1.9. Isto só ocorrerá se a aceleração for maior que 0, está verificação é feita na linha 154. Se a aceleração já for igual a 0, então o timer é invalidado, pois não há mais o que desacelerar.

```
153 -(void)calculateDeceleration {
154     if((move == FALSE) && ([self lastAcceleration] > 0)){
155         [self setLastAcceleration: [self lastAcceleration] -
    [self accelerationRate]];
156         [self setAux: TRUE];
157     }else{
158         [self setAux: FALSE];
159         [timer invalidate];
160     }
161 }
```

Quadro 24 - Código fonte do método `calculateDeceleration` da classe `ICGasPedal`

3.3.2 Operacionalidade da implementação

Esta seção tem por objetivo mostrar a operacionalidade da implementação em nível de usuário, para isto será exemplificado o desenvolvimento do componente *joystick*, barra de vida, barra de tempo e pedal de aceleração. Para facilitar o entendimento e permitir o uso de todos os métodos disponíveis ao usuário, cada componente será exemplificado separadamente, e o retorno ao usuário será mostrado no *console*. Por fim será apresentado um exemplo dos componentes em um jogo, demonstrando sua utilidade prática. Toda linha de

código que alterar o estado visual da biblioteca terá uma figura demonstrativa da alteração.

3.3.2.1 Um exemplo simples usando o componente *joystick*

Nesta seção será mostrado como incluir corretamente um componente *joystick*, e como configurar suas funcionalidades básicas. O usuário somente tem acesso aos métodos públicos da classe `ICJoystickCompound` e eles fornecem tudo o que é preciso para implementar o componente *joystick* em um projeto. O Quadro 25, apresenta a classe `ICExampleJoystick` já com o componente *joystick* implementado.

```
1  @implementation ICExampleJoystick
2  @synthesize backgroundImagejoystick;
3  @synthesize topImagejoystick;
4  @synthesize joystickCompound;
5
6  - (id)initWithFrame:(CGRect)frame {
7      if (self = [super initWithFrame:frame]) {
8
9          // Inicialize ICJoystickCompound class
10         joystickCompound = [ICJoystickCompound alloc];
11
12         // load joystick background image
13       backgroundImagejoystick = [joystickCompound
14 loadImagejoystickBackground:30 :50 :100 :100
15 :@"/Users/bell_cristina/Desktop/joystick/ballOutImage.png"];
16
17       // load joystick top image
18       topImagejoystick = [joystickCompound loadImagejoystickTop:45
19 :45 :@"/Users/bell_cristina/Desktop/joystick/ballInImage.png"];
20
21       [joystickCompound initialSettings];
22     }
23     return self;
24 }
25
26 - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
27       UITouch *touch = [[event allTouches] anyObject];
28
29       if([touch view] == topImagejoystick){
30             [joystickCompound touchesEndedjoystick: touch];
32       }
32 }
33
34 - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
35       UITouch *touch = [[event allTouches] anyObject]; //detect the
36 location of the user's touch
37
38       // Enable User Interaction for top image of the joystick
39       topImagejoystick.userInteractionEnabled = YES;
40
41       //Check if user touched the top image of the joystick
42       if([touch view] == topImagejoystick){
43
44           int acceleration = [joystickCompound getAcceleration:
45 touch];
46           NSLog(@"Aceleração: %d", acceleration);
47       }
48 }
49
50 - (void)drawRect:(CGRect)rect {
51     // Drawing code
52       [self addSubview:backgroundImagejoystick];
53       [self addSubview:topImagejoystick];
54 }
56
57 - (void)dealloc {
58     [super dealloc];
59 }
60 @end
```

Quadro 25 - Exemplo de implementação do componente *joystick*

Na linha 10 do Quadro 25 a classe `ICJoystickCompound` é instanciada, já na linha 13 é onde é definido qual será a imagem de fundo do *joystick*, seu posicionamento e tamanho. Sendo neste caso o valor 30 para x, 50 para y, 100 para a largura e 100 para a altura e por fim o caminho completo da imagem incluído a extensão da mesma. Devido aos valores exemplos de x e y informados acima o *joystick* será criado no quadrante 1. A localização do quadrante 1 pode ser observado na Figura 27.

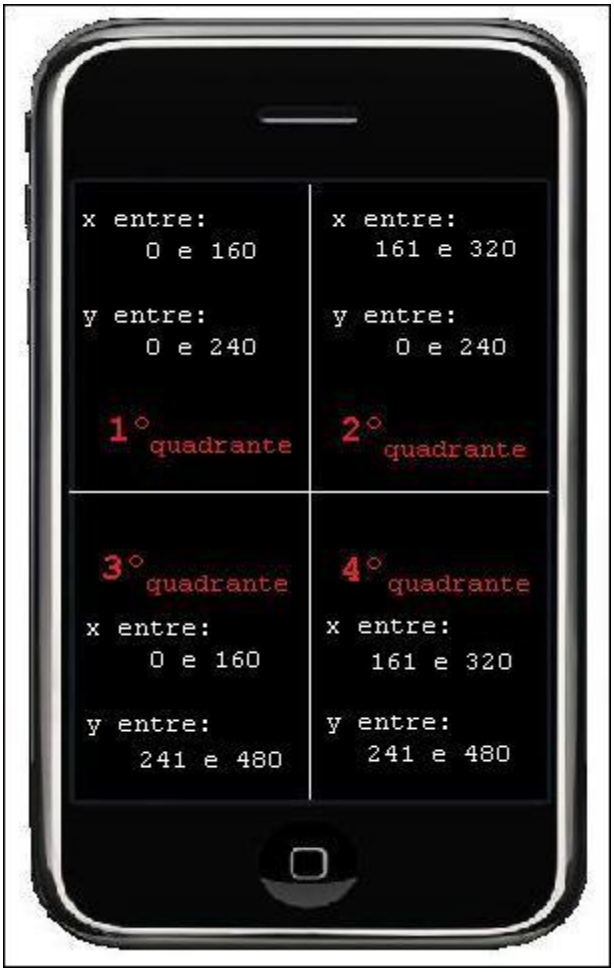


Figura 27 - Localização dos quadrantes no iPhone

Na linha 18 do Quadro 25 será criada a imagem que representa a parte superior do *joystick*, mas ao contrário da imagem anteriormente criada, nesta será necessário informar apenas uma valor para largura, um para a altura e o caminho da imagem. Para a imagem superior os valores de `x` e `y` não são informados, pois a biblioteca internamente se responsabilizara por inserir a imagem superior no centro da imagem de fundo do componente *joystick*. Na linha 21 é chamado o método responsável por fazer algumas configurações internas que são extremamente necessárias para o correto funcionamento do *joystick*. Na linha 51 e 52 é solicitado que as imagens anteriormente criadas sejam mostradas na tela. Fica a cargo do usuário ser responsável por escolher quando deseja ou precisa que o componente

seja mostrado.

Com as poucas linhas de código explicadas acima já é possível obter o componente *joystick* em tela, mas o mesmo estará estático, ou seja, a imagem superior não conseguirá se movimentar. Para que o *joystick* esteja totalmente funcional ainda há mais 3 linhas de código que precisam ser feitas, uma delas é o código da linha 39 que diz que a imagem superior está habilitada a receber interação de toque na tela.

Para oferecer maior flexibilidade optou-se deixar a cargo do usuário habilitar o componente *joystick* para interação quando o usuário desejar, assim sendo, o código da linha 39 poderia estar localizado em outro lugar, de acordo com a lógica do usuário. O código `[joystickCompound getAcceleration: touch]` da linha 44 é muito importante, pois é esta linha que permite receber a aceleração do componente *joystick* e também permite que o mesmo se movimente. Esta linha deve ser acionada sempre que ocorrer um movimento com este componente. Para este exemplo, visando a simplicidade, preferiu-se colocar esta linha dentro do método `touchesMoved` do próprio SDK do iPhone, sendo este método chamado automaticamente sempre que ocorra algum movimento causado por um toque na tela no componente.

Por fim o conteúdo da linha 30 também é um código importante, pois aciona o método que é responsável por informar ao componente *joystick* que ele deve voltar à posição original, imediatamente após perder a interação de toque, ou seja, o usuário soltou o toque que havia neste componente. Após as linhas de códigos explanadas acima, o *joystick* estará pronto para uso, restando ao usuário programar o resto da lógica da sua aplicação. Para mostrar que o componente *joystick* implementado no Quadro 25 funciona, será utilizada a linha 45. Nesta linha será impresso no *console* o valor da aceleração retornado pela linha 44, como pode ser observado na Figura 28. Na Figura 29 o componente *joystick* aparece sendo arrastado em várias direções.

Figura 28 - Componente *joystick* retornando a aceleração

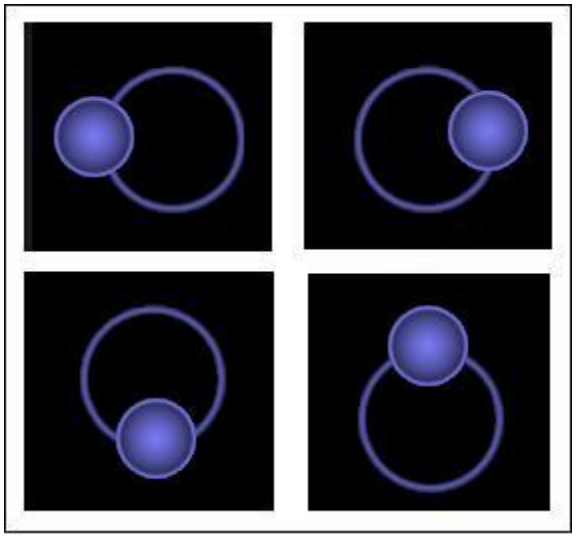


Figura 29 - Componente *joystick* arrastado em várias direções

3.3.2.2 Um exemplo simples usando o componente barra de vida

Esta seção apresentará como criar corretamente um componente barra de vida. O usuário fará uso dos métodos públicos da classe `ICLifeBar`. O Quadro 25, apresenta a classe `ICExampleLifeBar` já com o componente barra de vida implementado.

Na linha 11 do Quadro 26 a classe `ICLifeBar` é instanciada e na linha 14 é definido qual será a imagem de fundo da barra de vida, a imagem de fundo para a barra de vida é aquela que representará a perda de vidas. Os parâmetros são os mesmos usados no componente *joystick*, descrito na seção 3.3.2.1, assim sendo, para este exemplo os valores de `x` e `y` serão 30 e 70 respectivamente, já largura será de 260 e a altura terá o valor 20.

```
1   #import "ICExampleLifeBar.h"
2
3   @implementation ICExampleLifeBar
4         @synthesize barBackgroundImage;
5         @synthesize barTopImage;
6         @synthesize lifeBar;
7
8   - (id)initWithFrame:(CGRect)frame {
9       if (self = [super initWithFrame:frame]) {
10         // Inicialize ICExampleLifeBar class
11         lifeBar = [ICLifeBar alloc];
12
13         // load background image bar
14         barBackgroundImage = [lifeBar loadImageBarBackground: 30.0:
15  70.0: 260.0: 20.0:
16  @"/Users/bell_cristina/Desktop/joystick/barBackgroundImage.png"];
17
18        // load top image bar
19        barTopImage = [lifeBar loadImageBarTop:
20  @"/Users/bell_cristina/Desktop/joystick/barTopImage.png"];
21
22        [lifeBar initialSettings: 50: 10];
23      }
24      return self;
25  }
26
27  - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
28        [lifeBar updateBar];
29  }
30
32  - (void)drawRect:(CGRect)rect {
32      // Drawing code
33        [self addSubview:barBackgroundImage];
34        [self addSubview: barTopImage];
35  }
37  @end
```

Quadro 26 - Exemplo de implementação do componente barra de vida

Na linha 19 é determinada a imagem superior da barra de vida, ou seja, a imagem que representa as vidas restantes. Para este método será preciso informar apenas o caminho completo da imagem, pois as demais informações serão copiadas da imagem criada na linha 14. A Figura 30 mostra um exemplo de uma imagem de fundo para a linha 14 e o exemplo de uma imagem superior para a linha 19.

Figura 30 - Exemplo de barra de vida vazia e cheia

A linha 22 se encarrega de inicializar algumas configurações básicas que são necessárias para este componente, para que isto funcione é preciso informar dois valores relacionados aos parâmetros. O primeiro parâmetro é referente a quantidade máxima de vidas que será disponibilizada, já o segundo parâmetro é um valor para a taxa de atualização. A taxa de atualização é um valor referente à quantidade de vida que será decrementada da quantidade máxima de vidas informada. Neste exemplo os valores 50 e 10 foram usados, o que quer dizer que a quantidade máxima de vida disponível será 50 e que cada vez que uma vida for perdida serão decrementados 10 pontos de vida.

Por fim o código da linha 28 é responsável por atualizar a barra, é ele que vai descontar a vida perdida e calcular como a barra de vida se parecerá após isto. Este código deve ser chamado sempre que o usuário desejar alterar a barra de vida descontando uma vida da mesma. Neste exemplo o código encontra-se dentro do método `touchesBegan` que significa que cada vez que houver um toque em tela, uma vida será perdida.

A Figura 31 mostra na primeira parte como ficará graficamente a barra de vida com a implementação acima e como esta se parecerá após haver um toque na tela e serem descontados 10 pontos de vida da mesma.

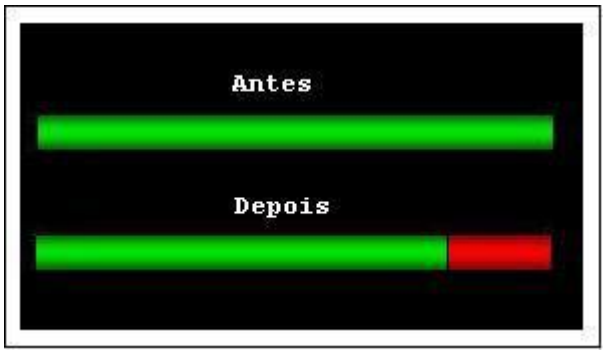


Figura 31 - Antes e depois do componente barra de vida

O componente barra de vida também disponibiliza um método chamado `getLifeLeft` que retorna ao usuário quanto ainda resta de vida. Também é possível mudar a quantidade de vida restante usando o método `setLifeLeftNewValue` passando como parâmetro a nova quantidade de vida restante. O novo valor deve obedecer algumas regras como ser maior que 0 e menor ou igual ao valor máximo de vidas e o novo valor também deve ser múltiplo da taxa de atualização.

Para exemplificar a funcionalidade destes dois novos métodos, o método `touchesBegan` do Quadro 26 será modificado para suportar estas duas novas funcionalidades. O novo método `touchesBegan` contido no Quadro 27 deverá substituir o método antigo e ao fazer isso o que acontecerá é que a cada toque na tela serão decrementados 10 pontos de

vidas, isso ocorrerá até que a quantidade de vidas restantes seja igual a 20 pontos. Quando os 20 pontos forem atingidos a quantidade de vida será atualizada para o novo valor informado pelo usuário.

```
1  - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
2        [lifeBar updateBar];
3        NSLog(@"Life Left: %d", [lifeBar getLifeLeft]);
4        if([lifeBar getLifeLeft] == 20){
5              [lifeBar setLifeLeftNewValue: 40];
6        }
7  }
```

Quadro 27 - Inclusão de novas funcionalidades na classe exemplo `ICExampleLifeBar`

Na linha 4 do Quadro 27 é utilizado o método `getLifeLeft` para recuperar a quantidade atual de vidas restantes e se este valor for igual a 20, a linha 5 é executada. Na linha 5 o método `setLifeLeftNewValue` como já explicado, irá substituir a quantidade de vidas restantes pelo novo valor informado, neste exemplo o novo valor será 40. Isto significa que a partir da linha 5 a quantidade de vidas restantes passou de 20 para 40 pontos de vidas. A Figura 32 mostra a barra de vida antes da modificação da quantidade de vida e o depois da modificação quando a barra de vida passou a ter 40 pontos.

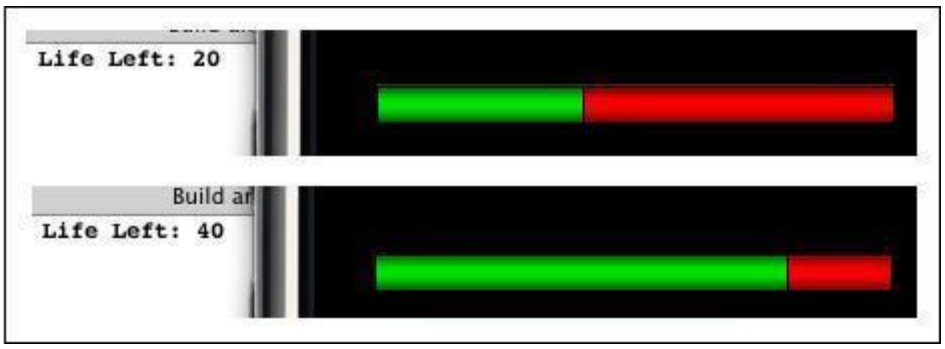


Figura 32 - Componente barra de vida antes e após o método setLifeLeftNewValue

Por fim há mais duas funcionalidades para o componente barra de vida. O método `isOver` retorna `true` caso a quantidade de vidas seja igual a 0 e `false` caso ainda restem pontos de vida. Para exemplificar esta funcionalidade foi adicionado o método `touchesEnded` a classe exemplo, este método será executado sempre que um toque em tela for liberado. Na linha 2 do Quadro 28, o retorno do método `isOver` será impresso no console.

```
1  - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
2        NSLog(@"Barra vazia: %@"[lifeBar isOver];
3  }
```

Quadro 28 - Método `touchesEnded` adicionado para uso da funcionalidade `isOver`

Já o método `setUpdateRateNewValue` permite ao usuário mudar a taxa de atualização quando assim o desejar. Neste exemplo a taxa de atualização será mudada para 5. O Quadro 29 apresenta o método `touchesBegan` com as modificações para as novas funcionalidades já incluso.

```
1  - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
2        [lifeBar updateBar];
3        NSLog(@"Life Left: %d", [lifeBar getLifeLeft]);
4        if([lifeBar getLifeLeft] == 20){
5              [lifeBar setLifeLeftNewValue: 40];
6              [lifeBar setUpdateRateNewValue: 5];
7        }
8  }
```

Quadro 29 - Inclusão de novas funcionalidades na classe exemplo `ICExampleLifeBar`

A linha 6 do Quadro 29 muda a taxa de atualização para 5, ou seja, após a execução da linha 4 será decrementado 5 pontos de vida, sempre que houver um toque em tela.

3.3.2.3 Um exemplo simples usando o componente barra de tempo

O componente barra de tempo é muito parecido com o componente barra de vida, tendo assim várias das funcionalidades oferecidas pelo componente barra de vida. Esta seção apresentará como criar corretamente um componente barra de tempo. O usuário fará uso dos métodos públicos da classe `ICTimeBar`. O Quadro 30, apresenta a classe `ICExampleTimeBar` já com o componente barra de tempo implementado.

```
1  #import "ICExampleTimeBar.h"
2
3  @implementation ICExampleTimeBar
4        @synthesize barBackgroundImage;
5        @synthesize barTopImage;
6        @synthesize timeBar;
7
8  - (id)initWithFrame:(CGRect)frame {
9      if (self = [super initWithFrame:frame]) {
10        // Inicialize ICExampleTimeBar class
11        timeBar = [ICTimeBar alloc];
12
13        // load background image bar
14        barBackgroundImage = [timeBar loadImageBarBackground: 30.0:
15 70.0: 260.0: 20.0:
16 @"/Users/bell_cristina/Desktop/joystick/barBackgroundImage.png"];
17
18       // load top image bar
19       barTopImage = [timeBar loadImageBarTop:
20 @"/Users/bell_cristina/Desktop/joystick/barTopImage.png"];
21
22       [timeBar initialSettings: 30: 0.05];
23       [timeBar inicializeBar];
24     }
25     return self;
26 }
27
28 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
29       [timeBar setTimeLeftNewValue: 30];
30 }
32
32 - (void)drawRect:(CGRect)rect {
33     // Drawing code
34       [self addSubview:barBackgroundImage];
35       [self addSubview: barTopImage];
36 }
37 @end
```

Quadro 30 - Exemplo de implementação do componente barra de tempo

Na linha 14 e 19 do Quadro 30 as imagens do componente barra de tempo são carregadas, os parâmetros a serem informadas em ambas são os mesmos que precisam ser informados no componente barra de vida, descrito na seção 3.3.2.2.

Na linha 22 as configurações básicas deste componente são feitas, os parâmetros que devem ser informados são tempo máximo e intervalo de tempo. O parâmetro intervalo de tempo é dado em segundos e se refere a de quantos em quantos segundos a barra de tempo será atualizada. Neste exemplo o tempo máximo será 30 e o intervalo de tempo será de 0.05 segundos, ou seja, em 15 segundos esta barra de tempo estará vazia.

Por fim a linha 23 na qual é sinalizado para o componente começar a funcionar. Ao ser executada esta linha, a barra de tempo começa a decrementar até que chegue a 0. A Figura 33 mostra primeiramente a barra de tempo cheia, logo depois na metade do tempo e por fim a barra vazia.

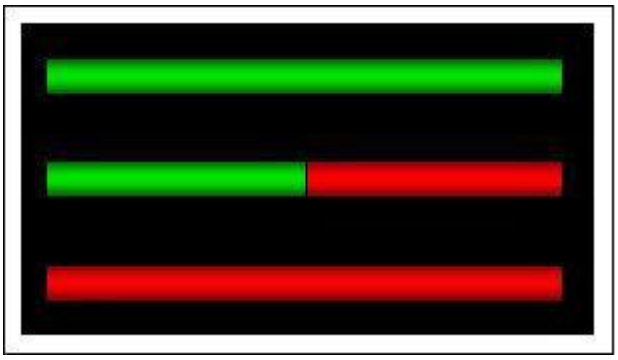


Figura 33 - Componente barra de tempo quando cheia, na metade e vazia

A barra de tempo ainda permite mudar o tempo restante da barra usando o método `setTimeLeftNewValue`, como demonstra a linha 29 do Quadro 30. Neste exemplo esta funcionalidade encontra-se dentro do método `touchesBegan` o que significa que sempre que houver um toque na tela o tempo restante será modificado para 30, pois este é o valor informado na linha 29 e a barra será atualizada para o tempo informado. Na Figura 34 é possível observar a barra de tempo perto da finalização e após o toque na tela a barra aparece atualizada.

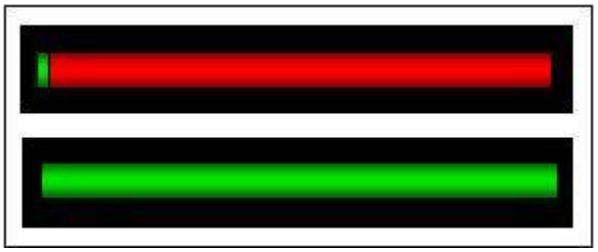


Figura 34 - Componente barra de tempo antes e depois do método `setTimeLeftNewValue`

O Quadro 31, mostra o método `touchesBegan` com novas funcionalidades, sendo que a primeira se encontra na linha 3, onde será apresentado no painel de controle o tempo restante, toda vez que um toque na tela aconteça, o tempo restante é dado pelo método `getTimeLeft`. Já na linha 4 será apresentado no painel de controle se o tempo está finalizado ou não. O método `isOver` é quem retornará esta resposta.

```
1 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
2       [timeBar setTimeLeftNewValue: 30];
3       NSLog(@"Tempo restante: %d",[timeBar getTimeLeft]);
4       NSLog(@"Tempo finalizado: %@", [timeBar isOver]);
5 }
```

Quadro 31 - Inclusão de novas funcionalidades no método `touchesBegan`

A Figura 35, mostra na primeira parte a barra de tempo atualizada com o novo valor de tempo, já na segunda parte esta figura mostra a saída das linhas 3 e 4.

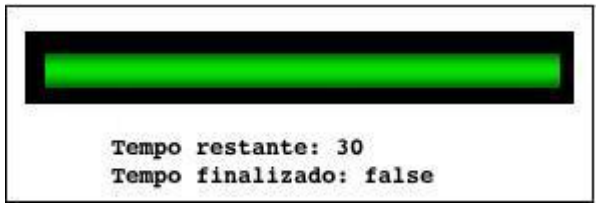


Figura 35 - Componente barra de tempo cheia e impressão no console do estado do mesmo

3.3.2.4 Um exemplo simples usando o componente pedal de aceleração

Nesta seção será mostrado como incluir corretamente um componente pedal de aceleração, e como configurar suas funcionalidades básicas. O usuário somente tem acesso aos métodos públicos da classe `ICGasPedal` e eles fornecem tudo o que é preciso para implementar o componente pedal de aceleração em um projeto. O Quadro 32, apresenta a classe `ICExamplePedal` já com o componente pedal de aceleração implementado.

```
1  #import "ICExamplePedal.h"
2
3  @implementation ICExamplePedal
4        @synthesize gasPedalImage;
5        @synthesize timer;
6        @synthesize gasPedal;
7
8  - (id)initWithFrame:(CGRect)frame {
9      if (self = [super initWithFrame:frame]) {
10        // Inicialize ICExamplePedal class
11        gasPedal = [ICGasPedal alloc];
12
13        // load pedal's image
14        gasPedalImage = [gasPedal loadImageGasPedal: 80.0: 50.0: 50.0:
15 50.0: @"/Users/bell_cristina/Desktop/joystick/ ballOutImage.png"];
16
17       [gasPedal initialSettings: 10: 0.05: NO];
18     }
19     return self;
20 }
21
22 - (void)drawRect:(CGRect)rect {
23     // Drawing code
24       [self addSubview: gasPedalImage];
25 }
26 @end
```

Quadro 32 - Exemplo de implementação do componente pedal de aceleração

No exemplo do Quadro 32, na linha 14 é criado a imagem que representa o pedal de aceleração, informando os parâmetros referentes a posição em x, y, largura, altura e o caminho completo da imagem. A linha 17 utiliza o método `initialSettings` que é responsável pelas configurações iniciais, sendo este já explanado na seção 3.3.1.9. Para este exemplo inicialmente usaremos o terceiro parâmetro que representa a desaceleração como

desligado, por este motivo o terceiro valor da linha 17 é `NO`. Com estas duas linhas acima explicadas já temos um pedal de aceleração criado, mas o mesmo ainda não está funcional.

Neste exemplo optou-se por adicionar o método `touchesBegan` na classe `ICExamplePedal`, no intuito de que sempre que ocorra um toque no pedal de aceleração este comece a acelerar e só pare quando o pedal de aceleração perder o toque, usando para isto o método `touchesEnded`. O Quadro 33 mostra o código referente a estes dois métodos.

```
1  -(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
2        UITouch *touch = [[event allTouches] anyObject];
3
4        gasPedalImage.userInteractionEnabled = YES;
5        if([touch view] == gasPedalImage){
6              [gasPedal touchGasPedal];
7        }
8  }
9
10 - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
11       UITouch *touch = [[event allTouches] anyObject];
12       if([touch view] == gasPedalImage){
13             [gasPedal touchEndedGasPedal];
14       }
15 }
```

Quadro 33 - Implementação dos métodos `touchesBegan` e `touchesEnded`

A linha 4 do Quadro 33 habilita o pedal de aceleração para interação de toques. Já a linha 6 aciona o método `touchGasPedal` da classe `ICGasPedal`, garantindo assim a inicialização da aceleração sempre que o componente receber um toque.

Neste exemplo a linha 13 do Quadro 32 será executada sempre que o componente deixar de ser pressionado, esta linha encarrega-se de que a desaceleração seja acionada ou não, dependendo do valor informado na linha 17 do Quadro 31.

No estado atual da implementação deste exemplo, o componente pedal de aceleração já é capaz de acionar a aceleração e desaceleração se esta foi escolhida, mas a classe `ICExamplePedal` ainda não consegue saber qual o valor da aceleração. Para que a classe exemplo possa ter acesso ao valor da aceleração o método `getAcceleration` deve ser usado.

O Quadro 34 demonstra o uso deste método, para isto optou-se por criar um método chamado `gameLoop` que será monitorado a cada 0.03 segundos e dentro deste método estará o método `getAcceleration`.

```
1  -(void)gameLoop {
2        NSLog(@"Aceleração: %d", [gasPedal getAcceleration]);
3  }
```

Quadro 34 - Implementação de uma nova funcionalidade

Na linha 2 do Quadro 34 a aceleração é apresentada no *console* para neste exemplo mostrar como saber o valor da aceleração. Como neste exemplo optou-se por monitorar o

método *gameLoop* então deve-se adicionar um *timer* que será responsável por este monitoramento. Este *timer* na lógica deste exemplo deverá ser acionado sempre que uma interação de toque ocorra com o componente e o *timer* deverá ser finalizado sempre que a interação seja finalizada. Para isto os métodos `touchesBegan` e `gameLoop` serão modificados como pode ser visto no Quadro 35.

```
1  -(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
2        UITouch *touch = [[event allTouches] anyObject];
3
4        gasPedalImage.userInteractionEnabled = YES;
5        if([touch view] == gasPedalImage){
6              [gasPedal touchGasPedal];
7
8              // listen the gameLoop method every 0.03 seconds
9              timer = [NSTimer scheduledTimerWithTimeInterval: 0.03
10 target:self selector:@selector(gameLoop) userInfo:nil repeats:YES];
11
12       }
13 }
14
15 -(void)gameLoop {
16       NSLog(@"Aceleração: %d", [gasPedal getAcceleration]);
17       if([gasPedal getAcceleration] == 0){
18             [timer invalidate];
19       }
   }
```

Quadro 35 - Implementação do *timer* de monitoramento do método `gameLoop`

O *timer* responsável pelo monitoramento do método `gameLoop` foi adicionado ao método `touchesBegan`, como mostra a linha 9 do Quadro 35. Já no método `gameLoop` as linhas 16, 17 e 18 foram adicionados, a linha 17 verifica se a aceleração for 0 então finaliza o monitoramento deste método. A Figura 36, mostra a saída escrita no *console* quando o pedal de aceleração é pressionado.

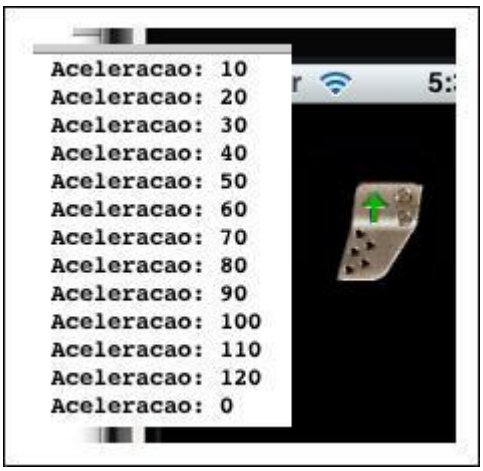


Figura 36 - Componente pedal de aceleração e a saída em tela da aceleração

Alterando o terceiro parâmetro da linha 17 do Quadro 32 para `YES`, irá ativar a desaceleração, e a saída escrita no painel de controle será conforme Figura 37.

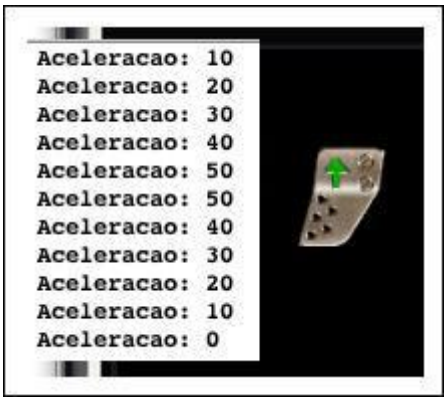


Figura 37 - Componente pedal de aceleração com a desaceleração ativada

3.3.2.5 Exemplo de um jogo usando os componentes *joystick* e barra de tempo

Nesta seção será mostrado os componentes *joystick* e barra de tempo inclusos em um jogo de labirinto.

O objetivo deste jogo é chegar ao final do labirinto, antes que o tempo se esgote, portanto o usuário utilizará o componente *joystick* para controlar o jogador, já o componente barra de tempo oferecerá uma resposta visual ao usuário indicando o tempo restante para a finalização da partida.

Este jogo é composto pelas classes `ICMap`, `ICPlayer` e `ICGameLabyrinth` sendo que a classe `ICMap` é responsável pelas ações relacionadas ao mapa deste jogo, como por exemplo, é ela que vai verificar se a próxima coordenada informada é válida ou se o jogador encontrou alguma barreira.

Já a classe `ICPlayer` terá como responsabilidade lidar com as ações referentes ao jogador como, por exemplo, atualizar a coordenada do jogador. Por fim a classe `ICGameLabyrinth`, que conterá a lógica do jogo Labirinto, é responsável principalmente por manipular os componentes desenvolvidos neste trabalho. Desta forma, apenas a classe `ICGameLabyrinth` será explanada aqui, pois a omissão das demais classes não prejudica o entendimento da utilização do jogo Labirinto para testar os componentes desenvolvidos.

Nas linhas 33 e 34 do Quadro 36, são instanciadas as classes que cuidaram do mapa e do jogador respectivamente.

```
1  #import " ICGameLabyrinth.h"
2
3  @implementation ICGameLabyrinth
4
5  // map and player
6  @synthesize player;
7  @synthesize playerImage;
8  @synthesize map;
9  @synthesize backgroundImageMap;
10
11 //joystick
12 @synthesize joystickCompound;
13 @synthesize backgroundImageJoystick;
14 @synthesize topImageJoystick;
15
16 // timeBar
17 @synthesize barBackgroundImage;
18 @synthesize barTopImage;
19 @synthesize timeBar;
20
21 // others
22 @synthesize youLoseImage;
23 @synthesize lastTouch;
24 @synthesize lastEvent;
25 @synthesize move;
26 @synthesize aux;
27 @synthesize timer;
28
29 - (id)initWithFrame:(CGRect)frame {
30     if (self = [super initWithFrame:frame]) {
32
32       // ======== Map and Player =========
33       player = [ICPlayer alloc] ;
34       map = [ICMap alloc] ;
35
36       // Load Background Map image
37       backgroundImageMap = [map loadImage: 0.0: 0.0: 320.0: 480.0:
38 @"/Users/bell_cristina/Desktop/Joystick/map01.png"];
39
40       // Load player image
41       playerImage = [player createPlayer: 16.0: 423.0: 25.0: 25.0:
42 @"/Users/bell_cristina/Desktop/Joystick/player01.png"];
43
44       [map setBackgroundImageRef: [map
45 getReferenceDataToImage:@"mapBlackWhite01.png"]];
46
47       // ========= Component Joystick =========
48       joystickCompound = [ICJoystickCompound alloc];
49
50       // load joystick background image
51       backgroundImageJoystick = [joystickCompound
52 loadImageJoystickBackground: 5 :400 :80 :80
53 :@"/Users/bell_cristina/Desktop/Labirinto/ballOut3.png"];
54
55       // load joystick top image
56       topImageJoystick = [joystickCompound loadImageJoystickTop:40 :40
57 :@"/Users/bell_cristina/Desktop/Labirinto/ballIn3.png"];
58
59       [joystickCompound initialSettings];
```

Quadro 36 – Primeiras configurações do componente *joystick* no jogo do Labirinto

Na linha 37 é carregada a imagem que representará o mapa deste jogo, são informados o x, y, largura, altura e caminho da imagem. Esta imagem referente ao mapa preenchera toda a tela do aparelho iPhone, e será esta que o usuário verá, quando estiver jogando. A linha 41 carrega a imagem do jogador e a linha 44 carrega a imagem do mapa que as classes utilizaram, ou seja, o mapa utilizado para verificações é um mapa em preto e branco (binário), onde o preto simboliza uma barreira.

A linha 48, instancia a classe `ICJoystickCompound`, responsável pelo componente *joystick* e as linhas 51 e 56 carregam as imagens que iram formar o componente *joystick*. Já a linha 59 certifica-se que o componente *joystick* terá as configurações básicas realizadas antes do uso deste componente.

A linha 2 do Quadro 37, que é uma continuação do quadro anterior é encarregada de instanciar o componente barra de tempo. Na linha 5 e 10 são responsáveis por carregar as imagens que formaram a barra de tempo.

Já a linha 13 o primeiro parâmetro define o tempo máximo (neste caso 60) e o segundo parâmetro é o intervalo de tempo em segundos da atualização. A próxima linha inicializa a barra de tempo e a linha 17 cria um *timer* que será responsável por monitorar o método chamado `gameLoop`.

```
1  // ========= Component Time Bar =========
2        timeBar = [ICTimeBar alloc];
3
4        // load background image bar
5        barBackgroundImage = [timeBar loadImageBarBackground: 60.0: 0.0:
6  260.0: 17.0:
7  @"/Users/bell_cristina/Desktop/Joystick/barBackgroundImage.png"];
8
9        // load top image bar
10       barTopImage = [timeBar loadImageBarTop:
11 @"/Users/bell_cristina/Desktop/Joystick/barTopImage.png"];
12
13       [timeBar initialSettings: 60: 1];
14       [timeBar inicializeBar];
15
16       // ========= Others ===========
17       timer = [NSTimer scheduledTimerWithTimeInterval:0.8 target:self
18 selector:@selector(gameLoop) userInfo:nil repeats:YES];
19
20       [[UIApplication sharedApplication] setStatusBarHidden:YES
21 animated:NO]; //hide iPhone status bar
22
23       [self refresh];
24
25     }
26     return self;
27 }
28
```

Quadro 37 - Primeiras configurações do componente barra de tempo no jogo do Labirinto

No Quadro 38 é criado um novo método que será acionado sempre que ocorrer um movimento causado por toques na tela. A linha 3, recebe a localização do último toque ocorrido na tela, e nas linhas 6, 7, 8 e 9 são capturadas a posição atual em tela do jogador.

A linha 13 habilita o componente *joystick* para interação com o usuário, já a linha 17 assegura-se que determinada lógica só seja executada caso o componente *joystick* receba uma interação. A linha 23 busca a aceleração do componente *joystick*, armazenando este valor em um atributo para futuro uso, e a linha 27 armazena o valor do penúltimo toque ocorrido na tela.

```
1  (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
2        UITouch *touch = [[event allTouches] anyObject];
3        CGPoint location = [touch locationInView:self];
4
5        //Get the x, y, width and height of the player
6        float x = [player x];
7        float y = [player y];
8        float width = [player width];
9        float height = [player height];
10
11
12       // Enable User Interaction for top image of the joystick
13       topImageJoystick.userInteractionEnabled = YES;
14
15       //Check if user touched the top image of the joystick
16 (topImageJoystick)
17       if([touch view] == topImageJoystick){
18             [self setMove: TRUE];
19             [self setAux: TRUE];
20             [self setLastTouch: touch];
21             [self setLastEvent: event];
22
23             int acceleration = [joystickCompound getAcceleration:
24 touch];
25             NSLog(@"ac %d", acceleration);
26
27             CGPoint previosLocation = [touch previousLocationInView:
28 self];
29             // Continue…
```

Quadro 38 – Método onde ocorre a interação com o componente *joystick*

A linha 2 do Quadro 39, verifica se a última posição do componente *joystick* é maior que a penúltima posição do mesmo, utilizando os valores encontrados no Quadro 38. Se a condição da linha 2 for verdadeira isto significa que a parte superior do componente *joystick* está se movendo para a direita, e na implementação deste jogo isto quer dizer que o jogador também deverá mover-se para a direita.

Como o jogador deseja mover-se para a direita as linhas 4 e 5 encontram o x, y do jogador referente ao canto superior direito e o canto inferior direito, e as linha 7 e 8 adicionam mais 1 referente ao que seria a próxima posição do jogador.

```
1  //walk rigth  +x
2  if(location.x > previosLocation.x){
3
4        CGPoint pointTR = [player getPlayerTopRightPoint];
5        CGPoint pointBR = [player getPlayerBottonRightPoint];
6
7        CGPoint nextStepTR = CGPointMake(pointTR.x + 1, pointTR.y);
8        CGPoint nextStepBR = CGPointMake(pointBR.x + 1, pointBR.y);
9
10       if(([self checkNextMove:nextStepTR] == false) || ([self
11 checkNextMove:nextStepBR] == false) ){
12             //collide than player come back to begin
13             [player updatePlayerCoordinate: [player xBegin] : [player
14 yBegin]];
15       }else {
16             [player updatePlayerCoordinate: (x + 1): pointTR.y];
17             }
18 }
19
20 //walk left -x
21 if(location.x < previosLocation.x){
22
23       CGPoint TL = CGPointMake(x, y);
24       CGPoint BL = CGPointMake(x, y + height);
25
26       CGPoint nextStepTL = CGPointMake(TL.x - 1, TL.y);
27       CGPoint nextStepBL = CGPointMake(BL.x - 1, BL.y);
28
29       if(([self checkNextMove:nextStepTL] == false) || ([self
30 checkNextMove:nextStepBL] == false)){
32             //collide than player come back to begin
32             [player updatePlayerCoordinate: [player xBegin] : [player
33 yBegin]];
34       }else{
35             [player updatePlayerCoordinate: nextStepTL.x :
36 nextStepTL.y];
37             }
38 }
```

Quadro 39 – Lê o movimento do *joystick* e direciona o jogador para a direita ou esquerda

Entre as linhas 10 e 17 são verificados se a próxima posição do jogador é válida, se for o jogador é atualizado, como pode ser observado na linha 17. Mas se a próxima posição não for válida significa que o jogador colidiu com um obstáculo sendo o mesmo retornado ao ponto de inicial.

A mesma lógica será usada para mover o jogador para a esquerda, somente a verificação referente a posição atual e a penúltima posição é que serão diferentes como mostra a linha 21.

No Quadro 40 o jogador será direcionado ou para cima ou para baixo, a lógica continuará sendo a mesma, mas novamente a verificação referente a posição atual e a penúltima posição serão diferentes. Na linha 2, encontra-se a verificação que diz que o componente está se movimentando para baixo e na linha 22 a verificação mostra que o componente está movendo-se para cima.

```
1  //walk botton +y
2  if(location.y > previosLocation.y){
3
4        CGPoint BL = CGPointMake(x, y + height);
5        CGPoint BR = CGPointMake(x + width, y + height);
6
7        CGPoint TL = CGPointMake(x, y);
8
9        CGPoint nextStepBL = CGPointMake(BL.x, BL.y + 1);
10       CGPoint nextStepBR = CGPointMake(BR.x, BR.y + 1);
11
12       if(([self checkNextMove:nextStepBL]  == false) || ([self
13 checkNextMove:nextStepBR] == false)){
14             //collide than player come back to begin
15             [player updatePlayerCoordinate: 16 : 423];
16       }else {
17             [player updatePlayerCoordinate: nextStepBL.x : TL.y + 1];
18             }
19 }
20
21 //walk up -y
22 if(location.y < previosLocation.y){
23
24       CGPoint TL = CGPointMake(x, y);
25       CGPoint TR = CGPointMake(x + width, y);
26
27       CGPoit nextStepTL = CGPointMake(TL.x, TL.y - 1);
28       CGPoint nextStepTR = CGPointMake(TR.x, TR.y - 1);
29
30       if(([self checkNextMove:nextStepTL] == false) || ([self
32 checkNextMove:nextStepTR] == false)){
32             //collide than player come back to begin
33             [player updatePlayerCoordinate: 16 : 423];
34       }else{
35             [player updatePlayerCoordinate: nextStepTL.x:
36 nextStepTL.y];
37             }
38 }
39 } // End if([touch view] == [self topImageJoystick])
```

Quadro 40 - Lê o movimento do *joystick* e direciona o jogador para a cima ou para baixo

O método `touchesEnded` é criado no Quadro 41 na linha 1, este método é acionado quando qualquer toque em tela for liberado, por isto na linha 4 é verificado se o toque liberado foi o do componente *joystick*, caso seja o método `touchesEndedJoystick` do componente *joystick* será acionado, garantindo assim que a parte superior deste componente volte a posição inicial.

Já o método `gameLoop` encarrega-se da lógica que precisa ser monitorada freqüentemente. É este método que o *timer* criado no Quadro 37 na linha 17 monitora a cada 0,08 segundos, valor este também definido na mesma linha. Na linha 12 do Quadro 41 é feita uma chamada ao método `isGameOver` que é definido na linha 20. O que este método faz é chamar o método isOver do componente barra de tempo, o qual retornará se o tempo se esgotou. Caso o tempo tenha se esgotado, isto significa que o usuário perdeu, assim sendo, na

linha 22 o método `isGameOver` irá finalizar o timer que estava fazendo o monitoramento do método `gameLoop`, pois o jogo está sendo encerrado e na linha 23 é carregada a imagem que informa ao usuário que ele perdeu.

Na linha 29, encontra-se o método `checkNextMove` que é acionado sempre que é preciso verificar se a próxima coordenada não é um obstáculo e a linha 30 chama um método da classe `ICMap` que se encarrega de validar a coordenada informada. O método `checkNextMove` é acionado nas linhas 10 e 29 do Quadro 39 e nas linhas 12 e 30 do Quadro 40, onde ocorrem as verificações para mover o jogador.

```
1  - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
2        UITouch *touch = [[event allTouches] anyObject];
3
4        if([touch view] == topImageJoystick){
5              [joystickCompound touchesEndedJoystick: touch];
6              [self setMove: FALSE];
7              [self setAux: FALSE];
8        }
9  }
10
11 -(void)gameLoop {
12       [self isGameOver];
13
14       if([self move] == false && [self aux] == true){
15             [self touchesMoved:[self lastTouch] withEvent:[self
16 lastEvent]];
17       }
18 }
19
20 -(void)isGameOver{
21       if([timeBar isOver] == true){
22             [timer invalidate];
23             youLoseImage =  [map loadImage: 0.0: 0.0: 320.0: 480.0:
24 @"youLoseImage.png"];
25             [self addSubview: youLoseImage];
26       }
27 }
28
29 -(BOOL)checkNextMove:(CGPoint) newLocation {
30       return [map isValidNextCoordinate: newLocation];
32 }
```

Quadro 41 - Implementação dos métodos `touchesEnded` e `gameLoop`

Por fim no Quadro 42 na entre as linhas 1 e 9 são informadas todas as imagens que se deseja mostrar na tela, e entre as linhas 12 e 21 é liberado toda a memória alocada.

```
1  - (void)drawRect:(CGRect)rect {
2      // Drawing code
3        [self addSubview: backgroundImageMap];
4        [self addSubview: playerImage];
5        [self addSubview: backgroundImageJoystick];
6        [self addSubview: topImageJoystick];
7        [self addSubview: barBackgroundImage];
8        [self addSubview: barTopImage];
9  }
10
11
12 - (void)dealloc {
13       [backgroundImageMap release];
14       [playerImage release];
15       [backgroundImageJoystick release];
16       [topImageJoystick release];
17       [player release];
18       [map release];
19       [joystickCompound release];
20     [super dealloc];
21 }
```

Quadro 42 – Métodos utilitários

A Figura 38 apresenta a tela do jogo Labirinto em andamento, na parte superior encontra-se o componente barra de tempo e na parte inferior esquerda encontra-se o componente *joystick*, com a parte superior sendo movida em direção a direita. Na parte inferior direita encontra-se um quadrado roxo representando o jogador.

Figura 38 – Jogo Labirinto usando os componentes *joystick* e barra de tempo

3.4 RESULTADOS E DISCUSSÃO

Os testes da biblioteca de interface desenvolvida foram realizados em quatro fases, sendo a primeira fase relacionada ao desenvolvimento dos componentes de interface, a segunda fase para testes práticos de implementação, a terceira fase na qual foram realizados testes de usabilidade dos componentes de interface e por fim a quarta fase onde forem realizados testes de desempenho e uso de memória para os componentes.

Todas as fases de testes foram desenvolvidas e testadas utilizando-se o simulador disponibilizado pela Apple para a criação de aplicações para o iPhone. Para que este projeto pudesse ser testado em um iPhone real, além do aparelho ainda seria preciso inscrever-se no

programa de desenvolvedores da Apple para iPhone por uma taxa de 99 dólares, para que o aparelho fosse liberado para testes de aplicações. Assim sendo, optou-se por permanecer usando somente o simulador disponibilizado.

Na primeira fase foram desenvolvidos os componentes *joystick*, barra de vida, barra de tempo e pedal de aceleração. Todos os componentes foram desenvolvidos usando a técnica de orientação a objetos e reaproveitamento de código. Devido a algumas limitações da linguagem usada, algumas técnicas de orientação a objetos são só conceituadas neste trabalho, como por exemplo, não existe em Objective-C uma forma de assegurar que uma classe seja abstrata, embora conceitualmente possa ser criada uma classe abstrata. Optou-se por utilizar as técnicas de orientação a objetos, mesmo aquelas que só podem ser usadas de forma conceitual, pois a orientação a objetos facilita a leitura e a reutilização de código e permite que no futuro este trabalho possa ser reescrito em outra linguagem de programação sem maiores complicações.

Na segunda fase foram desenvolvidos testes práticos, nos quais os recursos disponibilizados por este trabalho foram usados. Estes testes foram descritos nas seções 3.3.2.1, 3.3.2.2, 3.3.2.3 e 3.3.2.4, onde são usados como exemplos de implementações, pois fazem uso de todas as funcionalidades deste trabalho.

Na terceira fase foram realizados testes de usabilidade do componente *joystick*, para estes testes foi desenvolvido um jogo, o Labirinto, com 3 fases, que foram disponibilizados a 15 usuários, sendo que a fase 1 foi considerada fácil, a fase 2 como média e por fim a fase 3 como difícil. Foi solicitado a cada usuário que jogasse todas as 3 fases do jogo 2 vezes, mas com os parâmetros de aceleração mudados a cada vez, permitindo assim ao jogo obter automaticamente dados referentes a quantidade de colisões ocorridas entre o jogador e as paredes do labirinto, o tempo total levado pelo usuário para terminar cada fase do teste.

O tempo total levado pelo usuário para terminar o teste é calculado usando a fórmula apresentada no Quadro 43, onde `TF` representa o tempo final, `TI` o tempo inicial, `C` representa a quantidade total de colisões ocorridas e por fim o `VP`, um valor em segundos que representa uma punição para cada colisão ocorrida, neste optou-se por utilizar o valor de 2 segundos como punição.

```
TG = TF - TI
TT = TG + C * VP
```

Quadro 43 – Fórmula matemática para o cálculo do tempo total do teste

O Quadro 44 apresenta os resultados obtidos na fase 1, com os parâmetros de aceleração entre 0 e 100, sendo que a maior quantidade de colisões individual foi 13, tempo

sem punição foi de 49 segundos e o maior tempo com punição foi de 50 segundos.

Usuário	Colisões	Tempo	Tempo Total
1	4	00:19	00:27
2	0	00:18	00:18
3	2	00:09	00:13
4	3	00:08	00:14
5	0	00:10	00:10
6	10	00:14	00:34
7	0	00:07	00:07
8	0	00:08	00:08
9	0	00:49	00:49
10	4	00:28	00:36
11	2	00:46	00:50
12	13	00:09	00:35
13	5	00:10	00:20
14	3	00:13	00:19
15	2	00:10	00:14

Quadro 44 – Fase 1 com aceleração entre 0 e 100

Com aceleração entre 0 e 50 (Quadro 45), observa-se que o menor tempo foi de 09 segundos sem nenhuma colisão, já o maior tempo foi de 47 segundos onde houve 8 colisões.

Usuário	Colisões	Tempo	Tempo Total
1	0	00:16	00:16
2	0	00:19	00:19
3	0	00:09	00:09
4	4	00:11	00:19
5	1	00:31	00:33
6	8	00:31	00:47
7	4	00:25	00:33
8	2	00:31	00:35
9	0	00:21	00:21
10	3	00:36	00:42
11	13	00:20	00:46
12	0	00:21	00:21
13	1	00:19	00:21
14	0	00:10	00:10
15	2	00:20	00:24

Quadro 45 – Fase 1 com aceleração entre 0 e 50

O Quadro 46 mostra o resultado dos testes da fase 2 com a aceleração entre o e 100, onde observa-se que a quantidade de colisões totais aumentou consideravelmente em relação ao teste da fase 1 (Quadro 44) com mesma aceleração. Nesta fase ocorreram 125 colisões e o maior tempo foi de 4 minutos e 25 segundos.

Usuário	Colisões	Tempo	Tempo Total
1	15	00:30	01:00
2	13	00:29	00:55
3	18	00:20	00:56
4	13	00:30	00:56
5	14	00:27	00:55
6	1	02:44	02:46
7	0	04:25	04:25
8	7	01:25	01:39
9	3	02:25	02:31
10	7	02:35	02:49
11	10	02:45	03:05
12	5	01:45	01:55
13	3	02:00	02:06
14	6	01:05	01:17
15	10	00:38	00:58

Quadro 46 – Fase 2 com aceleração entre 0 e 100

No Quadro 47, apresenta os resultados da fase 2 com aceleração entre 0 e 50, sendo que o menor tempo neste teste foi de 43 segundos onde ocorreram 7 colisões.

Usuário	Colisões	Tempo	Tempo Total
1	12	00:46	01:10
2	7	01:32	01:46
3	1	01:00	01:02
4	1	01:21	01:23
5	13	00:26	00:52
6	15	00:23	00:53
7	16	00:22	00:54
8	7	00:29	00:43
9	10	00:28	00:48
10	6	00:32	00:44
11	0	05:08	05:08
12	3	01:00	01:06
13	5	00:59	01:09
14	9	00:26	00:44
15	12	00:20	00:44

Quadro 47 – Fase 2 com aceleração entre 0 e 50

No Quadro 48 representando a aceleração entre 0 e 100 da fase 3, foi onde obteve-se 260 colisões, sendo este a maior quantidade total de colisões entre todas as fases.

Usuário	Colisões	Tempo	Tempo Total
1	56	00:25	02:17
2	12	00:17	00:41
3	22	00:25	01:09
4	28	00:22	01:18
5	33	00:10	01:16
6	14	00:24	00:52
7	7	01:25	01:39
8	10	00:50	01:10
9	7	02:00	02:14
10	3	02:52	02:58
11	15	00:21	00:51
12	18	00:26	01:02
13	20	00:18	00:58
14	10	00:25	00:45
15	5	02:01	02:11

Quadro 48 - Fase 3 com aceleração entre 0 e 100

O Quadro 49, que mostra os resultados dos testes da fase 3, com aceleração entre 0 e 50, teve o maior tempo em 2 minutos e 37 segundos com um total de 3 colisões para este tempo.

Usuário	Colisões	Tempo	Tempo Total
1	3	02:31	02:37
2	64	00:08	02:16
3	34	00:08	01:16
4	28	00:07	01:03
5	10	00:08	00:28
6	15	00:08	00:38
7	13	00:12	00:38
8	7	00:43	00:57
9	4	01:27	01:35
10	11	00:08	00:30
11	7	00:55	01:09
12	4	02:05	02:13
13	8	01:58	02:14
14	3	02:00	02:06
15	12	00:08	00:32

Quadro 49 - Fase 3 com aceleração entre 0 e 50

Pode-se observar de maneira geral que em todas as fases com aceleração entre 0 e 50 foi onde houve as menores quantidades de colisões no total. Os tempos variaram bastante, mas em geral quanto maior o tempo menor foi o número de colisões para aquele usuário.

A Figura 39, mostra o gráfico referente ao grau de dificuldade do experimento para as fases 1, 2 e 3, onde foi calculado a média do tempo total de cada fase. Pode-se observar que a

média referente à fase 1, representado em azul na Figura 39 foi a fase onde os usuários levaram menos tempo. Já a fase 2, representado em vermelho teve dois picos (usuários 7 e 11) com valores de tempo elevados, mas em ambos os picos os usuários obtiveram um total de 0(zero) colisões, ou seja, ambos optaram por não causar nenhuma colisão, mas como conseqüência demoraram mais tempo para concluir a fase. Por fim a fase 3, representado em verde não teve nenhum valor referente ao tempo total muito elevado, mas nenhum usuário conseguiu terminar esta fase sem nenhuma colisão.

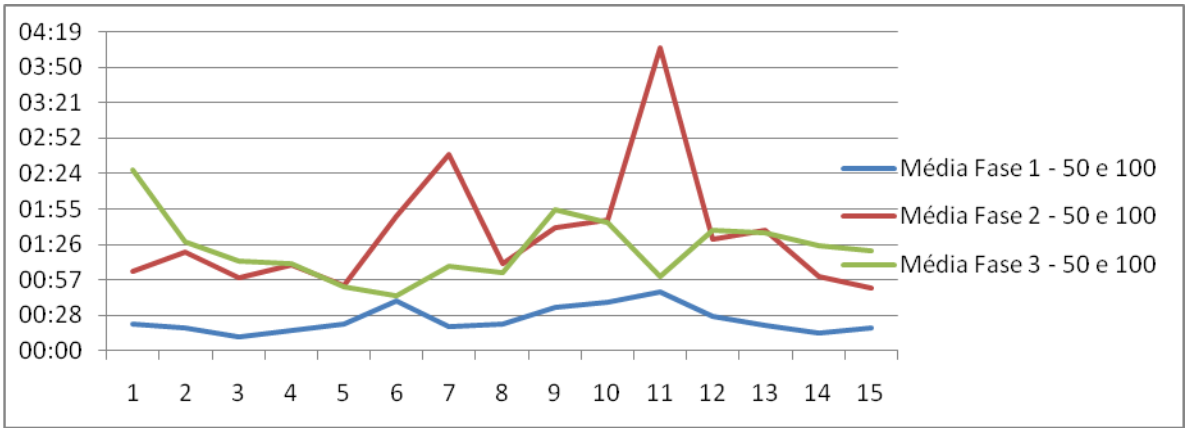


Figura 39 – Gráfico do grau de dificuldade do experimento para as fases 1, 2 e 3

Na Figura 40, é apresentado o gráfico referente à análise da utilização da aceleração entre 0..50 e 0..100. Para este gráfico foi calculado a média do tempo total de todas as fases para a aceleração de 0 até 50 e logo depois foi calculado a média para a aceleração de 0 até 100. O que permitiu verificar que na média para a aceleração entre 0 e 50 ocorreu um pico (usuário 11) com maior valor para o tempo total, mas no geral a aceleração entre 0 e 50 foi onde ocorrem os menores valores totais de tempo, assim como a menor quantidade total de colisões.

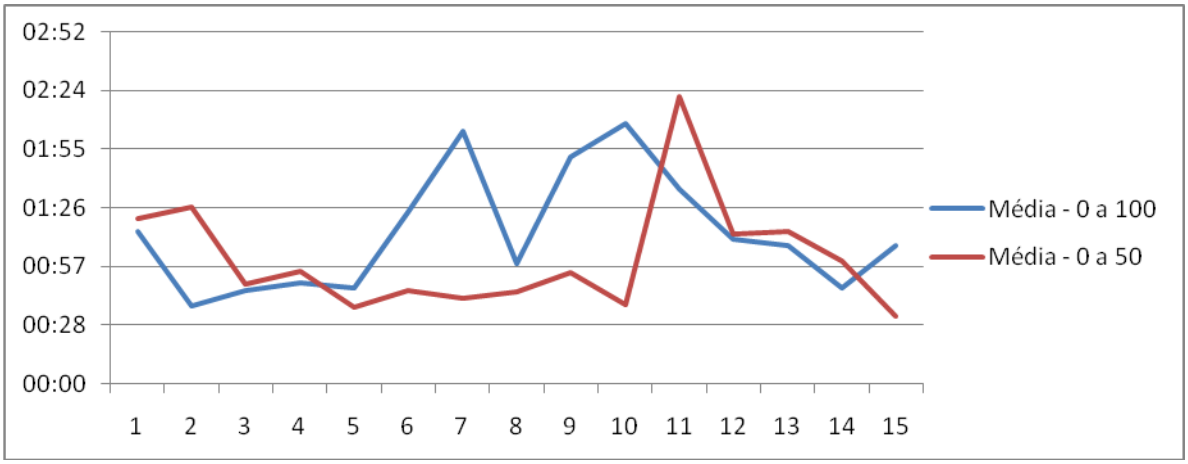


Figura 40 – Gráfico da análise da utilização da aceleração 0..50 e 0..100

Por fim a quarta fase foram realizados testes de desempenho e uso de memória para os

componentes, onde observou-se que o jogo Labirinto (Quadro 50) sem nenhum componente inserido utiliza um total de aproximadamente 9,81MB de memória, já o mesmo jogo com qualquer um dos componentes inseridos utiliza aproximadamente 9,82MB de memória, ou seja, um aumento de 1MB por componente.

Já com relação ao desempenho os testes mostraram que o componente *joystick* utiliza no máximo 5,8% da CPU, já o componente barra de vida é o que menos utiliza a CPU usando apenas 1%. O componente barra de tempo usa aproximadamente 1,4% da CPU e o componente pedal de aceleração usa no máximo 12% da CPU onde foi possível perceber que o componente pedal de aceleração tem o uso da CPU aumentado gradativamente conforme aumenta o tempo no qual o componente é usado sem interrupção, mas este parece estabilizar quando o valor chega em 12% de uso da CPU.

Componentes	**Joystick**	**Barra de Vida**	**Barra de Tempo**	**Pedal de aceleração**	**Labirinto**
Memória	9,82MB	9,82MB	9,82MB	9,82MB	9,81MB
CPU %	5,8%	1%	1,4%	12%	-

Quadro 50 – Valores do uso de memória e de CPU para os componentes e o jogo Labirinto

4 CONCLUSÕES

Este trabalho permite aos desenvolvedores maior agilidade na implementação de aplicações que utilizam os componentes do tipo disponibilizado por esta biblioteca de interface. Isto é possível porque o mesmo foi estruturado de forma a simplificar o trabalho do usuário-desenvolvedor, podendo assim suprir de forma fácil a necessidade dos usuários que utilizam componentes de interface do tipo desenvolvido por este trabalho.

O presente trabalho cumpriu todos os requisitos propostos e provou através dos testes de implementação realizados que os objetivos foram cumpridos e que esta biblioteca de interface de componentes para o iPhone OS é funcional.

Através dos testes de usabilidade para o componente *joystick* também foi possível perceber que quanto menor a aceleração do componente mais controle o usuário tem do jogador. Já os testes de memória mostraram que todos os componentes utilizam aproximadamente 1MB de memória e o teste de desempenho mostrou que o componente pedal de aceleração é o componente que mais usa a CPU.

Esta biblioteca tem como principal vantagem a facilidade de implementação dos componentes que a mesma disponibiliza, pois é necessário apenas a chamada de alguns métodos para que seja possível usufruir dos componentes prontos, que se fossem implementados pelo usuário-desenvolvedor demandariam tempo, aumento do nível de complexidade da aplicação e muitas vezes não permitiriam ao desenvolvedor reutilizar o código, tendo o mesmo que recomeçar o ciclo de desenvolvimento do componente quando uma nova aplicação fosse criada.

Este trabalho possui algumas limitações em certos componentes, uma delas é no componente *joystick*, pois este apenas consegue realizar corretamente os cálculos internos se a imagem utilizada for circular, do contrário os valores de aceleração podem não ser precisos, assim como a imagem superior do *joystick* poderá fugir do contorno da imagem de fundo deste componente.

4.1 EXTENSÕES

Sugere-se, para futuros trabalhos, a criação de outros componentes de interface, por

exemplo, um velocímetro e uma *label*. Também sugere-se a criação de novas funcionalidades para que seja possível oferecer suporte ao uso do acelerômetro nos componentes deste trabalho.

Recomenda-se também que os componentes apresentados neste trabalho, sejam reescritos para uso com a biblioteca de rotinas gráficas OpenGLES.

Também é sugerido a integração com o ambiente de desenvolvimento de telas Interface Builder, ou até mesmo a criação de um novo ambiente de desenvolvimento de telas, similar ao que a IDE NetBeans oferece. Na criação de um novo ambiente de desenvolvimento de telas este deve facilitar a construção gráfica dos componentes de interface, e seu funcionamento poderia ser feito utilizando *drag and drop* dos componentes. Desta forma tornaria a construção dos componentes completamente visuais, tirando do desenvolvedor a necessidade de se preocupar com a disposição dos componentes na tela através de código. A capacidade de informar as configurações básicas no ambiente de desenvolvimento também seria interessante.

Com relação aos testes é sugerido que seja realizada uma maior quantidade de testes, sob diversas situações e que os componentes possam ser testados em um aparelho iPhone.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE. **Acelerômetro:** feito para mudar. [S.l.], 2009a. Disponível em: <http://www.apple.com/br/iphone/features/accelerometer.html>. Acesso em: 19 mar. 2009.

_____. **App Store.** [S.l.], 2009b. Disponível em: <http://www.apple.com/iphone/appstore/>. Acesso em: 30 mar. 2009.

_____. **A tour of Xcode.** [S.l.], 2009c. Disponível em: <http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/A_Tour_of_Xcode/A_Tour_of_Xcode.pdf>. Acesso em: 07 nov. 2009.

_____. **Cocoa fundamentals guide.** [S.l.], 2009d. Disponível em: <http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf>. Acesso em: 04 nov. 2009.

_____. **Dashcode user guide.** [S.l.], 2009e. Disponível em: <http://developer.apple.com/mac/library/documentation/AppleApplications/Conceptual/Dashcode_UserGuide/Contents/Resources/en.lproj/Dashcode_UserGuide.pdf>. Acesso em: 20 ago. 2009.

_____. **Framework programming guide.** [S.l.], 2006. Disponível em: <http://developer.apple.com/mac/library/documentation/MacOSX/Conceptual/BPFrameworks/BPFrameworks.pdf>. Acesso em: 03 nov. 2009.

_____. **GPS.** [S.l.], 2009f. Disponível em: <http://www.apple.com/br/iphone/features/gps.html>. Acesso em: 19 mar. 2009.

_____. **Interface Builder.** [S.l.], 2008a. Disponível em: <http://developer.apple.com/tools/interfacebuilder.html>. Acesso em: 20 ago. 2009.

_____. **Interface Builder user guide.** [S.l.], 2008b. Disponível em: <http://developer.apple.com/iphone/library/documentation/DeveloperTools/Conceptual/IB_UserGuide/IB_UserGuide.pdf>. Acesso em 20 ago. 2009.

_____. **iPhone 3GS.** [S.l.], 2009g. Disponível em: <http://www.apple.com/iphone/compare-iphones>. Acesso em: 15 set. 2009.

_____. **iPhone.** [S.l.], 2009h. Disponível em: <http://www.apple.com/br/iphone/features>. Acesso em: 19 mar. 2009.

_____. **iPhone application programming guide.** [S.l.], 2009k. Disponível em: <http://developer.apple.com/iphone/library/DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf>. Acesso em: 03 nov. 2009.

_____. **iPhone development guide.** [S.l.], 2009i. Disponível em: <http://developer.apple.com/iphone/library/documentation/Xcode/Conceptual/iphone_development/iPhone_Development_Guide.pdf>. Acesso em: 03 nov. 2009.

_____. **iPhone:** How to. [S.l.], 2009j. Disponível em: <http://www.apple.com/br/iphone/how-to/#basics.typing>. Acesso em: 04 nov. 2009.

_____. **iPhone human interface guidelines.** [S.l.], 2009l. Disponível em: <http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html>. Acesso em: 19 mar. 2009.

_____. **Keyboard.** [S.l.], 2009m. Disponível em: <http://www.apple.com/iphone/features/keyboard.html>. Acesso em: 19 mar. 2009.

_____. **Leopard technologys for developers.** [S.l.]: [s.n.], 2007. Disponível em: <http://developer.apple.com/leopard/overview/dashcode.html>. Acesso em: 20 ago. 2009.

_____. **Mac Dev Center:** Xcode release notes. [S.l.], 2009n. Disponível em: <http://developer.apple.com/mac/library/releasenotes/DeveloperTools/RN-Xcode/index.html#//apple_ref/doc/uid/TP40001051-SW26>. Acesso em: 04 nov. 2009.

_____. **Multi-touch.** [S.l.], 2009o. Disponível em: <http://www.apple.com/br/iphone/features/multitouch.html>. Acesso em: 19 mar. 2009.

_____. **NSObject class reference.** [S.l.], 2009p. Disponível em: <http://developer.apple.com/mac/library/documentation/cocoa/reference/Foundation/Classes/NSObject_Class/NSObject_Class.pdf>. Acesso em: 03 nov. 2009.

_____. **OpenGL ES programming guide for iPhone OS.** [S.l.], 2009q. Disponível em: <http://developer.apple.com/iphone/library/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLES_ProgrammingGuide.pdf>. Acesso em: 29 set. 2009.

_____. **OpenGL programming guide for Mac OS X.** [S.l.], 2009r. Disponível em: <http://developer.apple.com/mac/library/documentation/GraphicsImaging/Conceptual/OpenGL-MacProgGuide/OpenGLProg_MacOSX.pdf>. Acesso em: 07 set. 2009.

_____. **Technical specifications.** [S.l.], 2009s. Disponível em: <http://www.apple.com/iphone/specs.html>. Acesso em: 19 mar. 2009.

_____. **The Objective-C programming language.** [S.l.], 2009t. Disponível em: <http://developer.apple.com/mac/library/DOCUMENTATION/Cocoa/Conceptual/ObjectiveC/ObjC.pdf>. Acesso em: 03 nov. 2009.

_____. **Tools:** Xcode. [S.l.], 2009u. Disponível em: <http://developer.apple.com/tools/xcode/>. Acesso em: 04 nov. 2009.

_____. **UIKit framework reference.** [S.l.], 2009v. Disponível em: <http://developer.apple.com/iphone/library/documentation/uikit/reference/uikit_framework/UIKit_Framework.pdf>. Acesso em: 04 nov. 2009.

ALVARENGA, Filipe. **BLOG.MACMAGAZINE.** [S.l.], 2008. Disponível em: <http://macmagazine.com.br/blog/2008/12/02/brothers-in-arms-hour-of-heroes-chega-a-iphone-app-store/>. Acesso em: 28 mar. 2009.

BUCHANAN, Levi. **IGN:** assassin's creed. [S.l], 2009a. Disponível em: <http://wireless.ign.com/articles/971/971971p1.html>. Acesso em: 04 ago. 2009.

_____. **IGN:** Prey invasion preview. [S.l], 2009b. Disponível em: <http://au.wireless.ign.com/articles/952/952649p1.html >. Acesso em: 29 mar. 2009.

CARMO, Izabel C. [S.l.], 2009. Disponível em: <http://www.inf.furb.br/~dalton/TCC/Izabel/Documentacao.zip>. Acesso em: 24 nov. 2009.

CYBIS, Walter; BETIOL, Adriana H.; FAUST, Richard. **Ergonomia e usabilidade:** conhecimentos, métodos e aplicações. [S.l.]: Novatec, 2007.

ELLIS, Nick. **AppStore blog.** [S.l.], 2008. Disponível em: <http://appstoreblog.com.br/2008/11/review-asphalt-4-elite-racing-um-jogo-de-corrida-radical-com-desconto-de-20/>. Acesso em: 04 ago. 2009.

_____. **AppStore blog.** [S.l.], 2009a. Disponível em: <http://appstoreblog.com.br/2009/04/assassins-creed-altairs-chronicles-da-gameloft/>. Acesso em: 04 ago. 2009.

_____. **AppStore blog.** [S.l.], 2009b. Disponível em: <http://appstoreblog.com.br/2009/07/minigore-um-shooter-cartoon-com-excelentes-graficos/>. Acesso em: 04 ago. 2009.

_____. **AppStore blog.** [S.l.], 2009c. Disponível em: <http://appstoreblog.com.br/2009/08/pac-man-remix-so-mesmo-para-os-fas-do-jogo-classico/>. Acesso em: 04 ago. 2009.

FAYAD, Mohamed E.; JOHNSON, Ralph E.; SCHMIDT, Douglas C. **Building application frameworks:** object-oriented foundations of framework design. [S.l.]: John Wiley & Sons, 1999.

FERREIRA, Aurélio B. **Novo dicionário da língua portuguesa.** 2. ed. Rio de Janeiro: Nova Fronteira, 1986.

FIEMG. **m-Leaning**: a educação com mobilidade. [S.l.], 2007. Disponível em: <http://www.fiemg.com.br/ead/site-ead/artigos.htm>. Acesso em: 28 mar. 2009.

GAMELOFT. **Gameloft:** jogos para celular. [S.l.], 2008a. Disponível em: <http://br.gameloft.com/jogos-iphone/brothers-in-arms-hour-of-heroes/>. Acesso em: 28 mar. 2009.

_____. **Gameloft:** jogos para celular. [S.l.], 2008b. Disponível em: <http://br.gameloft.com/jogos-iphone/asphalt-4-elite-racing/>. Acesso em: 04 mar. 2009.

GENTIL, Breno. **Estudo de usabilidade de ambientes virtuais tridimensionais através do Second Life.** 2008. 73 f. Dissertação (Mestrado em Design) – Curso de Pós-graduação em Design, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

GNU Project. **GNU project:** Free Software Foundation. [S.l.], 2007. Disponível em: <http://gcc.gnu.org/gccmission.html>. Acesso em: 04 nov. 2009.

GSMFANS. **GSMFans.** [S.l], 2007. Disponível em: <http://www.gsmfans.com.br/index.php?topic=556>. Acesso em: 06 nov. 2009.

HEESCH, Dimitri. **Doxygen**. [S.l.], 2009. Disponível em: <http://www.stack.nl/~dimitri/doxygen/>. Acesso em: 06 nov. 2009.

HEWETT, Thomas T. et al. Curricula for human-computer interaction. In: ACM SPECIAL INTEREST GROUP ON COMPUTER-HUMAN INTERACTION CURRICULUM DEVELOPMENT GROUP, 1992, [New York]. **Proceedings...** [New York]: [s.n.], 1992. p. 1-161. Disponível em: <http://sigchi.org/cdg/index.html>. Acesso em: 13 mar. 2009.

HODAPP, Eli. **Touch Arcade.** [S.l.], 2009. Disponível em: <http://toucharcade.com/2009/07/30/minigore-a-furry-survival-shooter/>. Acesso em: 04 ago. 2009.

KICHALOWSKY, Marco A. **Yahoo! Brasil:** notícias. [S.l.], 2009. Disponível em: <http://br.noticias.yahoo.com/s/11022009/7/tecnologia-negocios-prey-jogo-tiro-pessoa.html>. Acesso em: 29 mar. 2009.

MANSSOUR, Isabel H. **Introdução à OpenGL.** [S.l.], 2003. Disponível em: <http://www.inf.pucrs.br/~manssour/OpenGL/Introducao.html>. Acesso em: 07 set. 2009.

MORAN, Thomas. The command language grammars: a representation for the user interface of interactive computer systems. **International Journal of Man-Machine Studies**, [S.l.], n. 15, p. 3-50, 1981.

OPEN SOURCE INITIATIVE. **Open source.** [S.l.], 2006. Disponível em: <http://www.opensource.org/docs/definition.php>. Acesso em: 23 mar. 2009.

ORTH, Afonso I. **Interface homem-máquina.** Porto Alegre: AIO, 2005.

PALLEY, Steve. **Slide to play.** [S.l.], 2009. Disponível em: <http://www.slidetoplay.com/story/pac-man-remix-review>. Acesso em: 04 ago. 2009.

ROCHA, Heloísa V.; BARANAUSKAS, Maria C. C. **Design e avaliação de interfaces humano-computador**. Campinas: NIED/UNICAMP, 2003.

TECHBOOK. **Techbook**. [S.l], 2009. Disponível em: <http://sandeep.weblogs.us/archives/apple-iphone-for-att-subscribers-still-no-tethering.html>. Acesso em: 06 nov. 2009.

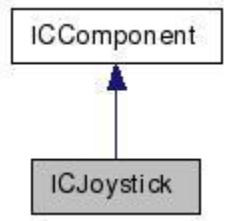
U.S. DEPARTMENT OF DEFENSE. **Global positioning system policy announced**. [S.l.], 1996. Disponível em: <http://www.defenselink.mil/releases/release.aspx?releaseid=866>. Acesso em: 20 mar. 2009.

ANEXO A – Documentação parcial da biblioteca de componentes de interface para o iPhone OS

Segue abaixo a documentação desta biblioteca gerada a partir da documentação no código fonte. Devido a esta documentação ser extensa, optou-se por colocar neste anexo somente a documentação referente ao componente *joystick*. A formatação da documentação foi mantida a mesma do arquivo completo.

ICJoystick Class Reference

`#import <ICJoystick.h>`
Inherits **ICComponent**.Inherited by **ICJoystickCompound**.Collaboration diagram for ICJoystick:



Public Member Functions

- (UIImageView *) - **loadImage:::::**

Detailed Description

IMPORTANT: This class should be considered abstract. This class has the commons methods for the joystick's components.

Member Function Documentation

- (UIImageView *) loadImage: (float) *x* : (float) *y* : (float) *width* : (float) *height* : (NSString *) *imagePath*

Responsable for loading an image.

Parameters:

x : the x position where the image should be placed.
y : the y position where the image shoud be placed.
width : the desired width for the given image. The image it's resized.
height : the desired height for the given image. The image it's resized.
imagePath : the full path of image.

Returns:

UIImageView* : the ready image.

Reimplemented from **ICComponent** (*p.**Erro! Indicador não definido.***).

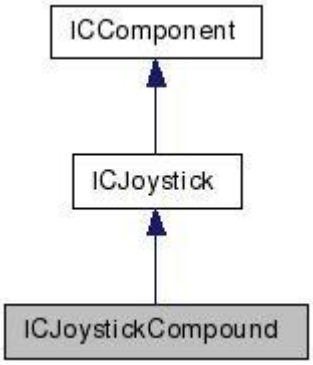
The documentation for this class was generated from the following files:

- Desktop/Joystick/Classes/**ICJoystick.h**
- Desktop/Joystick/Classes/**ICJoystick.m**

ICJoystickCompound Class Reference

`#import <ICJoystickCompound.h>`

Inherits **ICJoystick**.Collaboration diagram for ICJoystickCompound:



Public Member Functions

- (void) - **initialSettings**
- (UIImageView *) - **loadImageJoystickBackground:::::**
- (UIImageView *) - **loadImageJoystickTop:::**
- (int) - **getAcceleration:**
- (void) - **touchesEndedJoystick:**

Private Attributes

- UIImageView * **backgroundImageJoystick**
- UIImageView * **topImageJoystick**
- float **x1**
- float **y1**
- float **radiusJoystickBackground**
- float **radiusDoubleJoystickBackground**
- int **lastAcceleration**
- UIImageView * **boundBoxImage**
- CGPoint **pointsBoundingBox** [BOUND_BOX_POINTS]

Member Function Documentation

- (int) getAcceleration: (UITouch *) *touch*

Responsable for get the acceleration and update the top image.

Parameters:

touch : a UITouch object that represents the presence or movement of a finger on the screen for a particular event. In this case, the particular event that we want is when the user's touch on the joystick's top image.

Returns:

int : the number that represents the current acceleration. It will be a number between 0 and 100.

- (void) initialSettings

Do some initial necessary settings. This method should be called after the images are loaded, or an error will be throw.

- (UIImageView *) loadImageJoystickBackground: (float) *x* : (float) *y* : (float) *width* : (float) *height* : (NSString *) *imagePath*

Responsable for loading a joystick's background image.

Parameters:

x : the x position where the image should be placed.
y : the y position where the image shoud be placed.
width : the desired width for the given image. The image it's resized.
height : the desired height for the given image. The image it's resized.
imagePath : the full path of image.

Returns:

UIImageView* : the ready image.

- (UIImageView *) loadImageJoystickTop: (float) *width* : (float) *height* : (NSString *) *imagePath*

Responsable for loading a joystick's top image.

Parameters:

width : the desired width for the given image. The image it's resized.
height : the desired height for the given image. The image it's resized.
imagePath : the full path of image.

Returns:

UIImageView* : the ready image.

- (void) touchesEndedJoystick: (UITouch *) *touch*

This method should be called when any single touch is ended. If the ended touch was on joystick's top image, then the joystick's top image come back to it's original position.

Parameters:

touch : a UITouch object that represents the ended presence or movement of a finger on the screen for a particular event.

Member Data Documentation

- (UIImageView*) backgroundImageJoystick `[private]`

- (UIImageView*) boundBoxImage `[private]`

- (int) lastAcceleration `[private]`

- (CGPoint pointsBoundingBox[BOUND_BOX_POINTS]) `[private]`

- (float) radiusDoubleJoystickBackground `[private]`

- (float) radiusJoystickBackground `[private]`

- (UIImageView*) topImageJoystick `[private]`

- (float) x1 `[private]`

- (float) y1 `[private]`

The documentation for this class was generated from the following files:

- Desktop/Joystick/Classes/**ICJoystickCompound.h**
- Desktop/Joystick/Classes/**ICJoystickCompound.m**

ICJoystickCompound() Class Reference

Public Member Functions

- (void) - **loadNewPosition:**
- (int) - **calculateAcceleration:**
- (void) - **createBoundingBoxInterna:**
- (BOOL) - **isInsideBoundingBox:**
- (BOOL) - **isInside:**

Properties

- UIImageView * **backgroundImageJoystick**
- UIImageView * **topImageJoystick**
- float **x1**
- float **y1**
- float **radiusJoystickBackground**
- float **radiusDoubleJoystickBackground**
- int **lastAcceleration**
- UIImageView * **boundBoxImage**
- CGPoint **pointsBoundingBox**

Member Function Documentation

- (int) calculateAcceleration: (float) *distance*

- (void) createBoundingBoxInterna: (float) *radius*

- (BOOL) isInside: (float) *distance*

- (BOOL) isInsideBoundingBox: (CGPoint) *newLocation*

- (void) loadNewPosition: (CGPoint) *location*

Property Documentation

- (UIImageView*) backgroundImageJoystick `[read, write, retain]`

- (UIImageView*) boundBoxImage `[read, write, retain]`

- (int) lastAcceleration `[read, write, assign]`

- (CGPoint) pointsBoundingBox `[read, write, assign]`

- (float) radiusDoubleJoystickBackground `[read, write, assign]`

- (float) radiusJoystickBackground `[read, write, assign]`

- (UIImageView*) topImageJoystick `[read, write, retain]`

- (float) x1 `[read, write, assign]`

- (float) y1 `[read, write, assign]`

The documentation for this class was generated from the following file:

- Desktop/Joystick/Classes/**ICJoystickCompound.m**