Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

JOSÉ MARCELO PACHECO

DESENVOLVIMENTO DE APLICATIVO PARA IPHONE

ASSIS
2009

Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

DESENVOLVIMENTO DE APLICATIVO PARA IPHONE

JOSÉ MARCELO PACHECO

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito de Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientadora: Profa. Dra. Marisa Atsuko Nitto

Analisador (1): Alexandre Charles Cassiano

ASSIS
2009

Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

JOSÉ MARCELO PACHECO

DESENVOLVIMENTO DE APLICATIVO PARA IPHONE

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis,como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientadora: Profa. Dra. Marisa Atsuko Nitto

Área de Concentração: Informática

Assis
2009

Dedico este trabalho ao meu pai Leonel Pacheco, a minha mãe Nilza M. F. Pacheco e aos meus irmãos Leonel Pacheco Junior e Maria Cristiane Pacheco.

AGRADECIMENTOS

Ao nosso todo poderoso Deus Jeová por ter me dado força e saúde para estudar e obter os conhecimentos necessários para realizar este trabalho;

Á minha amiga e orientadora Professora Marisa Atsuko Nitto, por sua dedicação e empenho não só neste trabalho de Conclusão de Curso, mas sim por todo período da formação acadêmica. Muito obrigado por ter me ajudado a adquirir as ferramentas necessárias e fundamentais para a realização deste trabalho;

Á Fundação Educacional do Município de Assis por ter-me possibilitado a realização deste trabalho;

Á Minha família, que sempre esteve ao meu lado, dando segurança e tranqüilidade nas horas que precisava de concentração;

Aos meus amigos e companheiros de trabalho Eduardo Borges da Costa, Thiago Tiedtke do Reis e Roberto da Silva Ruy, por cederem espaço e terem auxiliado quando precisei;

Ao meu grande amigo e sócio Luís Fernando Modotti, pelo apoio e sugestões na idéia principal do jogo, além de ter contribuído no desenvolvimento da parte gráfica do game.

RESUMO

O objetivo deste trabalho é desenvolver um aplicativo para a plataforma iPhone SDK (Software Development Kit). Esta plataforma contém o código, informações e ferramentas necessárias para desenvolver, testar, executar e depurar, além de conter as aplicações para ajustar o sistema operacional do iPhone.

O aplicativo é um game simples que desfruta das tecnologias que estão disponíveis somente nos aparelhos da Apple, como o acelerômetro. O desenvolvimento do aplicativo será feita com a Engine Cocos2d, que é um framework para criar jogos 2D e outras aplicações gráficas em *python*. Ela é muito utilizada devido ao fato de simplificar o desenvolvimento de jogos em diversas áreas (controle de fluxo, sprites, efeitos, etc.) e ser uma fonte aberta. É baseada no design Cocos2D que utiliza a mesma API, mas em vez de *python* utiliza Objective-C.

Palavras-chaves: acelerômetro, cocos2d, iPhone, SDK, Game, Objective-C

ABSTRACT

The objective of this paper is develop a application for the iPhone using the SDK (Software Development Kit). This platform contains the code, informations and tools needed to develop test, implement and debug, and also has a applications to fit the iPhone`s operation system.

The application is a simple game that enjoys the technologies that are available only on Apple devices such as accelerometer. The development application will be made with the Engine Cocos2d, which is a framework for creating 2D games and others graphics applications in *python*. It is very used due to fact that simplify the development of games in several areas. (flow control, sprites, effects, etc.) and be a open source. It based on Cocos2d design that uses the same API, but instead of *python* uses Objective-C.

Keywords: accelerometer, Apple, iPhone, SDK, Game, Objective-C

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

CAPITULO 1

INTRODUÇÃO

A proposta deste projeto é desenvolver um aplicativo para iPhone explorando todos os recursos disponíveis. O iPhone é um *smartphone* desenvolvido pela Apple Inc com funções de iPod, câmera digital, internet, mensagens de texto (SMS), visual *voicemail* e conexão wi-fi local. A interação com o usuário é feita através de uma tela sensível ao toque. A Apple registrou mais de duzentas patentes relacionadas com a tecnologia que criou o iPhone [1]. O aparelho está disponível em dois modelos: 8 GB e 16 GB.

A tela sensível ao toque HVGA (320×480 px em 160 ppi) com vidro de qualidade óptica foi especialmente criada para ser usada com um dedo, ou múltiplos dedos para sensibilidade *multi-touch*. Não é necessária uma caneta *stylus*, e nenhuma pode ser usada, visto que o sistema é sensível apenas à capacitância da pele humana. Por esse motivo, às vezes pode ser incômodo manusear o aparelho, como em regiões frias em que os usuários terão que tirar suas luvas para usar a touchscreen. Para entrada de texto, o dispositivo tem um teclado virtual na tela, com um sistema de correção ortográfica, previsão de palavras e um dicionário inteligente que tem a capacidade de aprender novas palavras. Para novos usuários o sistema pode frustrar no começo, com falhas simples como pressionamento errado de teclas e ritmo de digitação. O *multi-touch* permite não só executar comandos tocando na tela, mas ir além, rolando listas com o deslizar do dedo e ampliando e reduzindo imagens com o movimento de pinça do polegar e do indicador. A tela é capaz de reconhecer toques simultâneos, permitindo ao telefone reconhecer gestos e oferecer uma interação mais rica, permitindo que o usuário "manipule" os objetos.

O sistema operacional ocupa aproximadamente 300 MB do total do dispositivo de 8 ou 16 gigabytes. Isto o tornará futuramente capaz de suportar aplicativos futuros da Apple, a qual pretende oferecer um modo simples e prático de fazer o update do sistema operacional do iPhone, de uma maneira *fashion* e *clean* similar à maneira pela qual Mac OSX e iPod são atualizados, e isso é uma grande vantagem se comparados com outros telefones celulares [2].

Em Março de 2008, foi disponibilizado um software *roadmap* do iPhone para desenvolvedores, com muitos recursos corporativos e também foi lançado o SDK, uma plataforma de desenvolvimento de softwares para o iPhone, em que o aparelho receberá jogos que utilizam todo o potencial do dispositivo. A programação SDK envolve o desenvolvimento de um programa que executa nativamente no iPhone [3], [4] e [5]. Estes programas são escritos na linguagem *objective*-C e são compilados em Xcode [6].

Ele oferece ainda suporte a OpenGL/OpenAL para vídeo e áudio, e os desenvolvedores se beneficiarão do acelerômetro presente no iPhone para criar controles baseados em movimentos [7].

O interesse em utilizar esta tecnologia neste projeto é que o mercado corporativo também está crescendo muito, e diversas empresas estão buscando incorporar aplicações móveis a seu dia-a-dia para agilizar seus negócios e integrar as aplicações móveis com seus sistemas de *back-end*. Empresas obviamente visam lucros e mais lucros, e os celulares e *smartphones* podem ocupar um importante espaço em um mundo onde a palavra "mobilidade" está cada vez mais conhecida.

1.1- OBJETIVOS

O objetivo deste projeto é desenvolver um aplicativo para iPhone, que explore suas características inovadoras como o *multi-touch* e seu acelerômetro. Além disso, mostrar a viabilidade de se desenvolver aplicativos para uma plataforma ainda não muito difundida entre os alunos desta universidade, deixando frutos e incentivos para novos projetos.

1.2- JUSTIFICATIVAS

O mercado de celulares está crescendo cada vez mais. Estudos mostram que hoje em dia mais de três bilhões de pessoas possuem um aparelho celular, e isso corresponde a mais ou menos metade da população mundial. As empresas e os desenvolvedores buscam uma plataforma moderna e ágil para o desenvolvimento de aplicações corporativas para auxiliar em seus negócios e lucros. Já os usuários comuns buscam um celular com um visual elegante e moderno, de fácil navegação e uma infinidade de recursos. Para acompanhar

essa evolução da tecnologia é necessário o desenvolvimento de aplicativos para competir por este nicho de mercado. Estes fatos por si só já justificam o desenvolvimento do projeto.

1.3- MOTIVAÇÃO

A tecnologia usada no iPhone ainda é relativamente nova, e sua plataforma de desenvolvimento é pouco difundida na maioria das empresas e entre os programadores. A motivação deve-se ao fato de desenvolver em uma plataforma não muito utilizada nos meios acadêmicos, já que esta tecnologia exige uma plataforma específica (Mac Os) e a própria linguagem de programação que também é especifica (*Objective*-C).

Este trabalho tem também como meta gerar um produto final onde poderá ser distribuído na *Apple Store* (local onde a *Apple* disponibiliza vendas de seus produtos).

1.4- ESTRUTURA DO TRABALHO

O trabalho está divido em capítulos que serão explicados a seguir:

O primeiro capítulo apresente uma introdução, assim como também as justificativas, motivações que levaram ao desenvolvimento deste trabalho.

No segundo capítulo é feita uma descrição da linguagem Objective-C, que será utilizada para o desenvolvimento do aplicativo deste trabalho.

O terceiro capítulo descreve o aparelho iPhone e seu sistema operacional, juntamente com as ferramentas que auxiliam no desenvolvimento de aplicativos para a plataforma iPhone.

O quarto capítulo aborda a Engine Cocos2d que é utilizada para o desenvolvimento de games 2D.

O quinto capítulo descreve a definição do problemas, a modelagem do problema e o desenvolvimento do game.

CAPÍTULO 2

LINGUAGEM OBJECTIVE-C

Neste capítulo será feita uma descrição da linguagem de programação que será usada para o desenvolvimento do aplicativo.

2.1- *OBJECTIVE-C*

O Objective-C (ou Objective C ou ObjC) é uma linguagem de programação refletiva orientada a objetos criada nos idos da década de 1980, mais ou menos na mesma época que o C++, pelos fundadores da Stepstone, mais tarde adquirida pela Next. É na verdade uma camada bem fina sobre a linguagem C padrão. A NeXT Software licenciou a Objective-C em 1988, desenvolvendo suas bibliotecas e o ambiente de desenvolvimento chamado NEXTSTEP. Em dezembro de 1996, a Apple Computer anunciou que estava adquirindo a NeXT Software, assim o ambiente NEXTSTEP se tornou a base para os próximos lançamentos do sistema operacional, MAC OS X. A versão da Apple para esse ambiente de desenvolvimento se chamou Cocoa. [12].

Objective-C é uma extensão do padrão ANSI C, ela é baseada em C. Ela começa com C, em seguida, adiciona algumas pequenas, mas significantes adições à linguagem..Objective-C é uma simples linguagem computacional projetada para fazer sofisticados programas orientados a objeto. Programas escritos para MacOS X são escrito em Objective-C, usando uma tecnologia da década de 80 que amadureceu em um poderoso conjunto de ferramentas.

Compiladores da Apple são baseados na Coleção de Compiladores da GNU. A sintaxe Objective-C herda a sintaxe da GNU C/C++, e compiladores Objective-C trabalham com códigos C, C++ e Objective-C. O compilador reconhece arquivos de códigos Objective-C pela extensão .m, assim como reconhece arquivos de C pela extensão .c [9].

Quando o iPhone foi lançado em 2007, desenvolvedores ficaram entusiasmado pela oportunidade de desenvolver para esse revolucionário aparelho. A Apple então logo providenciou um poderoso Conjunto de Desenvolvimento de Software (Software Development Kit ‘SDK’) que permitiu uma maneira rápida de desenvolver aplicações para iPhone.

A tabela 1 mostra a descrição de cada uma das extensões utilizadas.

Tabela 1: Descrição das extensões.

Extensão	Descrição
.h	Arquivos de cabeçalho. Nos arquivos de cabeçalho contem as classes, tipos e funções.
.m	Típica extensão usada nos códigos fontes. Esses códigos podem conter códigos em Objetive-C e em C.
.mm	Códigos fontes. Podem conter códigos em C++, Objetive-C e em C. Esta extensão deve ser usada apenas se referenciar classes em C++.

2.2- *OBJECTIVE-C* OBJETOS

A Programação Orientada a Objetos (OOP) é construída em torno de objetos. Um objeto associa atributos com operações particulares que os afetam. Em Objective-C essas operações são conhecidas como métodos dos objetos, e os atributos como instancias de variáveis. Sendo assim um objeto é formado de instancias de variáveis e de um grupo de métodos.

2.2.1- Classes

Como muitas outras linguagens orientadas a objeto, classes em Objetive-C provêm uma construção básica de encapsulamento de alguns dados e ações que operam esses dados.

A especificação de classes em Objective-C tem duas distintas peças: a *interface* e a *implementation*, como mostra a figura 1. As interfaces contem a declaração de classe e define as variáveis e os métodos associados à classe. A *interface* geralmente esta dentro do arquivo .h. A *implementation* contem os códigos para os métodos da classe. A *implementation* geralmente esta dentro do arquivo .m.

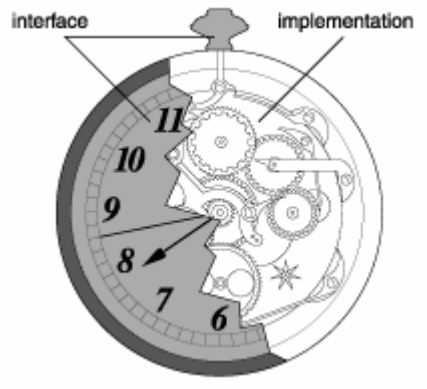

Figura 1: Especificação de classes. Fonte: [9].

A sintaxe de uma declaração de classe começa com uma diretiva do compilador, seguido do nome da classe, a classe a qual ira herdar (separado por dois pontos ":"), declaração das variáveis (dentro de chaves {}), a lista dos métodos da classe e termina com o final da diretiva do compilador (@end). O caracter "ponto e vírgula" (;) marca o final de cada declaração de variável e métodos. A figura 2 mostra a sintaxe de uma declaração de classe.

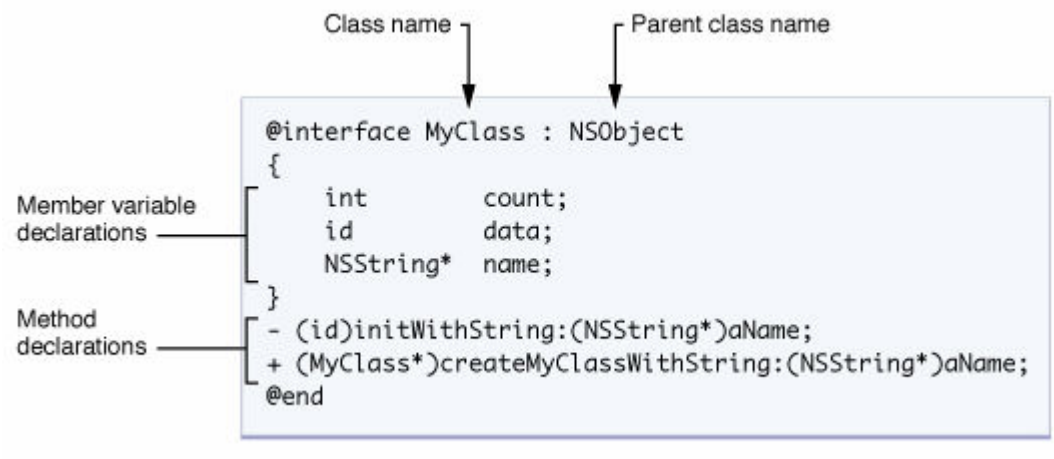

Figura 2: Sintaxe de uma declaração de classe.

Uma classe em Objective-C declara dois tipos de métodos: métodos de instancia e métodos de classe. O método de instancia é um método que só pode ser chamado se houver uma instancia da classe (objeto declarado), enquanto os métodos de classe não necessitam de uma instancia de classe para ser chamado. Para identificar ou declarar um método de instancia é utilizado o caracter "-" e para declarar um método da classe é utilizado o caracter "+"

A sintaxe para declaração de um método é dada por: tipo identificador de método, o tipo de retorno, uma ou mais palavras chaves, o tipo do parâmetro e o nome do(s) parâmetros, como mostra a figura 3.

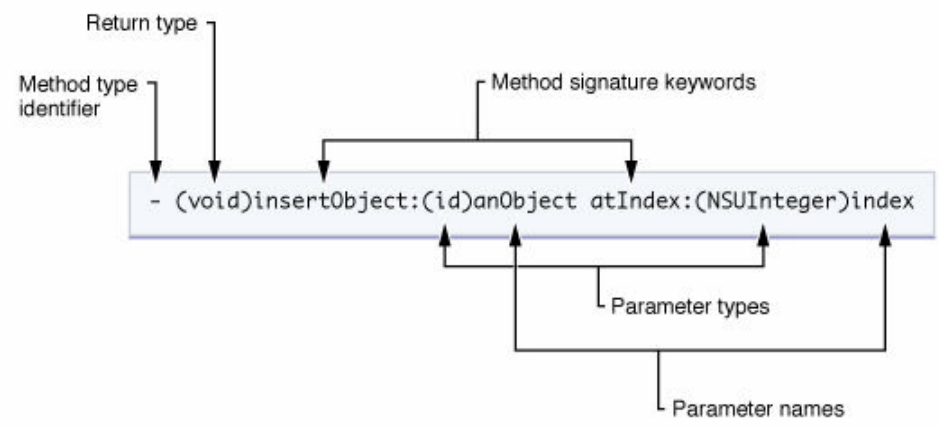

Figura 3: Sintaxe de declaração de um método.

A declaração é precedida do sinal de menos (-), que indica que é um método de instância. O nome do método é a concatenação de todas as palavras chaves (insertObject:atIndex) incluindo o caracter "dois pontos" (:). O "dois pontos" indica a presença de parâmetro, se acaso o método não tiver parâmetro esse caracter deve ser omitido.

A implementação de classes bem como a dos métodos, começam e terminam com diretivas de compilador (*@implementation* e *@end*).

Exemplo:

```
@implementation MyClass

- (id)initWithString:(NSString *)aName
{
    if (self = [super init]) {
        name = [aName copy];
    }
    return self;
}

+ (MyClass *)createMyClassWithString: (NSString *)aName
{
    return [[[self alloc] initWithString:aName] autorelease];
}
@end
```

As diferenças para linguagem C é que Objectice-C adiciona os seguintes recursos:

- Orientação a objetos, adaptada de Smalltalk;
- Tipo boleano – BOOL. Verdadeiro é YES e falso é NO;
- Comentário de linha, iniciado por // (na época C não suportava);
- A diretiva de preprocessador #import, que funciona como #include, mas só inclui o cabeçalho **se** ele não foi incluído ainda;
- O tipo id, similar a void * de C/C++;
- A constante nil, que é um objeto id sem valor: (id)0.

A maior parte dos comandos referentes a orientação a objetos começam com @, como @interface, @implementation, @end, @private, @protected, @public, @protocol, etc.… Entre colchetes é avaliado o ambiente de mensagem, com uma sintaxe muito parecida com a de Smalltalk. O Smalltalk é uma linguagem de programação orientada a objet fracamente tipada.

CAPÍTULO 3

IPHONE

Neste capítulo, será feita uma descrição do iPhone e seu Sistema Operacional, serão também apresentadas as ferramentas que serão usadas no desenvolvimento do aplicativo.

3.1- O APARELHO IPHONE

O iPhone foi lançado no dia 29 de junho de 2007 nos EUA. No *WorldWide Developers Conference* 2008 (WWDC 2008) uma nova versão do iPhone foi lançada pela Apple, o iPhone 3G, que veio com novidades como GPS, jogos, reconhece rede 3G oferecendo uma velocidade de conexão a internet rápida onde há pontos de cobertura 3G. Há apenas quatro botões no aparelho, sendo um o botão *Home*, dois referentes ao controle de volume e o botão *Hold*.

Uma nova versão do iPhone 3G foi lançada no WWDC 2009 denominado iPhone 3G S, o "S" é de *speedy* (velocidade). A figura 4 mostra a imagem do aparelho Iphone.

Figura 4: Aparelho Iphone.

3.1.1- Especificações técnicas

- **Dimensões e Peso:**
  - Altura: 4,5 polegadas (115,5 mm)
  - Largura: 2,4 polegadas (62,1 mm)
  - Profundidade: 0,48 polegadas (12,3 mm)
  - Peso: 133 gramas

- **Cores:**
  - Modelo de 8GB: Preta
  - Modelo de 16GB: Preta ou Branca

- **Celular e tecnologia sem fio**
  - UMTS/HSDPA (850, 1900, 2100 MHz)
  - GSM/EDGE (850, 900, 1800, 1900 MHz)
  - Wi-Fi (802.11b/g)
  - Bluetooth 2.0 + EDR

- **GPS**
  - GPS Assistido – menos para o Egito

- **Monitor**
  - Tela widescreen de 3,5 polegadas com tecnologia *Multi-Touch*
  - Resolução de 480 por 320 pixel a 163 ppi
  - Visualização de vários idiomas e caracteres simultaneamente

A figura 5 mostra a tela do monitor do aparelho iPhone.

Figura 5: Monitor do iPhone.

- **Áudio**
  - Resposta de freqüência: 20Hz a 20.000Hz
  - Formatos de áudio suportados: ACC, Protected AAC, MP3, MP3 VBR, Audible (formatos 2, 3 e 4), Apple Lossless, AIFF e WAV
  - Limite máximo de volume configurável pelo usuário

- **Fones de ouvido**
  - Fone de ouvido estéreo com microfone integrado
  - Resposta de frenquência: 20Hz a 20.000Kz
  - Impedância: 32 ohm

A figura 6 mostra o fone de ouvido utilizado pelo Iphone.

Figura 6: Fone de ouvido.

- **Vídeo**

Formatos de vídeo suportados:

- Vídeo H.264, até 1.5 Mbps, 640 por 480 pixels, 30 quadros por segundo, versão Low-Complexity do Baseline Profile H.264 com áudio AAC-LC até 160 Kbps, 48kHz, som estéreo em formatos de arquivo .m4v, .mp4, e .mov;
- Vídeo H.264, até 768 Kbps, 320 por 240 pixels, 30 quadros por segundo, Baseline Profile até Level 1.3 com áudio AAC-LC até 160 Kbps, 48kHz, som estéreo em formatos de arquivo .m4v, .mp4, e .mov;
- Vídeo MPEG-4, até 2.5 Mbps, 640 por 480 pixels, 30 quadros por segundo, Simple Profile com áudio AAC-LC até 160 Kbps, 48kHz, som estéreo nos formatos de arquivo .m4v, .mp4, e .mov.

A figura 7 mostra a apresentação de um vídeo no monitor do aparelho iPhone.

Figura 7: Apresentação de vídeo no iPhone.

- **Câmera e fotos**
  - 2.0 megapixels
  - Identificação fotográfica de locais
  - Integra ao iPhone e aplicativos de terceiros

- **Suporte para idiomas**

  - Suporte para os idiomas dinamarquês, italiano, português, francês, inglês, alemão, japonês, holandês, espanhol, finlandês, norueguês, sueco, coreano, chinês simplificado, chinês tradicional, russo, polonês, turco e ucraniano.
  - Suporte para teclado internacional e dicionário nos idiomas inglês (Estados Unidos), inglês (Reino Unido), francês (França), francês (Canadá), alemão, japonês, holandês, italiano, espanhol, português (Portugal), português (Brasil), dinamarquês, finlandês, norueguês, sueco, coreano (não há dicionário), chinês simplificado, chinês tradicional, russo, polonês, turco e ucraniano.

- **Suporte para anexos no Mail**

  - Tipos de documentos suportados: .jpg, .tiff, .gif, .doc e .docx (Microsoft Word), .htm e .html (páginas da internet), .key (Keynote), .numbers (Numbers), .pages (Pages), .pdf (Preview e Adobe Acrobat), .ppt e .pptx (Microsoft PowerPoint), .txt (texto), .vcf (dados de contacto), .xls e .xlsx (Microsoft Excel).

- **Botões e controles externos**

  - Sleep/wake
  - Toque/silêncio
  - Volume +/-
  - Home

A figura 8 mostra os botões e controles externos do aparelho.

Figura 8: Botões e controles externos.

- **Sensores**
  - Sensor de luz ambiente
  - Sensor de proximidade
  - Acelerômetro

A figura 9 mostra os sensores do aparelho.

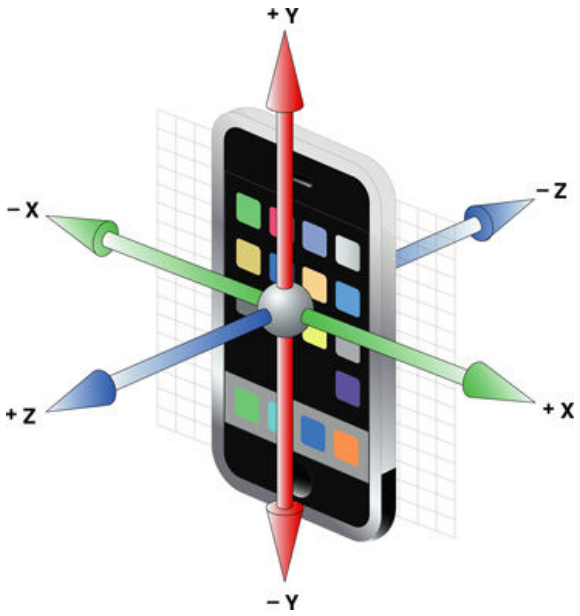

Figura 9: Sensores.

- **Alimentação e bateria**

  - Bateria recarregável de lítio íon incluída
  - Carga via USB do sistema ou fonte de alimentação
  - Tempo máximo de conversação
    Até 5 horas em 3G
    Até 10 horas em 2G
  - Tempo máximo no modo de espera: Até 300 horas
  - Uso de internet:
    Até 5 horas em 3G
    Até 6 horas em Wi-Fi
  - Reprodução de vídeo: até 7 horas
  - Reprodução de áudio: até 24 horas

- **Requisitos do sistema Mac**
  - Computador Mac com porta USB 2.0
  - Mac OS X v10.4.10 ou posterior
  - iTunes 7.7 ou posterior
- **Requisitos do sistema Windows**
  - Computador PC com porta USB 2.0
  - Windows Vista; ou Windows XP Home ou Professional com Service Pack 2 ou posterior
  - iTunes 7.7 ou posterior
- **Requisitos ambientais**
  - Temperatura operacional: 32° a 95° F (0° a 35° C)
  - Temperatura não-operacional: -4° a 113° F (-20° a 45° C)
  - Umidade relativa: 5% a 95% não-condensável
  - Altitude máxima de operação: 10.000 pés (3000 m)

3.2- Iphone OS (OS-Operating System)

O iPhone OS é um sistema operacional que funciona no iPhone e no iPod Touch, desenvolvido pela Apple, e que inclui tecnologias que serão usadas para executar as aplicações nativamente no Iphone. Embora algumas tecnologias do SO do Iphone sejam subjacente ao MacOS X, o Iphone OS foi concebido para satisfazer as necessidades de um ambiente móvel, onde as

necessidades do usuário são ligeiramente diferentes. Há varias tecnologias presente apenas no Iphone OS, como *mult-Touch* e acelerômetro.

O iPhone possui uma CPU ARM. O sistema operacional ocupa aproximadamente 300 MB.

3.3- Arquitetura do Iphone OS

A arquitetura do Iphone OS é similar a arquitetura básica encontrada no MacOS X. Num contexto de alto nível, o iPhone OS atua como um intermediário entre o hardware do iPhone e a aplicação que aparece na tela. As aplicações criadas nunca interagem diretamente com o hardware, ao invés disso interage com uma camada anterior, que por sua vez acessa os *drivers* apropriados. Esta abstração protege a aplicação de mudanças que pode haver no hardware.

O Framework do iPhone OS é dividido em quatro grandes camadas, como mostra a figura 10. Cada uma destas camadas contém uma variedade de frameworks que pode ter acesso ao escrever programas no iPhone SDK. Geralmente, deve se preferir uma camada de alto nível quando estiver codificando (os indicados em direção ao topo do diagrama).

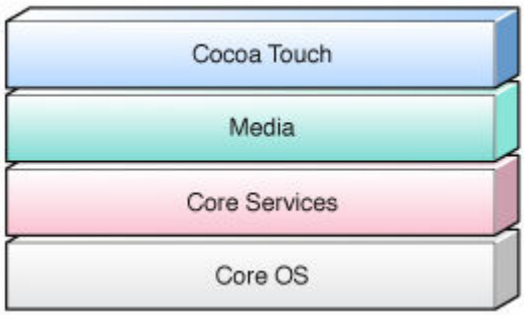

Figura 10: Arquitetura do iPhone OS. Fonte:[10].

3.3.1- Camada Cocoa Touch

A camada Cocoa Touch é uma das mais importantes no iPhone OS. Abrange os principais infra-estruturas necessárias para uma implementação no iPhone OS.

- **Apple Push Notification Service:** No Iphone OS 3.0 ou superior, o Apple Push Notification Service prove um modo de alertar seu usuário de uma nova notificação, mesmo quando sua aplicação não esta ativa.

- **Address Book UI Framework:** O Address Book UI Framework é uma interface em Objective-C que usa a interface padrão para criar um novo contato, editar e selecionar contatos existentes. Este framework simplifica o trabalho de mostrar as informações de um contato na sua aplicação e também garante que a aplicação desenvolvida usara a mesma interface de outras aplicações.

- **Map Kit Framework:** Prove uma inteface de mapa que pode ser inserida no desenvolvimento de uma aplicação. Podem ser inseridos vários atributos no mapa dinamicamente, incluindo o nome do lugar onde o mapa esta mostrando e localização do usuário no mapa.

- **UIKit Framework:** O UIKit framework contem interface programadas em Objetive-C que fornecem as infra-estruturas essenciais para execução gráfica, eventos de aplicação no iPhone OS. Todas as aplicações no iPhone OS usam esse *framework* para implementar suas características como tais:
  - Gerenciamento da aplicação;
  - Suporte a copiar, colar e recortar;
  - Suporte a gráficos e janelas na aplicação;
  - Suporte a toque, e manuseio baseado em eventos;
  - Suporte para Apple *push notification service*;
  - Suporte a controle de acesso a usuários;
  - Suporte a dados do Acelerômetro;
  - Acesso a câmera;
  - Suporte a biblioteca de fotos do usuário;
  - Informações do nome e modelo do aparelho;
  - Informação do estado da bateria;
  - Informação da proximidade do sensor.

3.3.2- Camada Media

Na camada de Media estão as tecnologias de áudio, vídeo e as gráficas usadas para crias as melhores multimídias presentes no aparelho. Usando essa camada torna-se fácil criar gráficos avançados e rápidas animações.

- **Tecnologias Gráficas:** Uma das mais importantes partes das aplicações para iPhone é a qualidade gráfica das mesmas. As tecnologias gráficas são divididas partes: Quartz, Core Animation e OpenGL ES.
  - **Quartz:** É a mesma *engine* utilizada pelo Mac OS X para desenhos vetoriais. Prove suporte a renderização com *anti-aliased*, imagens, cores, transformações de coordenadas espaciais, criação de documentos PDF, etc.
  - **Core Animation:** É um avançado sistema de animação que usa renderização otimizada para implementar complexas animações e efeitos visuais.
  - **OpenGL ES:** Disponibiliza ferramentas para desenhos 2D e 3D. Trabalha em perto do hardware e implementa uma taxa de frame para tela cheia para games. A versão OpenGL ES 2.0 tem suporte para Vertex Shaders.

- **Tecnologias de Audio:** As tecnologias de áudio disponibilizadas no iPhone OS são designadas para ajudar a prover ricas aplicações de áudio para os usuários. As tecnologias de áudio são divididas em três partes: AV Foundation, Core Audio e OpenAL.
  - **AV Foundation:** Suporta tocar arquivos de áudio, podendo tocar múltiplos sons simultaneamente.
  - **Core Áudio:** É nativo do iPhone OS, é usado para tocar e gravar áudios.
  - **OpenAL:** Adicionalmente ao Core Audio, iPhone OS inclue suporte para Open Áudio Library (OpenAL). Pode-se usa-la para melhorar a performace de áudio da aplicação. É muito usada em games e aplicações onde se necessite uma excelente performace de áudio.

3.3.3- Core Services Layer

Core Services implementa os serviços fundamentais do sistema que todas as aplicações usam. Mesmo que não se use esses serviços diretamente, muitas partes do sistema são construídas em cima dela.

- **Address Book:** Implementa aos contatos armazenados no aparelho do usuário. Se a aplicação usa informações de contato, pode-se usar esse

framework para acessa, modificar e gravar os contatos do usuário no bando de dados.

- **Core Foundation: Ne**ste framework esta incluso suporte a:
  - Tipo de coleções de dados (array, sets, e varias outras).
  - **Gerenciamento de String.**
  - **Gerenciamento de tempo e data.**
  - **Manipulação de URL e *streams*.**
  - **Threads.**
  - **Comunicação por *socket* e portas.**
- **Core Location:** Permite determinar a latitude e longitude do aparelho. O aplicação que implementará mapas usa essa interface para mostrar a posição do usuário no mapa.
- **SQLite:** Pode-se criar uma base de arquivos local e gerenciar tabelas e dados nesses arquivos.
- **XMLSupport:** Esta interface a classe NSXMLParser que tem os elemetos de um arquivo XML. Pode-se escrever ou ler arquivos XML rapidamente e também transforma-lo em arquivos HTML.

3.3.4- Core OS Layer

O Core OS Layer inclui o kernel-level Ele pode acessar threading, arquivos, rede, I/O, e memória

- **CFNetwork:** Esta interface implementa uma abstração orientada a objeto que permite trabalhar com protocolos de redes. Esta abstração da um detalhado controle sobre os protocolos. É usada para simplificar tarefas como serviços de FTP e HTTP.
- **Accessory Support:** Implementa suporte a comunicação entre iPhone ou iPod.
- **Segurity:** IPhone OS tem um framework explicito para controle da segurança. É usado para garantir a segurança dos dados das aplicações. Ele suporta a geração de criptografias
- **System:** Essa parte engloba o kernel, drivers, e as interfaces de baixo nível do sistema operacional. O kernel é baseado no Mac e é responsável por cada aspecto do sistema operacional. Ele gerencia a memória virtual, threads, sistema de arquivo e rede. Os drivers fazem uma "ponte" entre o hardware e as interfaces. Porem, p acesso ao kernel e aos drivers é restrito e limitado as interfaces com o

propósito de segurança. Iphone OS disponibiliza acesso para muitas funções de baixo nível do sistema operacional.

- Threads (POSIXthreads)
- Acesso a sistema de arquivos.
- Padrão de entrada e saída I/O.
- Alocação de memória.
- Computações matemáticas.

3.4- A hierarquia dos objetos do Iphone

Dentro dos frameworks do iPhone OS, o que se pode notar é a facilidade para acessar a uma imensa riqueza de classes que estão dispostos em uma enorme hierarquia. Na figura 11, serão mostradas muitas das classes que utilizadas e organizadas em hierarquia. Elas são apenas uma parte do que está disponível.

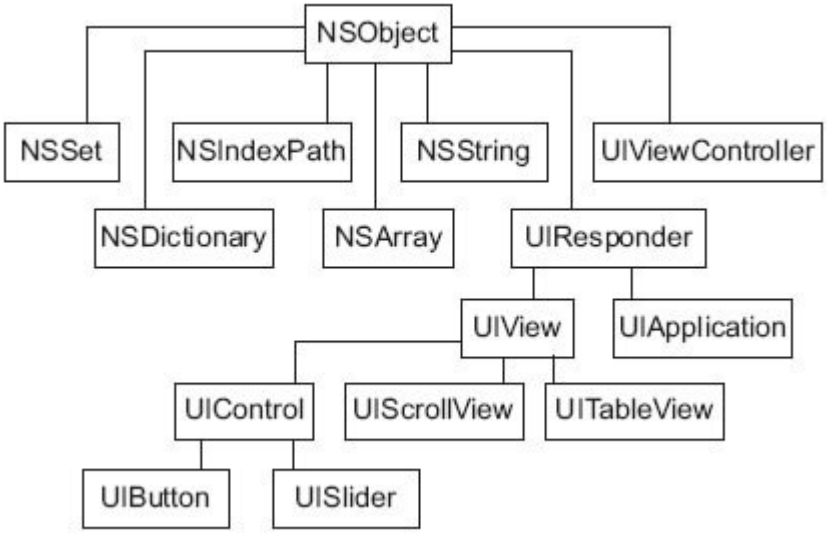

Figura 11: Hierarquia dos objetos do iPhone OS. Fonte: [11].

Conforme mostrado na figura 11, os objetos que tem maior probabilidade de uso estão divididos em duas categorias gerais.

3.4.1- AS Classes NS

As classes NS provêm do Core Services Foundation framework (um Cocoa equivalente do Core Foundation framework), que contém um grande número de tipos de dados fundamentais e de outros objetos. Em geral, deve se usar as classes fundamentais do Cocoa como NSString e NSArray sempre que puder. Isto porque eles tendem a se dar muito bem uns com os outros e com o framework UIKit, e portanto, é menos provável que sejam encontrados erros bizarros. Apesar de não ser mostrado, NSNumber é outra classe que sempre deve ser usada. Deve ser o seu principal objeto numérico quando estiver fazendo qualquer tipo de trabalho complexo com números. Pode ser utilizada para reter muitos tipos de valores numéricos, flutuantes, inteiros e mais. Os objetos que podem conter coleções de valores são NSArray (um array numérico) e NSDictionary (um array associativo). Somente estes interessam mais nas classes NS [12].

3.4.2- AS Classes UI (UI-User Interface)

A segunda categoria geral de classes contém as classes UI. Estes vêm do Cocoa Touch do framework do UIKit. Ela inclui todos os objetos gráficos que são usados, bem como todas as funcionalidades para o iPhone OS event model, muitos deles aparecem no UIResponder [12].

3.4.3- Windows e Views

Como as classes IU demonstram, o iPhoneOS é profundamente enraizada na idéia de uma interface gráfica de usuário. Por isso, será examinando algumas das principais abstrações gráfico embutidas no UIKit. Existem três abstrações principais: Windows, Views, e View Controllers. Um Window é algo que toma toda a tela do iPhone. Há apenas um deles para a sua aplicação, e é o recipiente para tudo o que a sua aplicação mostrar. Um View é o conteúdo real mostrado na sua tela. E pode se ter vários deles, cada um abrangendo diferentes partes do Window (Janela) ou fazendo coisas diferentes em tempos diferentes. Eles são todos derivados da classe UIView. No entanto, não basta pensar em um View como um recipiente vazio.

Atualmente, praticamente qualquer objeto que usar a partir do UIKit será uma subclasse do UIView que apresenta uma grande quantidade de comportamentos herdados de si mesmo. Entre as grandes subclasses de UIView são UIControls, que lhe dará botões, cursores e outros itens que os usuários podem manipular o seu programa e o UIScrollableViews, que

fornecerá aos usuários o acesso ao texto que não pode aparecer mais de uma vez. Um View Controller faz o que seu nome sugere. Ela atua como o elemento controlador do modelo MVC e no processo de gerenciamento de telas cheias de texto, que é às vezes chamado por um View. Como tal, ele cuida de eventos e de atualização para o seu view [12].

Os View Controllers são dividimos em dois tipos:

- **View Controllers Básicos:** são aqueles que apenas gerenciam telas de texto (como o tab bar controller);
- **View Controllers Avançados:** são aqueles que permitem que um usuário se desloque entre as várias páginas de texto (tais como o navigation bar controller e o tab bar controller).

A figura 12 mostra como esses três tipos de objetos inter-relacionam. Windows, Views, e View Controllers são, em última análise, parte de uma visão hierarquia. Esta é uma árvore de objetos que começa com o Window na sua raiz.

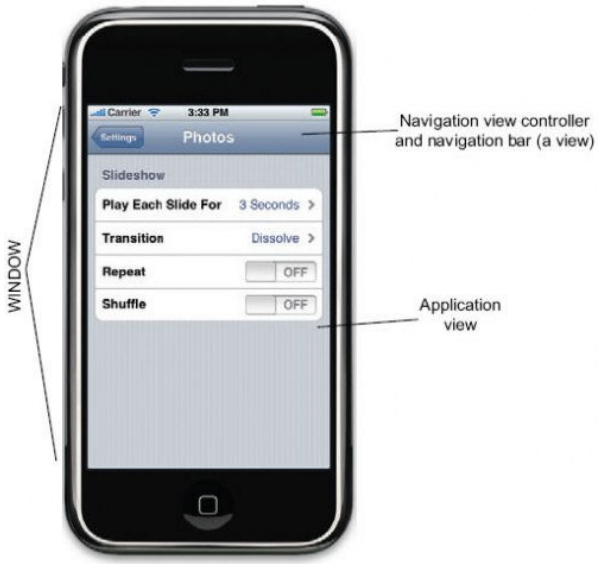

Figura 12: Inter-relação dos objetos. Fonte: [12].

Esta é uma árvore de objetos que começa com o Window na sua raiz. Um programa simples pode apenas ter um Window com um View nela. A maioria dos programas terá início com um Window, um View e um View Controlller em que, talvez apoiados por View Controllers adicionais, controlam Views que deverão ter os seus próprios subviews.

3.5- Ferramentas de Desenvolvimento

Qualquer aplicação computacional, independente da tecnologia empregada necessita de ferramentas de desenvolvimento. As escolhas dessas ferramentas dependem do tipo de aplicação a ser feita. Serão apresentados os passos para a instalação das ferramentas utilizadas neste projeto de pesquisa.

3.5.1- MacOS X instalação

Para instalar o Mac OS X (versão 10.4 ou superior) são necessários ter um DVD de instalação do mesmo. Coloque o DVD no drive, e reinicio o computador. Após ter iniciado o processo de boot irá aparecer a tela do Darwin, como mostra a figura 13. Pressione F8 e aguarde carregar o sistema.

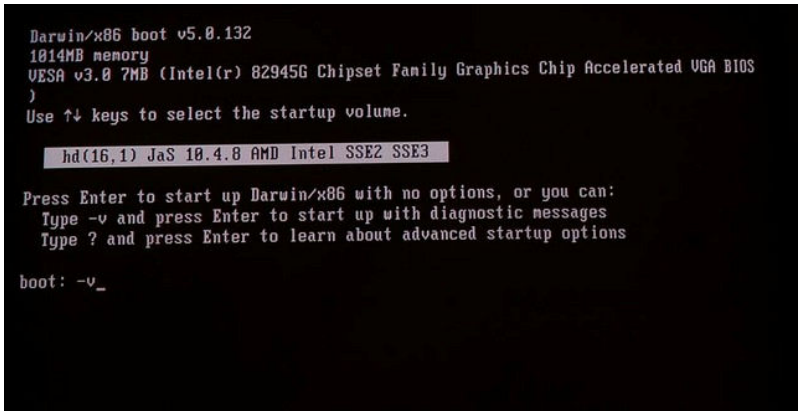

Figura 13: Interface do Darwin. Fonte:[13].

Selecione o idioma desejado e clique em avançar, como mostra a figura 14.

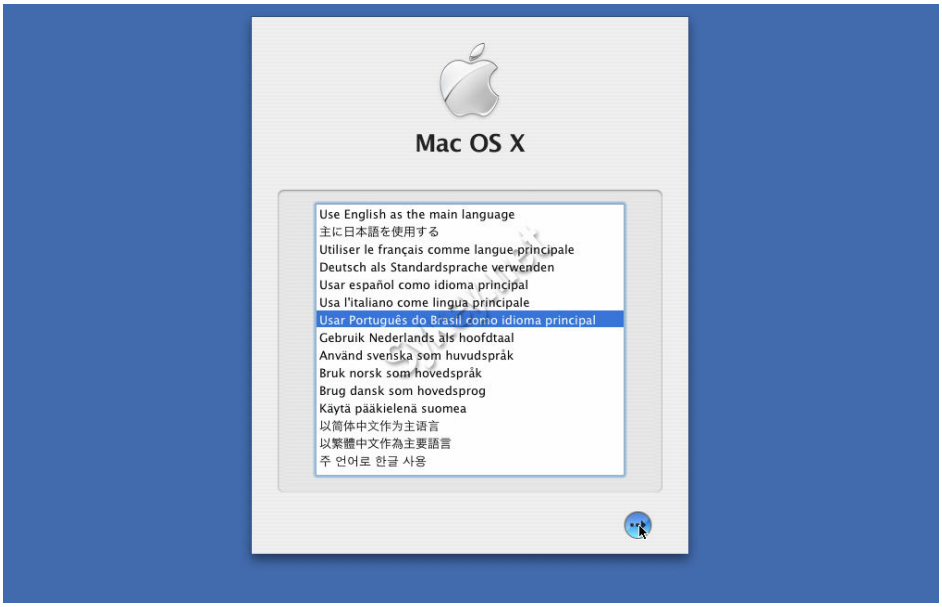

Figura 14: Interface de seleção de idiomas. Fonte:[13].

Para formatar a partição onde o Mac OS X será instalado Selecionar o menu Utilitários -> Utilitário de Disco, conforme figura 15.

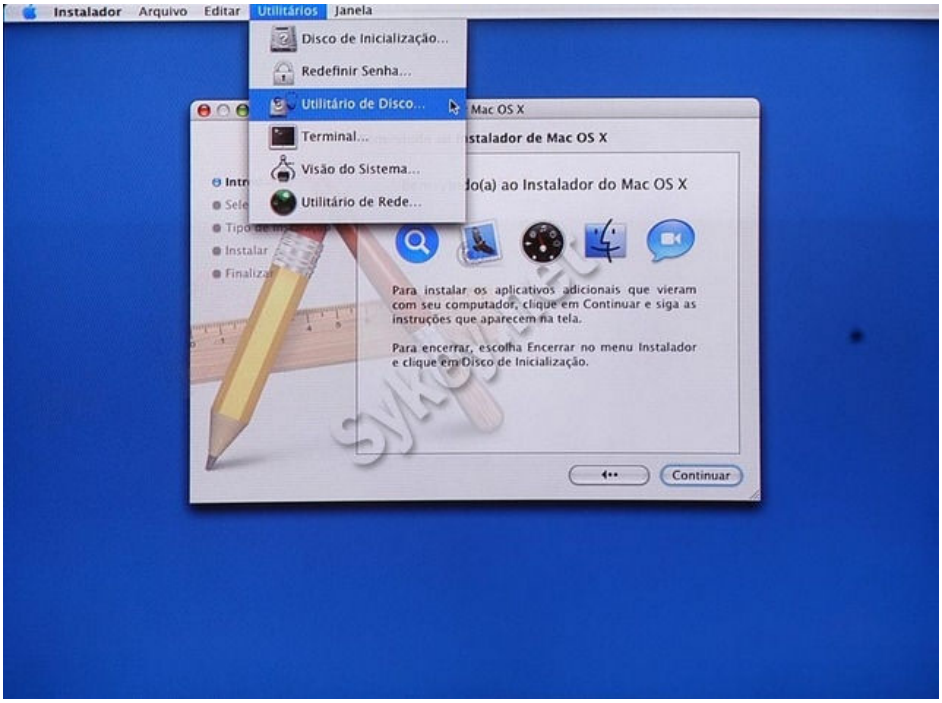

Figura 15: Acessando Utilitário de Disco Fonte:[13].

Será exibida a lista de partições. Clique partição que será instalada o sistema operacional e logo em seguida no botão Excluir. Como tipo de partição, escolha **Mac OS Expandido** e depois clique em **Excluir**, conforme figura 16.

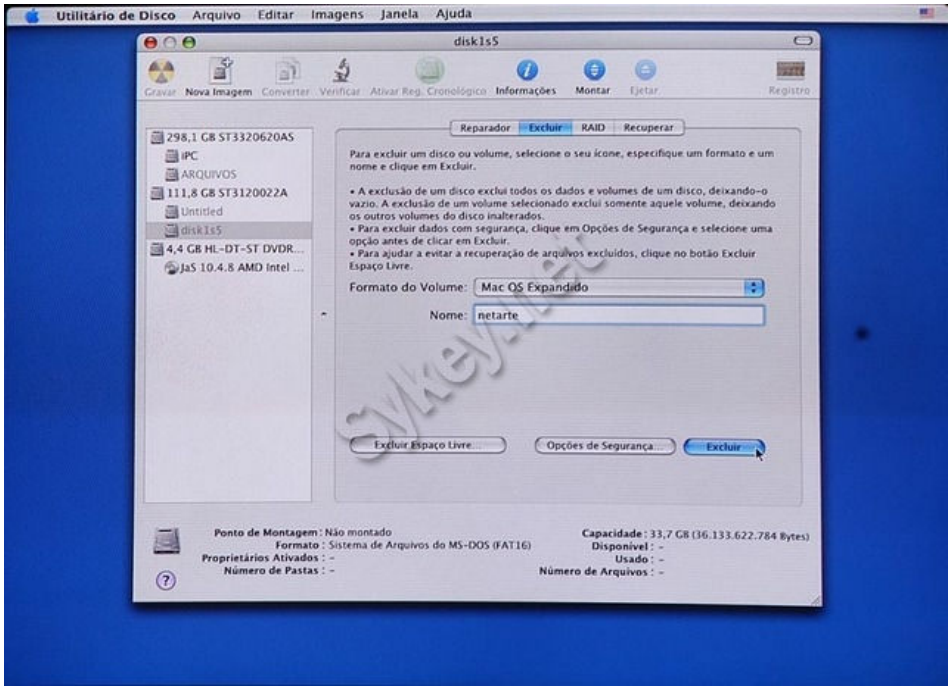

Figura 16: Excluir partição. Fonte: [13].

Após concluído a formatação da partição, feche o utilitário de disco, conforme a figura 17, assim voltara para a tela de instalação.

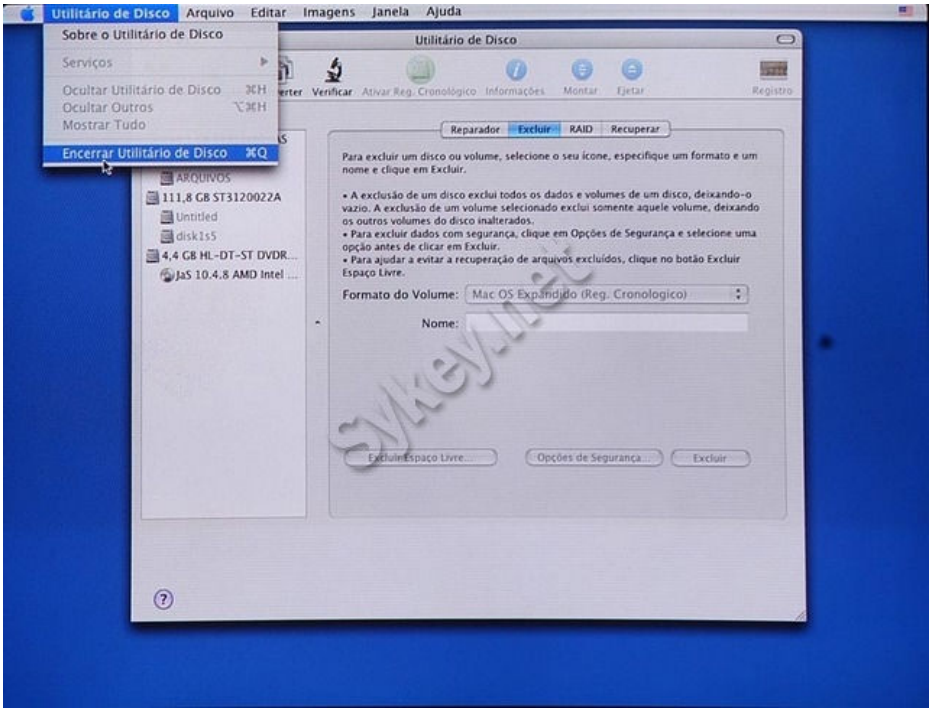

Figura 17: Encerrar utilitário de disco. Fonte: [13].

Clique em concordar no Contrato de Licença, selecione a partição criada para fazer a instalação, conforme figura 18.

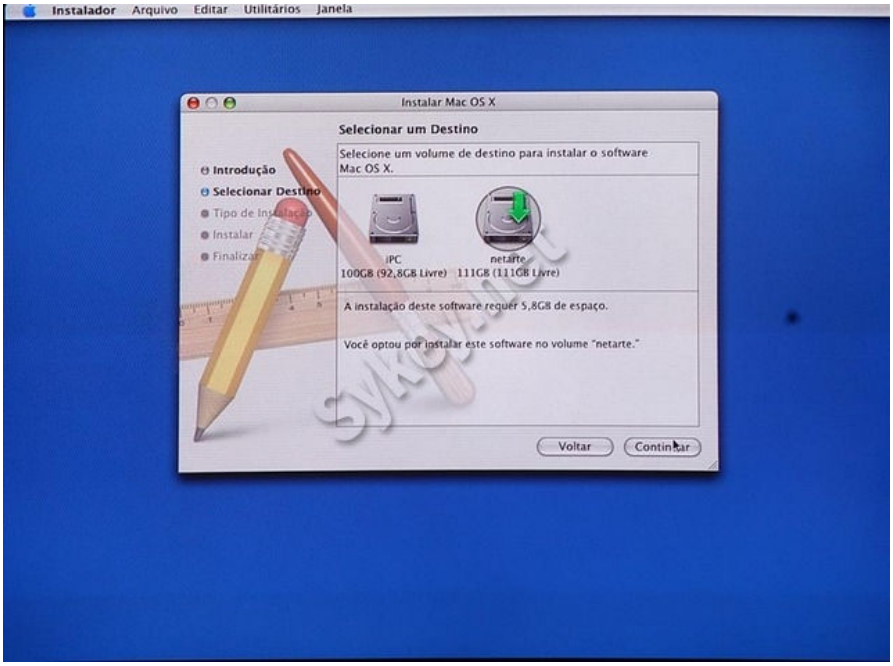

Figura 18: Seleção da partição. Fonte: [13].

Para personalizar sua instalação clique em Personalizar no canto da janela, conforme figura 19.

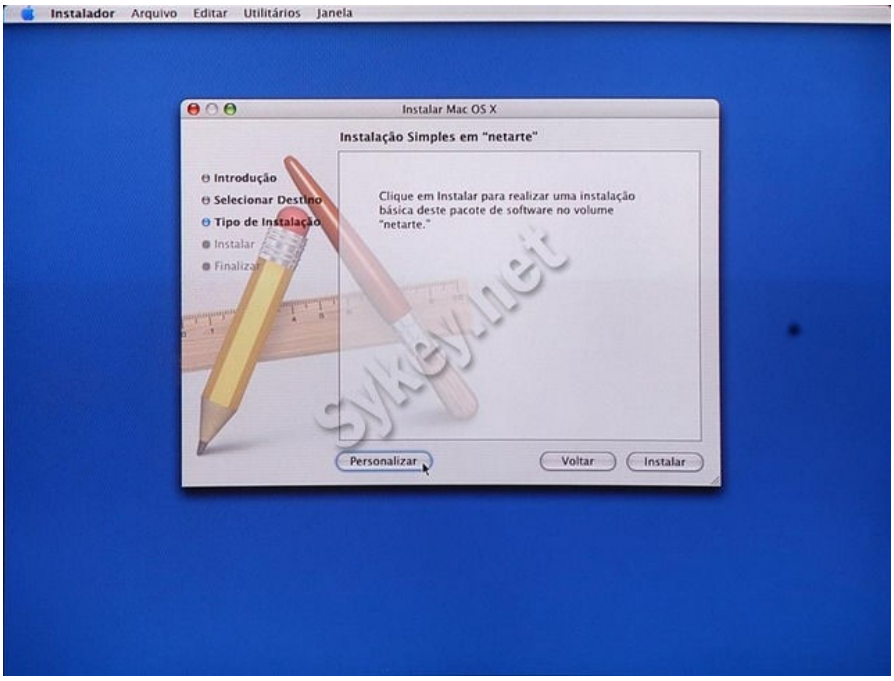

Figura 19: Botão personalizar. Fonte: [13].

Escolha os drivers para impressoras, fontes e linguagens adicionais, conforme figura 20.

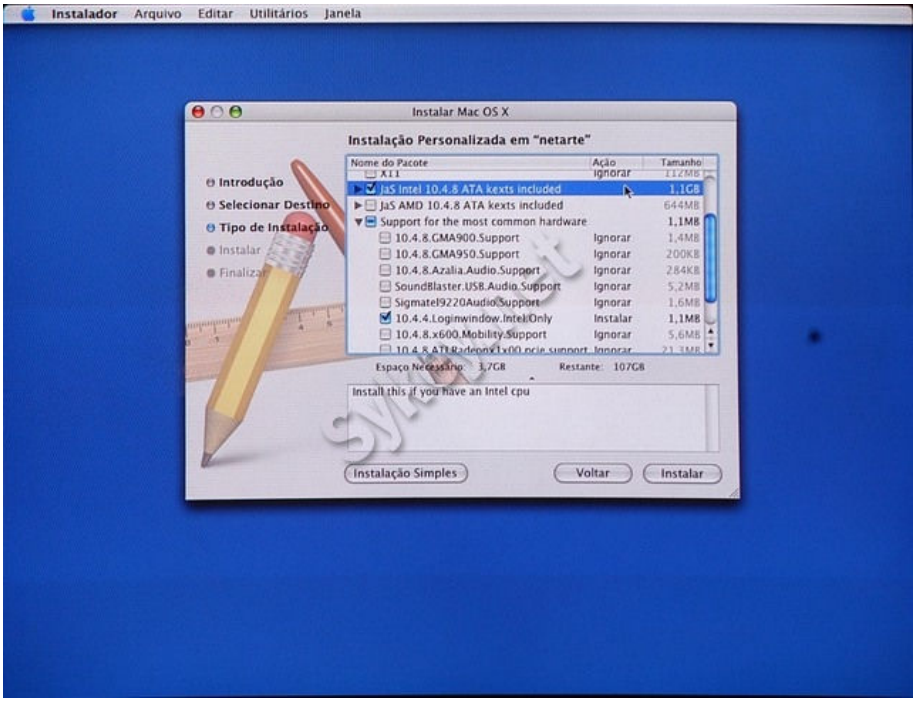

Figura 20: Configurar instalação. Fonte:[13].

Para instalar basta clicar no botão Instalar. Após alguns minutos o sistema estará instalado e pronto para ser usado, conforme figura 21.

Figura 21: Desktop do Mac OS X. Fonte: [13].

3.5.2- iPhone SDK Instalação

Os requisitos mínimos necessários para a instalação do iPhone SDK são: Sistema Operacional MacOS X Leopard e 3.5 Gb de espaço livre em disco para a instalação. O Primeiro é fazer o download do iPhone SDK direto do site da Apple Developer Connection pelo endereço: http://developer.apple.com/iphone, para isso é necessário se cadastrar no site. O download do SDK é gratuito. O tamanho do arquivo é de aproximadamente 2GB e seu formato é de uma imagem de arquivo (.dmg).

Após a realização do download, executar os seguintes passos: dois clique no arquivo para montar o volume do iPhone SDK, aparecerá um ícone no Desktop. Abra este ícone, aparecerá uma janela dentro desta janela contendo o pacote nomeado de iPhone SDK. Clicar duas vezes neste pacote para começar o processo de instalação. Após concordar com as licenças aparecerá uma interface de comunicação, conforme figura 22.

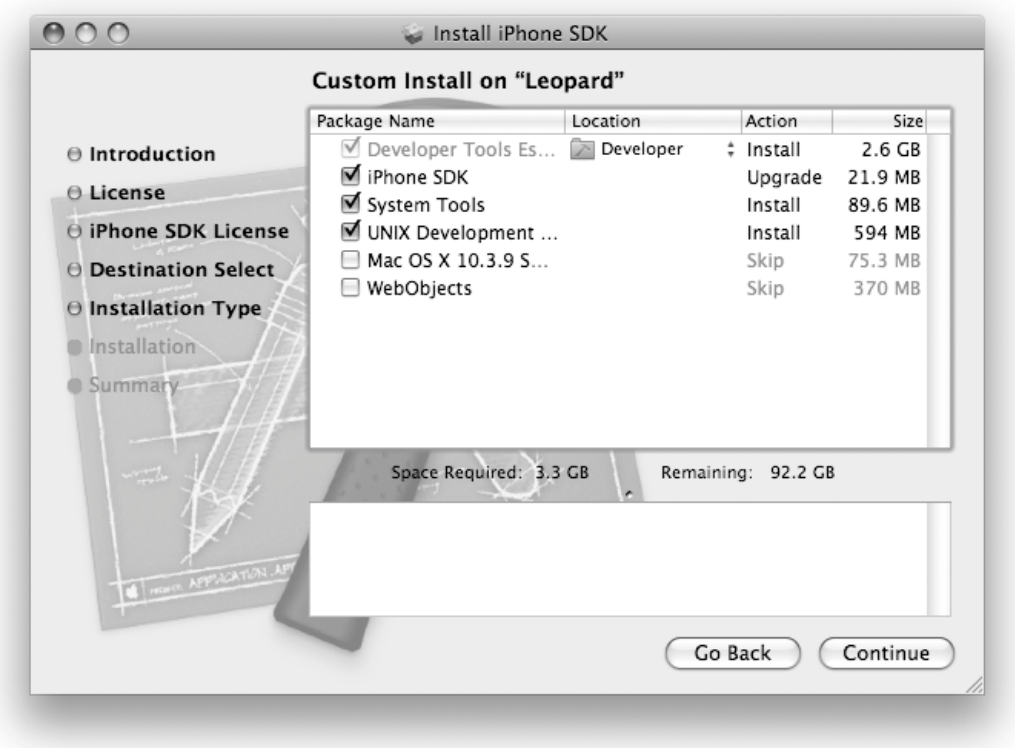

Figura 22: Interface de Instalação.

Ao final do processo serão instalados o XCode e o iPhone SDK dentro de /Developer. A figura 23 mostra o final da instalação

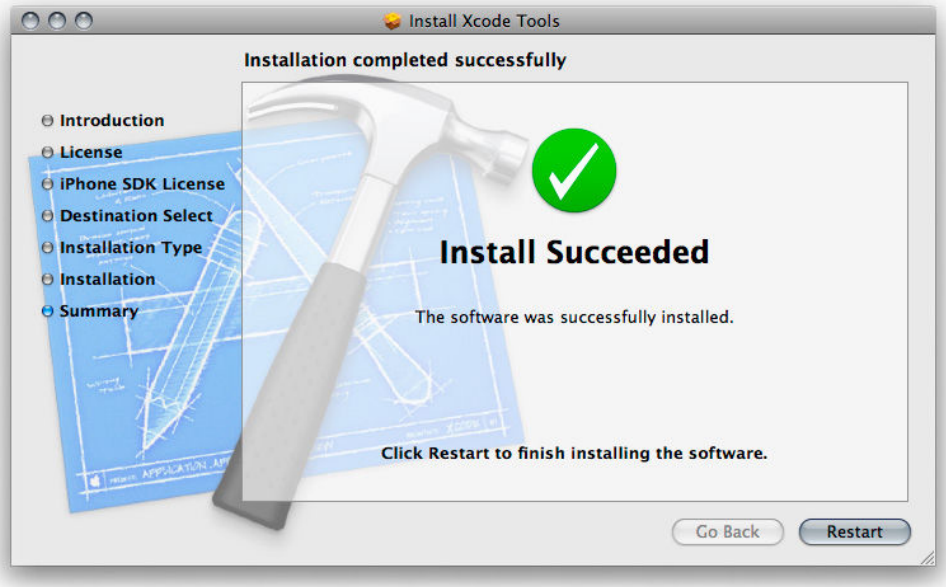

Figura 23: Final da instalação.

O conteúdo instalado com SDK é mostrado figura 24.

Figura 24: Conteúdo do SDK do iPhone.

3.5.3- Xcode

XCode é o ambiente de desenvolvimento para o Mac OS X, é uma maneira rápida e fácil para criar aplicações para Mac OS e iPhone OS. Xcode é a ferramenta mais importante no arsenal de desenvolvimento do iPhone. Ele disponibiliza um ambiente de desenvolvimento de projetos compreensivo, editor de código completo e um depurador gráfico. Xcode é construído em torno de diversas ferramentas GNU de código aberto, ou seja, o gcc (compilador) e (depurador).

3.5.3.1- Criando um Projeto no Xcode

Todo software começa com um projeto. Um projeto é um repositório para todos os elementos usados para modelar e construir um produto, incluindo códigos fontes, interface com o usuário, sons, imagens, bibliotecas e suporte para *frameworks*. Xcode disponibiliza vários modelos de projeto para iPhone. Ele fornece opção para selecionar vários tipos de aplicações, que serão descritos a seguir:

- **Navigation-Based Application** – Uma aplicação que apresenta uma hierarquia de dados, usando múltiplas telas. Uma agenda de contato é um exemplo de *navigation-Based Application;*
- **OpenGL ES Application** – Uma aplicação que usa OpenGL ES-based para apresentar imagens ou animações;
- **Utility Application** – Uma aplicação que apresenta um janela principal e permite que o usuário escolha várias outras janelas (telas);
- **View-Based Application** – Uma aplicação que usa apenas uma janela para a interface com o usuário;
- **Window-Based Application** – Este modelo serve como ponto de partida para qualquer outra aplicação. Ë usada quando se deseja implementar sua própria hierarquia de janelas.

Para desenvolver uma aplicação para iPhone é preciso trabalhar com projetos no Xcode. A maioria do seu trabalho será feito na janela principal do Xcode, que mostra e organiza o código e outros elementos necessários para construir o projeto. A janela principal é descrita na figura 25.

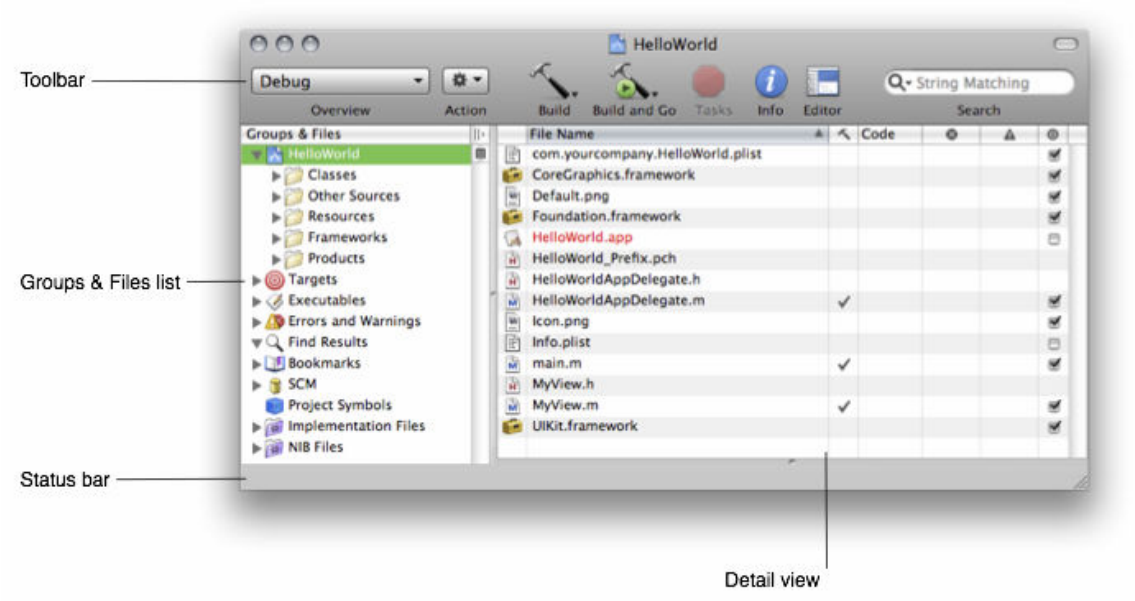

Figura 25: Janela principal do Xcode.

Será feita uma descrição da interface principal do Xcode.

- **Groups & Files list:** Disponibiliza uma visão geral dos arquivos do projeto. Ele permite mover arquivos e pastas e organizar a estrutura de arquivos e diretórios do projeto.
- **Detail view:** Mostra os itens selecionados no *Groups & Files list*.
- **ToolBar:** Disponibiliza um acesso rápido aos comandos mais comum do Xcode.
- **Status Bar:** Mostra as mensagens do seu projeto. Durante uma operação de compilação, por exemplo, Xcode mostra o progresso da compilação.
- **Favorite Bar:** Permite armazenar e recuperar rapidamente as localizações mais acessadas no projeto. A *Favorite Bar* não é mostrada por padrão. Para visualizar é necessário acessar *View->Layout->Show Favorites Bar*.

A figura 26 mostra o fluxo do funcionamento do Xcode.

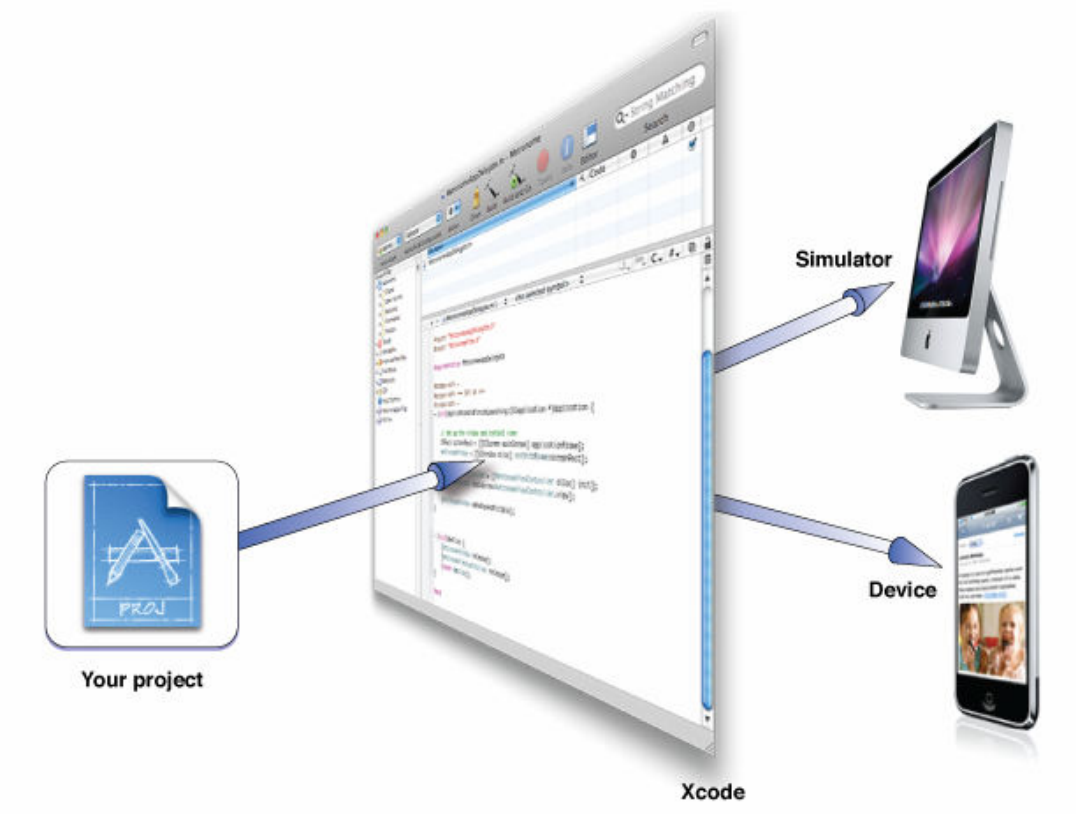

Figura 26: Fluxo do Xcode.

3.5.4- Interface Builder

A *Interface Builder* (IB) é uma ferramenta que auxilia na construção da interface gráfica para o usuário, e a liga com aquelas pré-compiladas no código feito em Xcode. Com a *Interface Builder* é possível desenhar toda interface usando uma ferramenta gráfica e conectar com os objetos e métodos chamados na aplicação.

As figuras 27, 28 e 29 mostram partes da *Interface Builder*, e a figura 30 mostra toda a tela do *Interface Builder.*

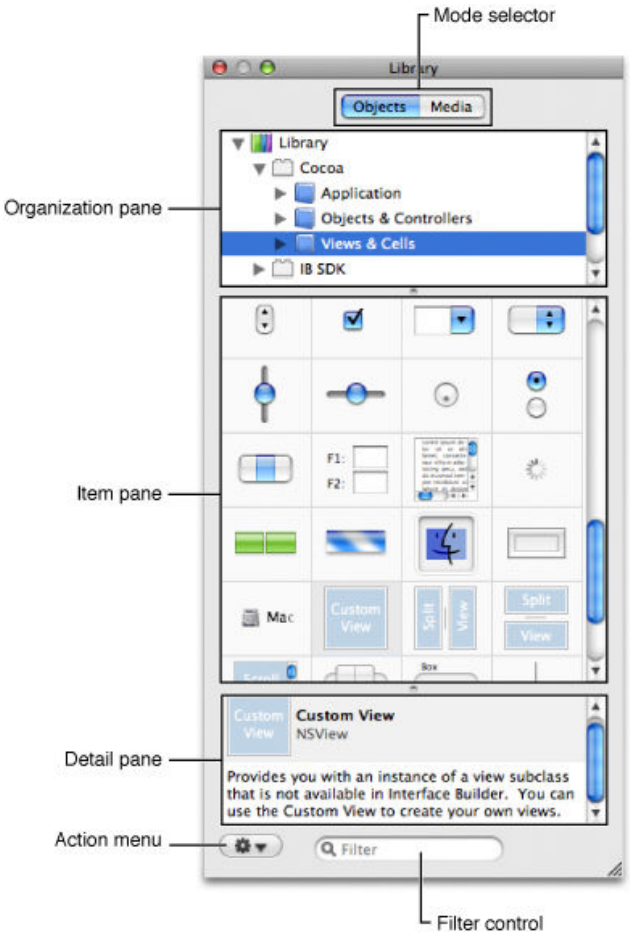

Figura 27: Biblioteca de componentes.

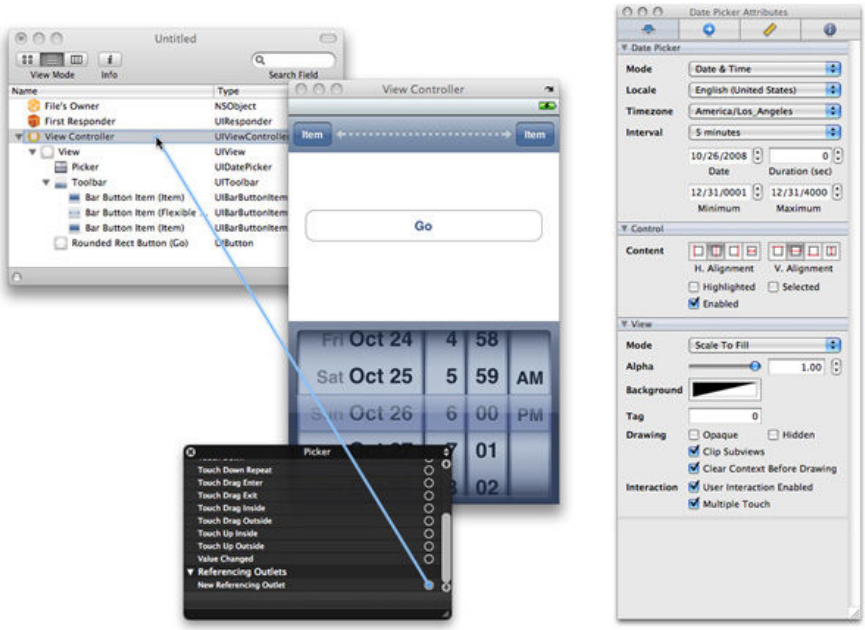

Figura 28: Parte da interface do *Interface Builder*.

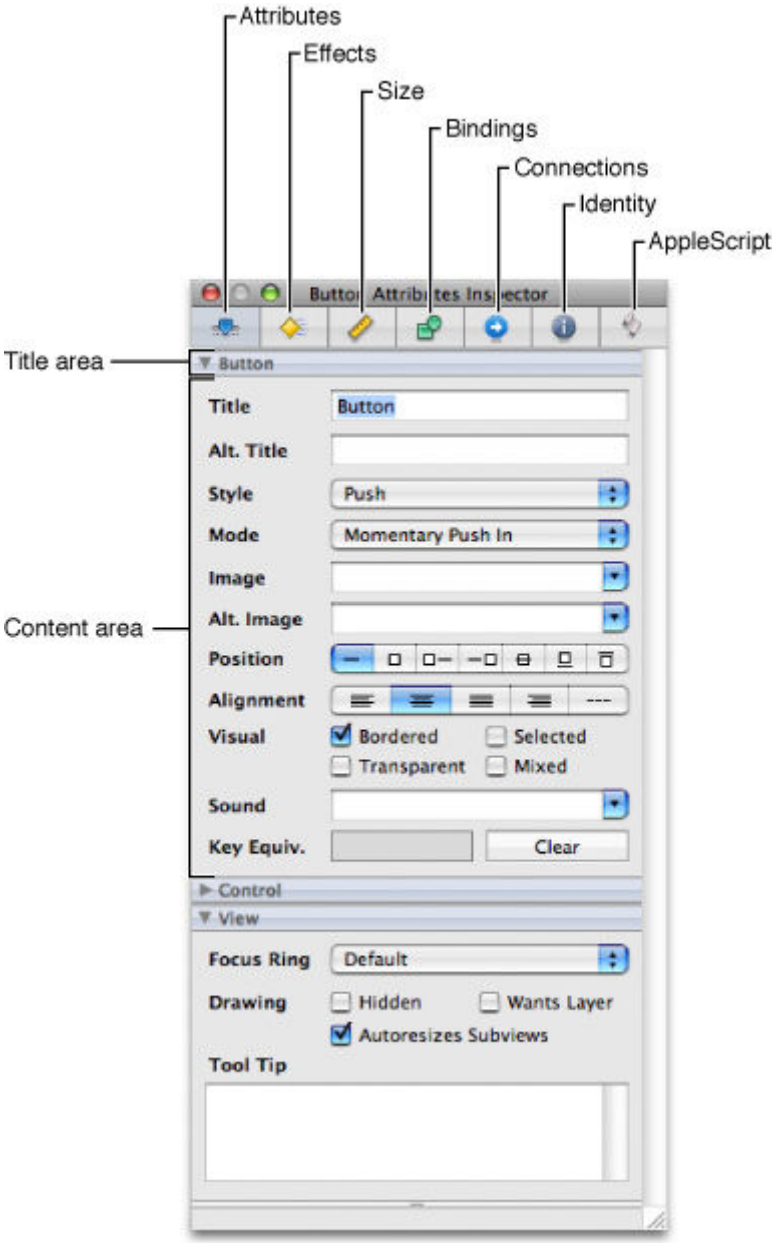

Figura 29: Aba de propriedade de um componente.

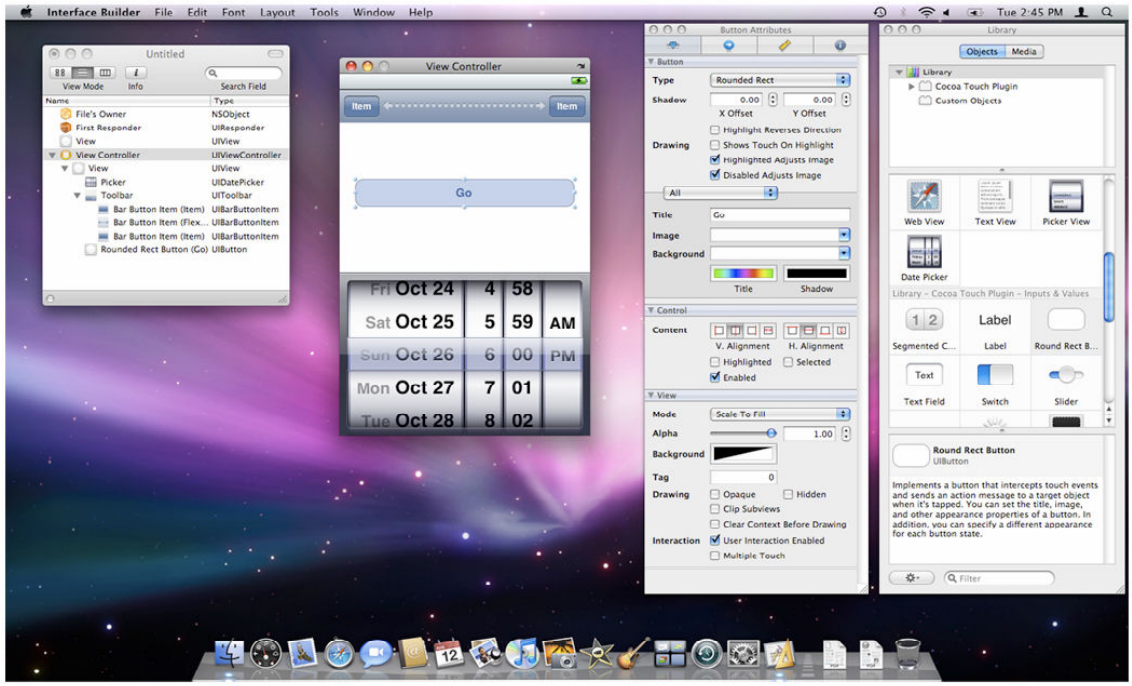

Figura 30: Desktop do MacOS X com o *Interface Builder* aberto.

3.5.5- Instruments

Instruments mostra o consumo de memória e monitora o sistema. Ela ajuda a identificar e identificar áreas criticas na aplicação e assim trabalhar com mais eficiência. *Instruments* oferece graficamente uma linha de performance que mostra onde a aplicação esta usando mais recursos do sistema. Essa ferramenta é construída em torno da DTrace de código aberto desenvolvida pela Sun Microsystems. *Instruments* desempenha um papel fundamental no gerenciamento de memória e fazendo assim as aplicações rodarem de forma eficiente no iPhone.

A figura 31 apresenta o funcionamento do *Instruments*

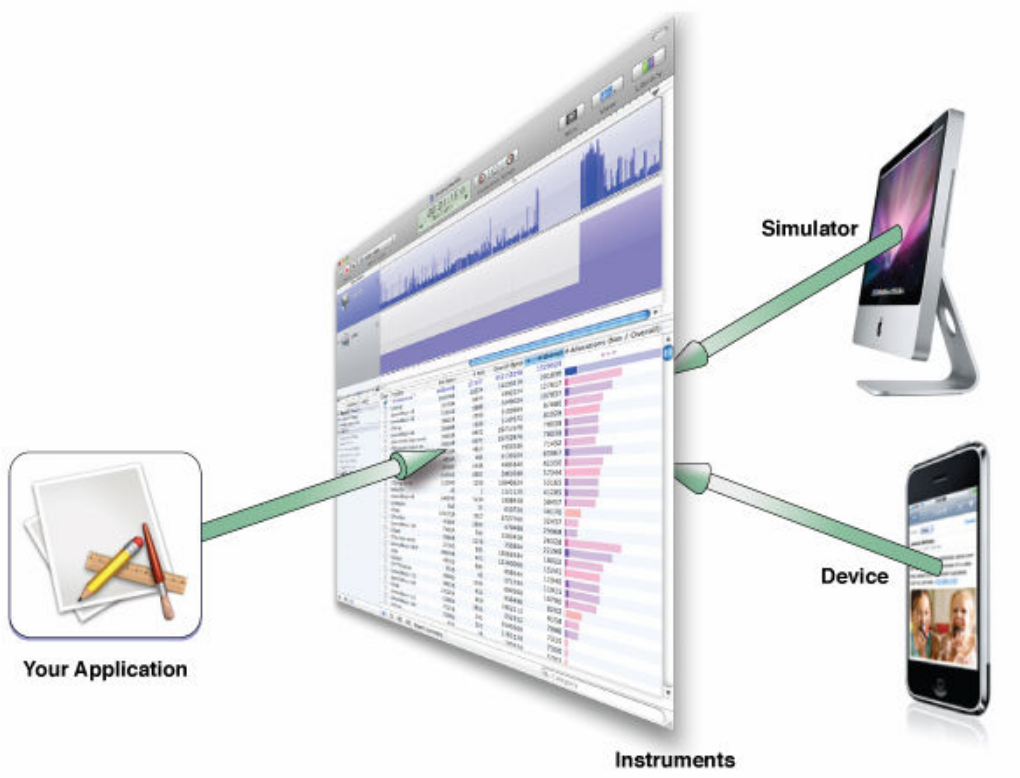

Figura 31: Fluxo do *Instruments*.

A tela do *Instruments* aberto e funcionando, é mostrada na figura 32.

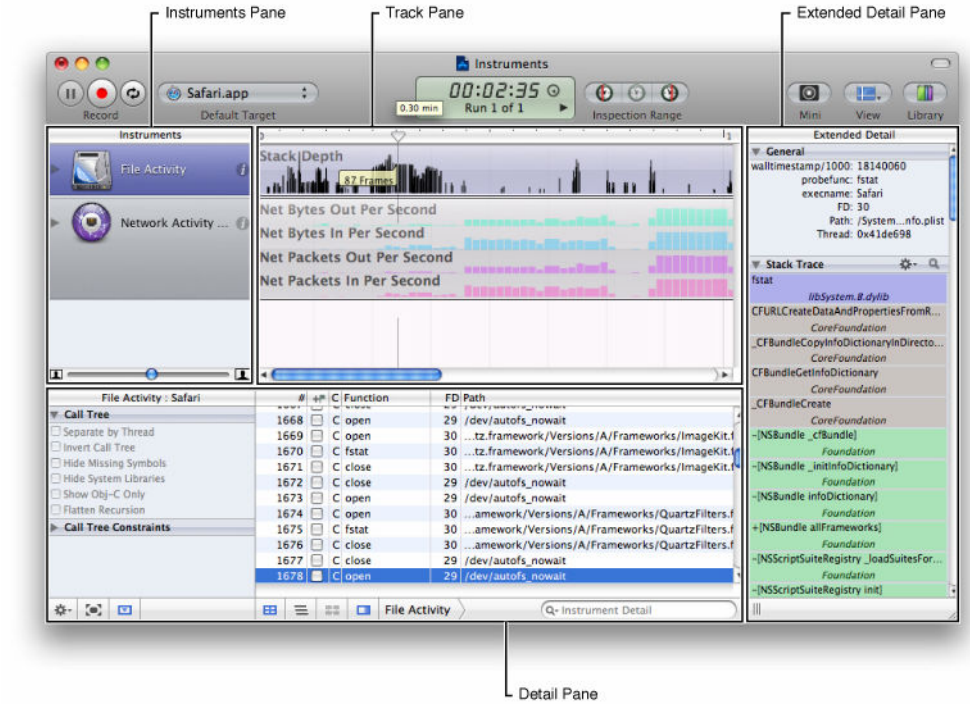

Figura 32: Tela do *Instruments* aberto e funcionando.

3.5.6- Shark

Shark é uma ferramenta de performance e otimização. Shark permite ver as entradas do sistema para ajudar a analisar a performance do código. Ela analisa o código enquanto ele está sendo executado para ver onde o tempo está sendo desperdiçado. Ela pode também dar sugestões de como melhorar o código. Shark é capaz de identificar "armadilhas" comuns de desempenho e visualmente apresentar os custos desses problemas.

3.5.7- Simulator

O iPhone Simulator roda no Macintosh e permite criar e testar aplicações no desktop. Também é possível fazer isso sem conexão com o iPhone. O Simulator oferece uma API usada no iPhone e disponibiliza uma pré-visualização de como ficará a aplicação. Quando se trabalha com o Simulator, o Xcode compila o código Intel X86 que roda nativamente no Macintosh e não códigos baseados em ARM que são usados no iPhone.

IPhone Simulator sendo executado com uma mensagem de boas vindas na tela (welcome) é mostrado na figura 33.

Figura 33: Tela do iPhone Simulator

CAPÍTULO 4

Engine Cocos2d

Neste capitulo, será feita uma descrição da Engine Cocos2d utilizada para o desenvolvimento do aplicativo, já que possui vários recursos para manipulação.

4.1- Cocos2d

O Cocos2d é um framework para construção de games 2D, versão demo, e outras aplicações gráficas e interativas escrito em python baseada em PyGlet e com isso, baseada em OpenGL [14]. Mesmo sendo um framework novo, cocos2d tem uma boa documentação. Formada por um guia de programação, uma referencia da API e um FAQ. Os principais recursos existentes no Cocos2d são:

- **Fluxo de Controle**: gerencia o controle do fluxo entre diferentes cenas de uma maneira simples e realiza as transições entre cenas;
- **Sprite**: manipulação de sprites de uma maneira rápida e fácil;
- **Ações**: ações de transformações como mover, rotacionar, escalonar, pular, etc. e as ações compostas como seqüência de ações, ações reversas, repetição de ações e/ou seqüências;
- **Efeitos**: efeitos como ondas, rotação, "lents" e muito mais;
- **Mapa de Titles**: suporta mapa de tiles retangular e hexagonal;
- **Menus**: menu e botões básicos;
- **Integração**: com as engines de física *Box2D* e *Chipmunk*;
- **Sistema de Partículas**;
- **Suporte**: a renderização de textura, ao acelerômetro, touch e ao som;
- **Documentação**: guia de programação, API referencia, tutoriais em vídeo e muitos testes simples mostrando como usa-la;
- **Baseado em OpenGL ES 1.1**: aceleração a nível de hardware;
- **Open Source**: compatível com códigos de projetos aberto e fechados;

4.1.1- Familias Cocos 2d

Existem várias famílias Cocos2d, e serão feitas uma breve descrição dessas famílias.

- **Cocos2d:**este é o projeto que começou a família cocos2d.
  - Linguagem: python
  - Plataformas: Mac, Linux, Windows
  - Pagina na internet: http://www.cocos2d.org
- **Cocos2d para iPhone, cocos2d em Objective-C**
  - Linguagem: Objective-C
  - Plataforma: iPhone
  - Pagina na internet: http://www.cocos2d-iphone.org
- **ShinyCocos**: um pacote da cocos2d para iPhone em Ruby.
  - Linguagem: Ruby
  - Plataforma: iPhone
  - Página na internet: http://www.cocos2diphone.org/wiki/doku.php/shinycocos:faq
- **Cocos2d para Android, cocos2d em Java**
  - Linguagem: Java
  - Plataforma: Android
  - Página na internet: http://code.google.com/p/cocos2d-android/
- **CocosNet, cocos2d em Mono**
  - Linguagem: C# / Mono
  - Plataforma: iPhone
  - Página na internet: http://github.com/city41/CocosNet

4.1.2- Conceitos básicos

Existem alguns conceitos básicos para se desenvolver uma aplicação usando cocos2d: cenas, diretor (director), camadas (layers) e sprites [15].

4.1.2.1- Cenas

Uma cena é mais ou menos uma peça independente do fluxo da aplicação. A aplicação pode ter varias cenas, mas apenas uma estará ativa em um

determinado momento. Uma cena é composta de uma ou mais camadas (layers). Camadas dão a cena aparecia e comportamento a mesma.
Uma cena pode ser uma transição entre cenas, tal como uma passagem de uma fase para outra, uma passagem do menu principal para o começo da fase. Essas transições podem ser: fadeIn/fadeOut, correr a cena para o canto e colocar a nova cena na tela, etc. A implementação de uma cena utilizando cocos2d é relativamente simples, como o mostra o exemplo. Os passos para para obter a cena são: adicionar primeiro a cocos2d na janela principal do iPhone, depois criar uma cena vazia, e tornar a janela principal visível e ao final "mandar" rodar a cena.

```
UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
   mainScreen] bounds]];

// anexa cocos2d na janela
[[Director sharedDirector] attachInWindow:window];
//Cria uma cena vazia
   Scene *scene = [Scene node];
   //Torna ajanela principal "window" visivel
   [window makeKeyAndVisible];
   // Roda a cena
   [[Director sharedDirector] runWithScene: scene];
```

4.1.2.2- Diretor

O Diretor é o objeto que controlara o fluxo da aplicação, apenas um diretor pode ser declarado, ele segue o padrão *Singleton*. Ele vai cuidar de ir uma cena para frente ou para trás, mudar de uma cena para outra. O Diretor também é responsável por inicializar a tela princinal (*Main Window*).

4.1.2.3- Camadas

Uma camada tem o tamanho de toda a área de desenho da tela. Ela pode ser semi-transparente, totalmente transparente ou parcialmente transparente em algumas partes dela, que permitem ver as outras camadas por trás deles. Os Layers são os que definem a aparência e comportamento, assim a maioria do seu tempo de programação será gasto codificação subclasses *Layer* que fazer o que você precisa. A figura 34 mostra uma cena de Menu, onde se tem uma cena e três camadas adicionadas como filhas da cena.

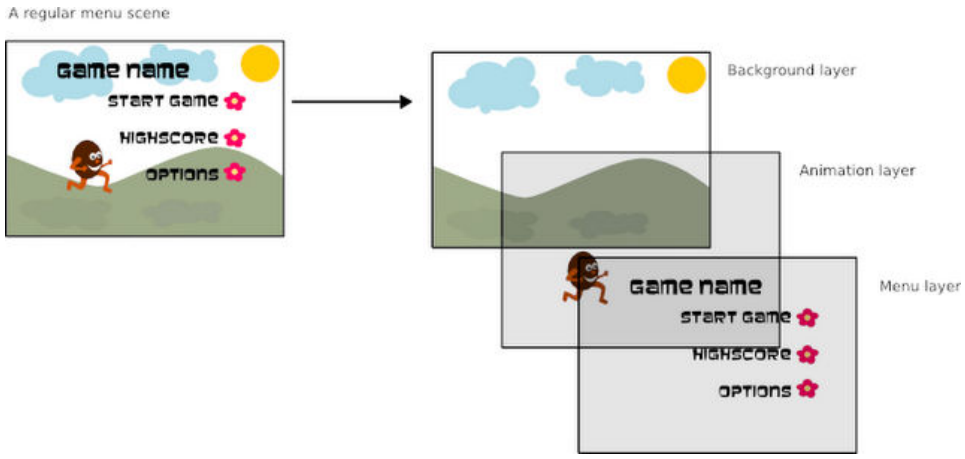

Figura 34: Exemplo de um Menu. Fonte: [16].

A camada é onde se define manipuladores de eventos. Os eventos são propagados para as camadas (de frente para trás), até que alguma camada pega o evento e aceita. As camadas podem conter Sprites, Labels e outras camadas filhas. O exemplo mostra como adicionar uma camada em uma cena.

```
UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
   mainScreen] bounds]];

// anexa cocos2d na janela
[[Director sharedDirector] attachInWindow:window];
//Cria uma cena vazia
   Scene *scene = [Scene node];
   //Torna ajanela principal "window" visivel

   // Cria e inicializa uma Camada
   Layer *layer = [Layer node];
   // adiciona a Camada como filha da cena princiapal
   [scene addChild:layer];

   [window makeKeyAndVisible];
   // Roda a cena
   [[Director sharedDirector] runWithScene: scene];
```

Seguindo o exemplo de como criar e inicializar uma cena, será acrescentada uma camada na cena. Os passos para acrescentar uma camada na cena são: criar primeiro a camada e depois adicionar a camada na cena principal, assim quando rodar a cena a camada como é filha da cena também será desenhada na tela.

4.1.2.4- Sprites

Um Sprites são objetos gráficos 2D, normalmente imagens carregadas de um arquivo ou mesmo objetos desenhados por vetores. Os Sprites pode ser animados, intercalando imagens para gerar a noção de movimento, podem ser rotacionados, movidos e escalonados. Sprite pode ter outro sprite como filho, ou seja, qualquer transformação feita no sprite pai, será automaticamente transferida ao sprite filho. O exemplo mostra como adicionar um Sprite em uma camada.

```
UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
    mainScreen] bounds]];

// anexa cocos2d na janela
[[Director sharedDirector] attachInWindow:window];
//Cria uma cena vazia
    Scene *scene = [Scene node];
    //Torna ajanela principal "window" visivel

    // Cria e inicializa uma Camada
    Layer *layer = [Layer node];
    // adiciona a Camada como filha da cena princiapal
    [scene addChild:layer];

//Cria um objeto sprite e o coloca na posição 0, 50 da tela
Sprite *sprite = [Sprite spriteWithFile:@"imgSprite.png"];
sprite.position = ccp( 0, 50);

//Adicona o sprite como filho da layer
[layer addChild:sprite z:0];

    [window makeKeyAndVisible];
    // Roda a cena
    [[Director sharedDirector] runWithScene: scene];
```

Mais uma vez foi usado o exemplo anterior, onde foram acrescentadas apenas três linhas de código para adicionar um Sprite à camada e consequentemente à cena também. Os Sprites são objetos que podem ser animados. O exemplo mostra como implementar uma animação nesses Sprites utilizando as ações da cocos2d. Criar um sprite a partir de uma imagem de arquivo, configura a posição inicial do sprite como o ponto x: 0 e y: 50 da tela.

```
// Cria um açao de RotateBy (rotacionar).
// "By" significa relativo. "To" significa absoluto.
id rotateAction = [RotateBy actionWithDuration:4 angle:180*4];
```

```
//pergunta pelo tamanho da janela
CGSize size = [[Director sharedDirector] winSize];
// cria a ação JumpBy (pular).
id jumpAction = [JumpBy actionWithDuration:4
position:ccp(size.width,0) height:100 jumps:4];

// spawn é a geração de açoes a serem executadas ao mesmo tempo
id fordward = [Spawn actions:rotateAction, jumpAction, nil];

    // Quase todas as açoes suportam o metodo reverso.
// Cria a açao reversa.
    id backwards = [fordward reverse];

// Sequencia é a ação executas açoes seguidas uma da outra
id sequence = [Sequence actions: fordward, backwards, nil];

// Pode-se repetir as açoes quantas vezes quiser.
// POde-se repetir infinitamente usando a ação "RepeatForEver"
    id repeat = [Repeat actionWithAction:sequence times:2];

    // Diz para o Sprite executar as ações.
    [sprite runAction:repeat];
```

O exemplo mostra que podem ser configuradas varias ações a serem executadas. A primeira ação é a de rotacionar (RotateBy). Ela foi configurada para rotacionar quatro vezes com um ângulo de 180°, Isto significa que será executada quatro vezes esse calculo, ou seja, é o total que o Sprite rotacionará. A segunda é a ação de pular (JumpBy). Assim como a ação de rotacionar, ela foi configurada para ser executada quatro vezes, as posições finais de cada eixo (x = tamanho da janela e y = 0), a altura em pixel que o pulo terá e a quantidade de pulos. A ação de gerar ações ao mesmo tempo (Spawn), tem de ser configurada para gerar a rotação e o pulo ao mesmo tempo. Também é gerada uma ação que será a inversa da ação Spawn configurado pelo método *Reverse* e configurado uma seqüência de ações (Sequence) que será as ações de rotação pulo e depois o inverso delas. Agora é configurado a repetição dessas ações (Repeat) por duas vezes, podendo repetir as ações infinitamente usando a ação "RepeatForEver". Finalmente, executa as ações configuradas com o método *runAction* da classe Sprite. As ações básicas são:

- **Position:** MoveBy, MoveTO, JumpBy, JumpTo e Place;
- **Scale:** ScaleBy e ScaleTo;
- **Rotation:** RotateBy e RotateTo
- **Visible:** Show, Hide, Blink e ToggleVisibility;
- **Opacity:** FadeIn, FadeOut e FadeTo,
- **R,G,B:** TintBy e TintTo.

4.2- Aplicação em Coco2d

Como a linguagem Objetive-C não é muito utilizada nos meios acadêmicos, será mostrado o código de um exemplo "Hello Word" com a finalidade de comparar a sintaxe de programação com as outras linguagens e também para conhecer o funcinamento da engine [17].

Arquivo: HelloWord.h

```
// Importando esse cabeçalho importa todas as classes da cocos2d
#import "cocos2d.h"

@interface AppController : NSObject <UIAccelerometerDelegate,
UIAlertViewDelegate, UITextFieldDelegate, UIApplicationDelegate>
{
        //Janela principal
        UIWindow *window;
}

// Deixa a janela principal como uma propriedade
@property (nonatomic, retain) UIWindow *window;
@end

// Camada HelloWord
@interface HelloWorld : Layer
{
}
@end
```

Arquivo: HelloWord.m

```
// Necessario para UIWindow, NSAutoReleasePool, e outros objetos
#import <UIKit/UIKit.h>

// Importando a inteface
#import "HelloWorld.h"

// implementaçao HelloWorld
@implementation HelloWorld

// em init necessita-se inicializar as instancias
-(id) init
{
        // sempre chama "super" init

        if( (self=[super init] )) {

                // cria e inicializa a Label
                Label* label = [Label labelWithString:@"Hello World"
fontName:@"Marker Felt" fontSize:64];

                // pergunta ao Diretor o tamanho da janela
```

```
                CGSize size = [[Director sharedDirector] winSize];

                // posiciona a label no centro da janela
                // 'ccp' é uma macro que ajuda a crier um ponto.
Quer dizer um 'CoCos Point'
                label.position =  ccp( size.width /2 , size.height/2
);

                // adiciona a label como filho da camada
                [self addChild: label];
        }
        return self;
}

// em "dealloc" necessita-se desalocar os objetos
- (void) dealloc
{
        // nao esquecer de chamar o "super dealloc"
        [super dealloc];
}
@end

@implementation AppController

// window é uma propriendade. @synthesize irá crier os metodos para
acesso a mesma
@synthesize window;

// Funcao que é chamada sempre que for inicializado a aplicação
- (void) applicationDidFinishLaunching:(UIApplication*)application
{
        // cria e inicializa main UIWindow
        window = [[UIWindow alloc] initWithFrame:[[UIScreen
mainScreen] bounds]];

        // Anexa cocos2d na janela pricinal
        [[Director sharedDirector] attachInWindow:window];

        // antes de crier qualquer layer, seta o modo landscape
        [[Director sharedDirector]
setDeviceOrientation:CCDeviceOrientationLandscapeLeft];

        // Torna a janela visivel
        [window makeKeyAndVisible];

        // Cria e inicializa uma cena vazia
        Scene *scene = [Scene node];

        // Cria e inicializa nossa Camada HelloWorld
        Layer *layer = [HelloWorld node];
        // Adiciona nossa camada como filha da janela principal
        [scene addChild:layer];

        // Roda!
        [[Director sharedDirector] runWithScene: scene];
}


- (void) dealloc
{
        [window release];
```

```
        [super dealloc];
}

@end


int main(int argc, char *argv[]) {
                NSAutoreleasePool *pool = [NSAutoreleasePool new];
        UIApplicationMain(argc, argv, nil, @"AppController");
        [pool release];
        return 0;
}
```

CAPÍTULO 5

Desenvolvimento do Game

5.1- Definição do problema

O problema consiste no desenvolvimento de um game que usará recursos do iPhone tais como o acelerômetro. O usuário (jogador) terá que interagir com o aparelho iPhone rotacionando e tocando o mesmo para poder manipular as ações do game.

5.2- Modelagem do problema

O game terá uma interface gráfica 2D e consiste em apenas uma tela, onde terá uma moldura fechada como se fosse um labirinto sem saída, terá uma diferente para cada fase, e dentro estarão dispostos blocos de diferente cores e tipos, separados um do outro. Haverá três blocos de cada tipo. Esses blocos se movem na tela obedecendo a determinadas regras lógicas.

O objetivo do jogo é juntar os três blocos de mesmo tipo para eliminá-los ficando assim sem nenhum bloco na tela.

Os blocos sofrerão ação da gravidade (o iPhone terá que estar "em pé"), então para movê-los o jogador terá que girar o iPhone (90 ou 180 graus) alterando a disposição do blocos em relação a moldura. Haverá apenas quatro posições para a moldura: 0, 90, 180 e 270 graus. Os blocos se movimentarão apenas na vertical, e acompanharão a mesma coluna que eles estiverem, ou seja, não se movimentarão na diagonal e nem na horizontal, pois para o movimento dos blocos a posição do iPhone tem que ser mudada, logo sempre o movimento se dará na vertical. Será um jogo de raciocínio lógico e haverá um limite de tempo para solucionar o desafio.

5.3- Desenvolvimento do game

Após os estudos realizados da linguagem Objective-C, do SDK do iPhone e da Engine cocos2d, parte-se para o desenvolvimento do game que juntará todas as tecnologias estudadas e o conhecimento adquirido.

5.3.1- Design do Game

Primeiramente no desenvolvimento do game foi feita a parte de design, que consiste em criar as imagens que constituirão a interface do game, ou seja, desenhar os blocos, a interface principal do jogo e a moldura de cada fase. A figura 35 mostra os blocos que compõem o jogo.

Figura 35: Blocos que compõem o jogo.

A figura 36 mostra a moldura de uma das fases do game.

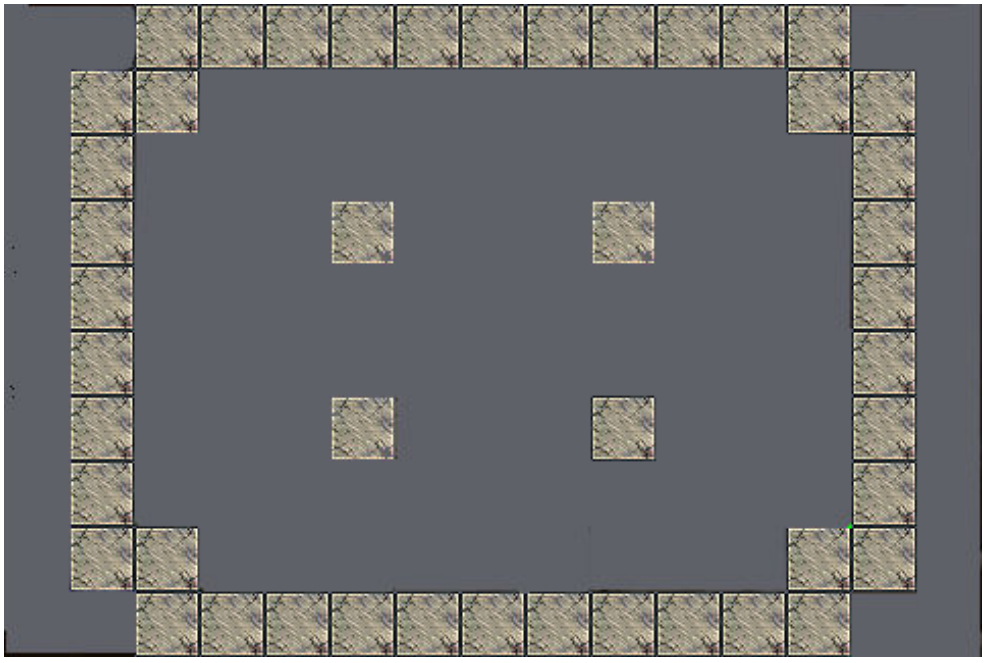

Figura 36: Exemplo de moldura do jogo.

Para o desenvolvimento das molduras e dos blocos que compõem o game foi utilizado os seguintes softwares de edição de imagem (Photoshop) e de desenho vetorial (CorelDraw) para desenhá-las e editá-la.

5.3.2- Geração de imagens

Para a geração das imagens na tela, ou seja, o desenho das imagens na tela do iPhone foi utilizada a Engine cocos2d que utiliza OpenGL ES. Será apresentado um trecho do código fonte que tem a função de acrescentar um Sprite na tela do iPhone.

```
Scene *scene = [Scene node];

Layer *layer = [Layer node];
[scene addChild:layer];

Sprite *sprite = [Sprite spriteWithFile:@"imgSprite.png"];
sprite.position = ccp( 0, 50);

[Layer addChild:sprite z:0];
```

Esse trecho de código adiciona um Sprite na posição x: 0 e y: 50 na tela do iPhone. Uma observação importante é que sempre será possível ver o que se passa dentro de uma cena, ou seja, sempre vai ter uma cena ativa. Outra coisa importante é que dentro da cena estará uma ou mais camadas, e dentro dessa(s) camada(s) serão adicionadas sprites, labels, tiles, etc. Sendo assim para se desenvolver tanto um Menu quanto uma fase foi usado esse fluxo.

5.3.3- Movimento dos blocos

Como os blocos sofrerão ação da gravidade, dependendo da posição do iPhone, foi necessário criar um mecanismo para tratar este tipo de força e o movimento dos blocos, que é realizado sobre a moldura.

A cocos2d tem suporte às engines físicas *Box2d* e *Chipmunk* para colisão e gravidade. De acordo com a rotação do iPhone, é necessário verificar em que posição o iPhone se encontra e assim mudar o valor (direção) da gravidade. Este mecanismo faz os blocos deslizarem em direção a força da gravidade. O controle do acelerômetro é feito através de um c*allBack* que é chamando sempre que alguma alteração é detectada no mesmo. Sabendo a posição inicial do iPhone, será feita a captura da rotação do aparelho utilizando o acelerômetro, e com isso será verificado o grau da

rotação feita, e dependendo desse grau é necessário mudar a direção da gravidade nas engines físicas. Será apresentado um trecho do código fonte em que foi utilizado o callback.

```
- (void)accelerometer:(UIAccelerometer*)accelerometer
didAccelerate:(UIAcceleration*)acceleration
{
      Sprite *sprite = [Sprite spriteWithFile:@"imgSprite.png"];


      // update a rotação baseado no rotacão em Z
      // o Sprite sempre estara "em pé"
      sprite.rotation = (float) CC_RADIANS_TO_DEGREES(
      atan2f((float)acceleration.x, (float)-acceleration.y
      ) + M_PI );

}
```

No trecho do código fonte apresentado, sempre que for detectado um movimento pelo acelerômetro este método será executado e de acordo com o código, um sprite será rotacionado ficando sempre "em pé" na tela, independente da posição do iPhone [18].

O tratamento das colisões de blocos com blocos e blocos com a moldura é designado a uma das duas engines físicas. Será apresentado um trecho do código fonte onde é feito o tratamento da colisão de uma bola e o "chão" do iPhone.

```
-(id)init{

      layer = [super init];

      ballSprite = [Sprite spriteWithFile:@"ball.png"];

      [ballSprite setPosition:CGPointMake(150, 400)];

      [layer add:ballSprite];

      [layer setupChipmunk];

}

-(void)setupChipmunk{

      cpInitChipmunk();

      space = cpSpaceNew();
```

```
	space->gravity = cpv(0,-2000);

	space->elasticIterations = 1;

	[self schedule: @selector(tick:) interval: 1.0f/60.0f];

	cpBody* ballBody = cpBodyNew(200.0, INFINITY);

	ballBody->p = cpv(150, 400);

	cpSpaceAddBody(space, ballBody);

	cpShape* ballShape = cpCircleShapeNew(ballBody, 20.0, cpvzero);

	ballShape->e = 0.8;

	ballShape->u = 0.8;

	ballShape->data = ballSprite;

	ballShape->collision_type = 1;

	cpSpaceAddShape(space, ballShape);

	cpBody* floorBody = cpBodyNew(INFINITY, INFINITY);

	floorBody->p = cpv(0, 0);

	cpShape* floorShape = cpSegmentShapeNew(floorBody, cpv(0,0),
	cpv(320,0), 0);

	floorShape->e = 0.5;

	floorShape->u = 0.1;

	floorShape->collision_type = 0;

	cpSpaceAddStaticShape(space, floorShape);

}


-(void)tick:(ccTime)dt{

	cpSpaceStep(space, 1.0f/60.0f);

	cpSpaceHashEach(space->activeShapes, &updateShape, nil);

}


void updateShape(void* ptr, void* unused){
```

```
        cpShape* shape = (cpShape*)ptr;
        Sprite* sprite = shape->data;
        if(sprite){
              cpBody* body = shape->body;
              [sprite setPosition:cpv(body->p.x, body->p.y)];
        }
  }
```

Foi definido e inicializado um objeto Sprite (ballSprite) no método *init*. Ainda no método *init* é feito a chamada do método *setupChipmunk*. Neste método é criado um espaço chipmunk atribui alguns parâmetros. Foi utilizado o método *schedule:interval* para programar o *seletor tick* com um intervalo, esse método (*tick*) vai chamar o método *cpShapeStep* para atualizar o espaço, e também vai chamar o método *cpSpaceHashEach*, que vai chamar *updateShape* para cada shape ativo. Assim a posição do sprite (ballSprite) será atualizada.

Após, o corpo e o *shape* da bola (ballSprite) serem criados (*cpBody* ballBody = cpBodyNew(200.0, INFINITY);* e *cpShape* ballShape = cpCircleShapeNew(ballBody, 20.0, cpvzero);*), adiciona o *Shape* da bola no espaço ChipMunck. Depois pode se criar mais um corpo e um *shape* para o chão (*cpBody* floorBody = cpBodyNew(INFINITY, INFINITY);* e *cpShape* floorShape = cpSegmentShapeNew(floorBody, cpv(0,0), cpv(320,0));*). Como o chão não muda e nem se mexe é usado o método *cpShapeAddStaticShape* para adicionar o chão ao espaço [19]. Utilizando essas técnicas foi feito o tratamento da detecção do movimento do acelerômetro com relação a gravidade e movimento dos blocos. As figuras 37 e 38 mostram como são feitos as movimentação dos blocos.

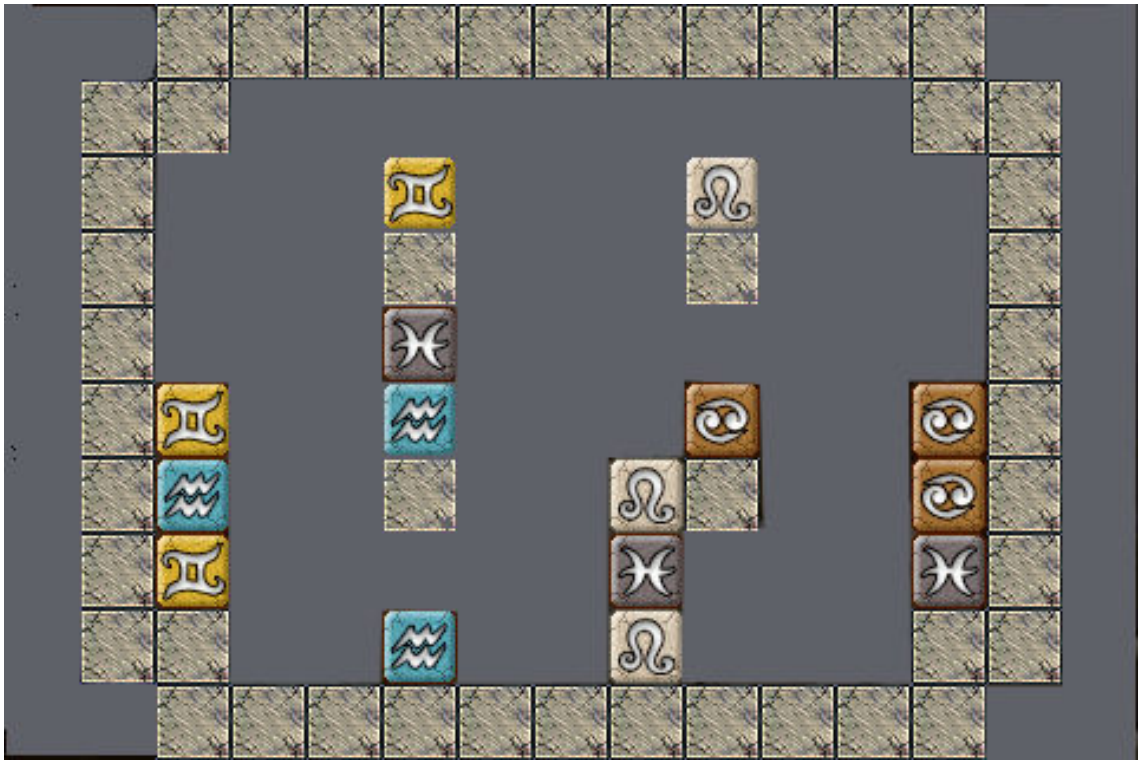

Figura 37: Movimentação dos blocos 1

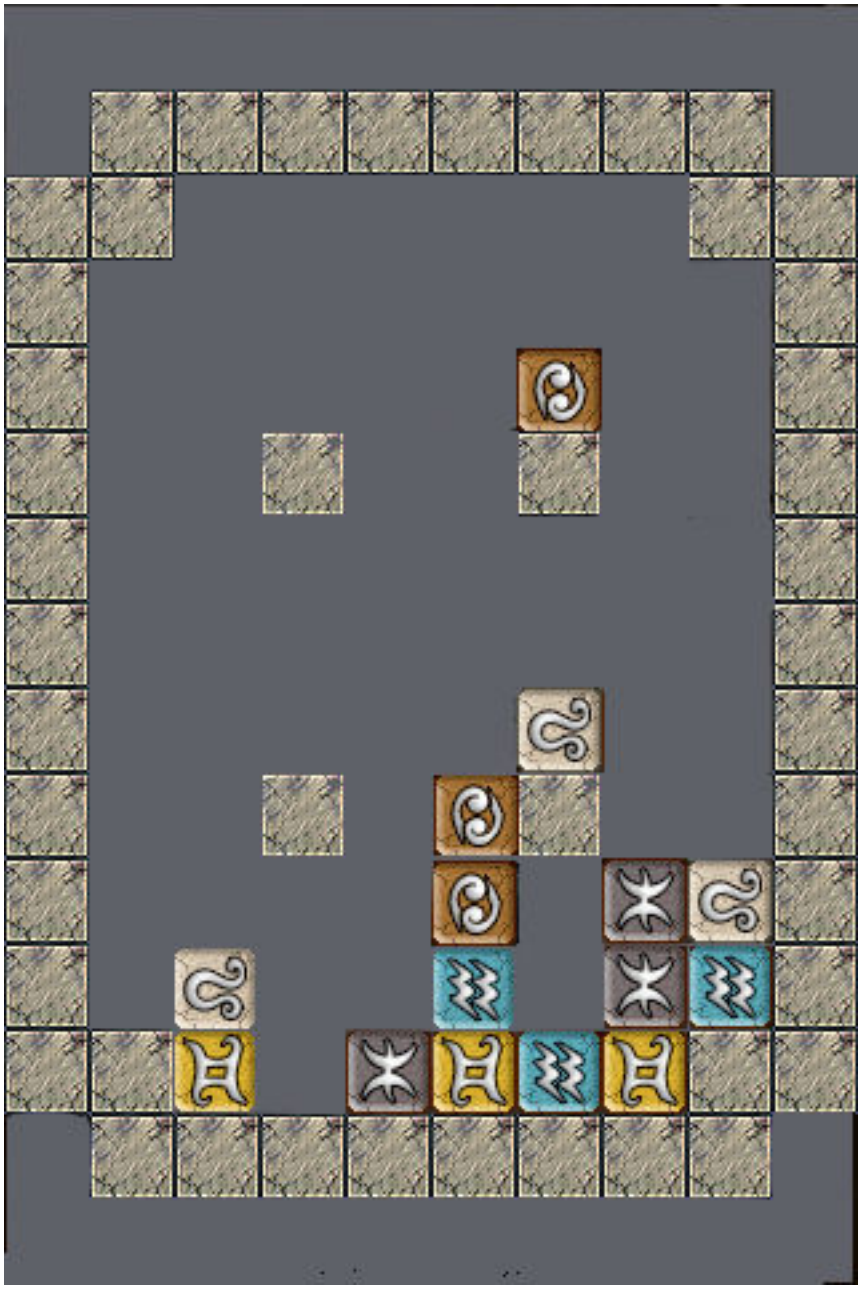

Figura 38: Movimentação dos blocos 2.

Neste caso, os movimentos do iPhone foram rotacionados com um ângulo de 90 graus a esquerda, mudando assim a força gravitacional exercida sobre os blocos, acarretando no movimento mostrado nas figuras 37 e 38. Com o movimento feito, nenhum bloco terá que ser eliminado, pois não há nenhum conjunto de três blocos de mesmo tipo.

5.3.4- Fluxo do Game

Dando seqüência ao desenvolvimento das ferramentas necessárias e dos módulos do game, foi feita a programação para verificar se há três blocos de mesmo tipo juntos, para eliminar os mesmo. Foram necessários também desenvolver a programação responsável pela animação dos blocos sendo eliminados e a verificação do fim da fase, ou seja, o momento em que todos os blocos forem eliminados. A figura 39 mostra as maneiras possíveis para que três blocos do mesmo tipo possam ser eliminados.

Figura 39: Formas para se eliminar os blocos.

Na classe "Bloco" existe um método (checkBlockTo) que verifica se há condições para eliminação dos mesmo, este método verifica os vizinhos de cada bloco e insere em um vetor (vetorDestricao), com a condição de que se esse vetor for preenchido com três blocos de mesmo tipo, ele chama o método responsável pela eliminação dos mesmo.

5.3.5- Fase do Game

Esta fase é uma das mais importantes, pois é o momento de juntar e agregar todas as funcionalidades desenvolvidas e testadas para montar as fases do game. A figura 40 mostra o fluxo que o game deve seguir.

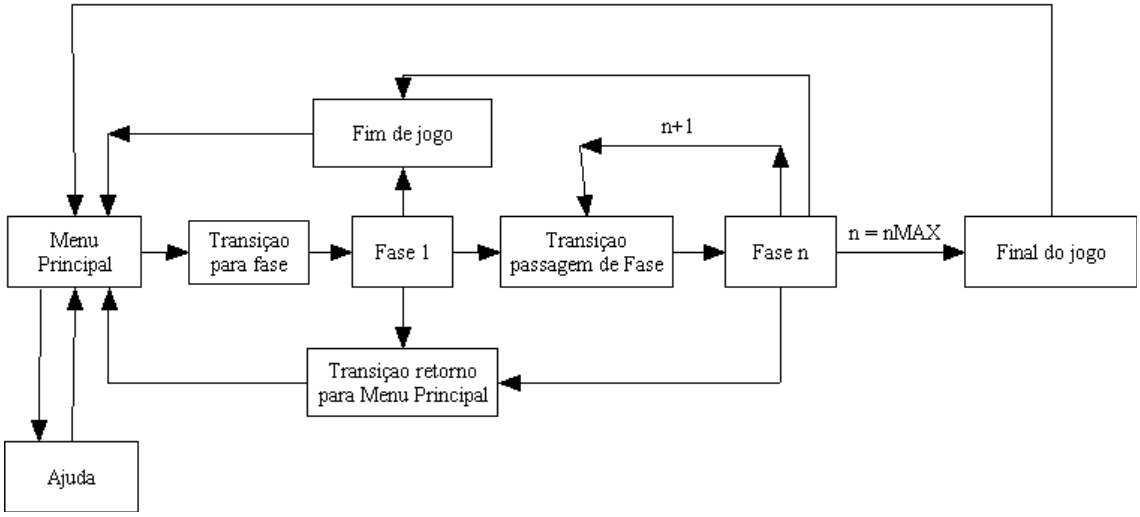

Figura 40: Fluxo de transição do game.

Cada item compõe uma cena, e tudo começa na cena "Menu Principal", logo após ou o game vai para a cena "ajuda" ou para a cena "fase 1" com uma transição entre as cenas. Se não concluída a fase o fluxo do game vai para o "fim do jogo" caso contrário vai para a fase seguinte, se a fase for a ultima então a cena final do jogo é apresentada. Para todas essas cenas há uma ou mais camadas, todas implementadas usando a engine cocos2d.

CAPÍTULO 6

Conclusão

A construção de games sempre envolverá novas tecnologias tanto em programação quanto em hardware. Neste trabalho foi abordado tecnologias novas, uma vez que a linguagem Objetive-C apesar de não ser nova, não é muito difundida entre os programadores, alunos e professores. O hardware explorado esta capacitado de novas tecnologias exploradas neste game, como o acelerômetro e o multi-touch.

A interação que o iPhone proporciona ao usuário, tanto pela manipulação das ações do smartPhone com o toque dos dedos, quanto o acelerômetro que é capaz de saber em que posição o aparelho se encontra, proporciona ao jogador uma maior proximidade ao mundo virtual levando o assim a uma maior diversão.

A proposta de aprender novas tecnologias, principalmente àquelas pouco difundidas no meio acadêmico foi desafiador. Foram encontradas algumas dificuldades no início, mas ao final deste projeto de pesquisa é possível dizer que foi bastante satisfatório. No futuro, estes conhecimentos adquiridos serão utilizados profissionalmente.

Uma contribuição importante resultante desta pesquisa é incentivar outras pessoas a desenvolver aplicativos com estas tecnologias. A documentação foi elaborada de maneira a conter as fundamentações teóricas principais. Para quem quer atuar na área de game esta é uma tecnologia que vale a pena aprender.

CAPÍTULO 7

Referências Bibliográficas

[1] http://abclocal.go.com/kgo/story?section=news/business&id=4920783 acesso março de 2009.

[2] http://pcworld.uol.com.br/reportagens/2007/07/02/idgnoticia.2007-07-02.3840668397/ acesso março de 2009.

[3] Lamarche, J. e Mark, D. : *Beginning iPhone Development Exploring the iPhone SDK*, Apress, 2008.

[4] Zdziarski, J. : *iPhone SDK Application Development*, O'Reilly Media Inc, 2009.

[5] Allen, C. e Appelcline, S. : *iPhone in Action – Introduction to Web and SDK Development,* Manning, 2008.

[6] Kochan, S. G. : *Programming in Objective-C*, Sams Publishing, 2003.

[7] Wagner, R. : *iPhone and iPod Touch Programming – Building Applications for Mobile Safari*, Wrox, 2008.

[8] Sadun, E. : *The iPhone Developer's Cookbook – Building Applications with the iPhone SDK,* Developer's Library, 2008.

[9] Apple Computer, Inc.: *The Objective-C Programming Language for Mac OS X version 10.3 and later*

[10]http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//apple_ref/doc/uid/TP40007592 acesso junho de 2009.

[11] Hillegass, A.: *Cocoa Programming for Mac OS X*,2nd edition, 2007.

[12] http://webclaudio.wordpress.com/ acesso setembro de 2009.

[13] http://www.sykey.net/hackintosh/ acesso junho de 2009.

[14] Kochan Stephen G. : *Programming in Objective-C 2.0*, 2009

[15] http://cocos2d.org/ acesso outubro 2009.

[16] http://www.cocos2d-iphone.org/wiki/doku.php/prog_guide:basic_concepts acesso outubro de 2009.

[17] http://www.cocos2d-iphone.org/wiki/doku.php/prog_guide:hello_world acesso outubro de 2009.

[18] http://iphonedev.net/2009/05/10/cocos2d-exemple-bouncing-ball/ acesso em outubro de 2009.

[19] http://www.cocos2d-iphone.org/wiki/doku.php/prog_guide:hello_events acesso em outubro de 2009.