UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

IFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA DO IPHONE

MARCO ANTÔNIO CORRÊA

BLUMENAU
2009

2009/2-15

MARCO ANTÔNIO CORRÊA

IFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA DO IPHONE

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. Mauro Marcelo Mattos, Doutor - Orientador

BLUMENAU
2009

2009/2-15

IFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA DO IPHONE

Por

MARCO ANTÔNIO CORRÊA

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

Presidente:	Prof. Mauro Marcelo Mattos, Doutor – Orientador, FURB
Membro:	Prof. Paulo Rodacki Gomes, Doutor – FURB
Membro:	Prof. Adilson Vahldick, Mestre – FURB

Blumenau, 15 de dezembro de 2009

RESUMO

O *framework* Furbot utilizado em disciplinas de introdução a programação, busca facilitar o aprendizado de lógica e linguagens de programação, utilizando como incentivo o desenvolvimento de aplicações envolvendo jogos. Este trabalho tem como objetivo apresentar a conversão do *framework* Furbot desenvolvido na linguagem de programação Java para a linguagem Objective-C, tornando possível sua utilização no desenvolvimento de aplicações para iPhone. São utilizados recursos como acelerômetro e *multi-touch*, disponíveis no iPhone, para atender as funcionalidades já existentes na versão do Furbot em Java. Como resultado desse desenvolvimento, tem-se uma versão do Furbot contendo as funcionalidades já oferecidas anteriormente na versão em Java além de um template para facilitar o desenvolvimento de aplicações com o *framework*.

Palavras-chave: Furbot. Objective-C. iPhone.

ABSTRACT

The framework Furbot is a tool to help students learning programming languages, as an incentive to develop applications involving games. This work aims to present the conversion the Furbot framework developed in Java programming language to Objective-C language, making possible its use in developing applications for iPhone. Resources as accelerometer and multi-touch, available on the iPhone, are used to complement some existing features in the Java version of Furbot. As a result of this development, we have a version of Furbot containing all the features already offered in the Java version and a template to help the development process of new applications using the framework.

Key-words: Furbot. Objective-C. iPhone.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

IDE - *Integrated Development Environment*

MVC – *Model View Controller*

RDF – *Resource Description Framework*

SDK – *Software Development Kit*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A cada ano novos alunos entram na universidade com o objetivo de aprender a programar e evoluírem profissionalmente. O aluno dá os primeiros passos na área através da disciplina de introdução a programação, a qual exige que o mesmo aprenda sintaxe, projeto, lógica e depuração (PHELPS; EGERT; BAYLISS, 2009, p. 4-5). Segundo Kumar (2003 apud GONDIM; AMBRÓSIO, 2008, p. 1), "problemas com esta disciplina são apontados como responsáveis pelo grande número de reprovações e desistências em cursos de Computação".

Buscando diminuir o número de reprovações e desistências, e também de instigar o aluno em desafios relacionados a algoritmos, são criados *frameworks* para facilitar o ensino. Os professores Adilson Vahldick e Mauro Marcelo Mattos criaram o Furbot que possui como objetivo principal diminuir as dificuldades de aprendizagem e ensino na lógica de programação através de um forte apelo à área de jogos, criando assim uma atmosfera facilitadora ao aprendizado (VAHLDICK; MATTOS, 2009, p. 1).

Este *framework* foi criado devido à experiência destes professores com alunos que não se sentiam motivados em resolver exercícios com enunciados como "digite cinco nomes e notas de alunos e mostre a média da sala, a maior e a menor nota" (VAHLDICK; MATTOS, 2009, p. 1). De acordo com Vahldick e Mattos (2009, p. 1), "o elemento central do Furbot é a programação de um robô que vive num mundo bidimensional juntamente com outros tipos de objetos, que também podem ser programados".

Um dos pré-requisitos para o desenvolvimento do Furbot é de que o mesmo fosse desenvolvido usando a tecnologia Java, pois essa era a linguagem adotada nos cursos onde o ambiente seria aplicado (VAHLDICK; MATTOS, 2009, p. 3). Apesar de Java oferecer portabilidade entre diversos sistemas operacionais e até mesmo ser funcional em dispositivos móveis, existem dispositivos que não são compatíveis com a linguagem.

Já é sabido que os dispositivos móveis, em especial os celulares, tem se tornado parte integrante da vida dos jovens. Cada dia com mais funcionalidades e serviços, o celular pode ser considerado como um facilitador de acesso a conteúdo, a qualquer hora, em qualquer lugar. Segundo Menezes (2009, p. 37) "a tecnologia móvel já pertence às boas práticas inovadoras de alguns docentes e tem demonstrado promover a aprendizagem e algumas das competências que os jovens de hoje necessitarão para competir e cooperar no século XXI".

Com o advento do iPhone e a constante utilização do mesmo para desenvolvimento de aplicações e jogos, percebeu-se uma oportunidade de testar o *framework* Furbot neste

dispositivo.

O iPhone, se comparado com outros dispositivos móveis, como por exemplo o N95 da Nokia, destaca-se por possuir uma arquitetura rica em componentes de interface, além de alta capacidade computacional e um eficiente sistema de distribuição através da App Store (MILUZZO et al., 2008, p. 41).

A plataforma do iPhone ainda fornece outras vantagens como a utilização de uma linguagem considerada de fácil aprendizado e utilização além de possuir várias *Application Programming Interface* (API) bem desenhadas e documentadas, permitindo ao desenvolvedor a abstração de componentes de baixo nível (MILUZZO et al., 2008, p. 42).

Levando em consideração os fatores mencionados acima, este trabalho tem o objetivo de converter o *framework* Furbot desenvolvido na linguagem de programação Java, para a linguagem de programação Objective-C, tornando possível sua utilização no ensino dessa linguagem.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar o *framework* do Furbot para a linguagem Objective-C, tornando possível a sua utilização na construção de aplicações na plataforma iPhone.

Os objetivos específicos do trabalho são:

a) converter o *framework* do Furbot para a linguagem Objective-C;
b) realizar o mapeamento de funcionalidades baseadas em interação via teclado do Furbot para interação via interface *multi-touch* e acelerômetro do iPhone;
c) permitir a criação de aplicações para o iPhone utilizando o *framework* e alguns conhecimentos básicos de programação.

1.2 ESTRUTURA DO TRABALHO

No primeiro capítulo é apresentado a introdução do trabalho, o objetivo principal, os objetivos específicos e uma descrição sobre a estrutura do trabalho.

O segundo capítulo contempla a fundamentação teórica do trabalho onde são descritos alguns recursos oferecidos pela linguagem de programação Objective-C comparando-a com a linguagem Java. Ainda, são apresentadas as ferramentas utilizadas no desenvolvimento, os trabalhos correlatos e é descrito o *framework* Furbot com informações sobre as suas funcionalidades e os seus principais conceitos.

No terceiro capítulo é descrita a implementação do Furbot em Objective-C, a apresentação de um roteiro para uso do *framework* e os resultados obtidos e discussão.

As considerações finais e extensões para trabalhos futuros são apresentados no quarto capítulo.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são apresentadas as ferramentas e conceitos utilizados no decorrer do texto. A seção 2.1 apresenta os conceitos de Objective-C e realiza uma análise comparativa com Java. A seção 2.2 apresenta a arquitetura do iPhone OS, a seção 2.3 apresenta o *Software Development Kit* (SDK) do iPhone e a seção 2.4 apresenta o cocos2d, um *framework* para desenvolvimento de jogos 2D para iPhone. A seção 2.5 apresenta os trabalhos correlatos.

2.1 OBJECTIVE-C

Objective-C é uma linguagem projetada para permitir a programação orientada a objetos. É definida como um pequeno, mas poderoso conjunto de extensões ao padrão da linguagem ANSI C. Seus aditamentos a C são essencialmente baseados em Smalltalk que é uma das primeiras linguagens de programação orientada a objeto. A linguagem Objective-C foi concebida para permitir a programação orientada a objeto de forma simples e direta (APPLE, 2009).

Grande parte das linguagens orientadas a objetos consistem em criar definições de objetos em novas classes. Diferentemente da linguagem Java, onde existe um único arquivo para a definição dos métodos e implementação das classes, em Objective-C elas podem ser definidas em duas partes:

a) interface onde são declarados os métodos e variáveis de instância da classe;
b) implementação onde está contido o código que implementa a classe.

Estas partes são tipicamente divididas em dois arquivos, sendo a interface conhecida como *header file* (utilizando o sufixo .h) e a implementação conhecida como *method file* (utilizando o sufixo .m) (APPLE, 2009). O Quadro 1 demonstra a estrutura da definição e implementação de uma classe em Objective C e o Quadro 2 demonstra a mesma classe sendo definida em Java.

File: ClassName.h	File: ClassName.h
#import “ItsSuperclass.h”	#import “ClassName.h”
@interface ClassName : ItsSuperclass {	@implementation ClassName
instance variable declarations	
}	
method declarations	method definitions
@end	@end

Fonte: adaptado de Apple (2009).

Quadro 1 – Estrutura de classe em Objective-C

```
File: ClassName.java
import ItsSuperclass;
public class ClassName extends ItsSuperclass {
   instance variable declarations

   method declarations/definitions
}
```

Quadro 2 – Estrutura de classe em Java

A comunicação entre objetos nesta linguagem é feita através de mensagens. Mensagens podem ser definidas como sendo métodos de uma classe, se comparado com a linguagem Java. O conceito de mensagem é oriundo da linguagem Smalltalk onde todos os elementos são considerados objetos puros e, neste contexto, os objetos interagem uns com os outros através de mensagens e não através de chamadas de função. De fato, na linguagem Smalltak não existe o conceito de funções somente classes, objetos, métodos e atributos.

O programador envia uma mensagem para um objeto solicitando a execução de um método (APPLE, 2009). Para instanciar um objeto de `NSString` por exemplo, é possível instanciar de maneira implícita, enviar uma mensagem chamando o construtor da classe `NSString` ou ainda, instanciar um objeto da classe `NSString` e inicializar o mesmo com uma string como mostra o Quadro 3.

```
NSString *myString = @”This is an example of a string.”;
NSString *newString = [NSString stringWithString: myString];
NSString *newString2 = [[NSString alloc] initWithString: myString]];
```

Quadro 3 – Três maneiras de instanciar um objeto do tipo `NSString`

Em Java também é possível instanciar um objeto de `String`, equivalente ao `NSString` do Objective-C, de maneira implícita ou chamando o construtor conforme mostra o Quadro 4.

```
String myString = “This is an example of a string.”;
String newString = new String(myString);
```

Quadro 4 – Duas maneiras de instanciar um objeto do tipo `String`

Na linguagem Java é possível passar parâmetros para o construtor não sendo necessária a existência de um método de inicialização como comummente é utilizado em Objective-C (DALRYMPLE; KNASTER, 2009, p. 314-315).

Na versão 2.0 do Objective-C foi introduzido à linguagem o conceito de propriedades, (seção 2.2.1) e o gerenciamento automático de memória (seção 2.2.5).

2.1.1 Propriedades

Característica da versão Objective-C 2.0, as propriedades são um intermediário entre as variáveis de instância e métodos. Elas acrescentam uma conveniência sintática, permitindo ao desenvolvedor endereçar variáveis diretamente, sem a necessidade da criação de métodos acessores conforme mostra o Quadro 5.

```
@property float value;

é equivalente:
- (float)value;
- (void)setValue:(float)newValue;
```

Fonte: adaptado de Apple (2009).

Quadro 5 – Definição de uma propriedade

As propriedades são semelhantes às variáveis públicas, mas permitindo que um comportamento seja efetuado quando ocorrer acesso ao mesmo (ZDZIARSKI, 2009, p. 21).

O uso da diretiva `@synthesize` no arquivo de métodos, informa ao compilador para criar automaticamente os métodos acessores sem a necessidade de codificação por parte do desenvolvedor (APPLE, 2009).

Java não possui o conceito de propriedades como em Objective-C, mas é possível criar uma funcionalidade equivalente declarando os métodos *setter* e *getter* como mostra o Quadro 6 (DALRYMPLE; KNASTER, 2009, p. 314-315).

```
public void setValue(float value) {
        this.value = value;
}
public float getValue() {
        return value;
}
```

Quadro 6 – Métodos *setter* e *getter* em Java

2.1.2 Categorias

Categorias permitem adicionar métodos a uma classe já existente, mesmo que o desenvolvedor não tenha acesso aos fontes da classe. Este recurso permite estender a funcionalidade das classes existentes, sem a necessidade de usar o conceito de herança (APPLE, 2009). Categorias anônimas ou *extensions* são categorias que possuem métodos e a implementação desses métodos é realizada juntamente com a implementação da classe no qual a categoria está adicionando métodos (APPLE, 2009). O Quadro 7 mostra a definição de uma categoria `Setter` para a classe `MyObject`.

```
@interface MyObject (Setter)
- (void)setNumber:(NSNumber *)newNumber;
@end
```

Fonte: Apple (2009).

Quadro 7 – Definição de uma categoria `Setter`

> Um dos usos mais comuns de métodos de categorias em Objective-C é a criação de cabeçalhos privados. Isto define que os métodos serão utilizados apenas por implementações internas de uma classe sem a possibilidade de qualquer tipo de acesso externo. (MEYERS; LEE, 2007, p. 504).

Apesar da definição e implementação de uma categoria ser parecida com a definição de uma classe, não é possível incluir novas variáveis na classe da categoria.

Em Java não há necessidade da existência de Categorias para permitir a criação de métodos privados já que o mesmo oferece isso nativamente (DALRYMPLE; KNASTER, 2009, p. 314-315).

2.1.3 Protocolos

Protocolos podem ser vistos como uma lista contendo definições de métodos que serão implementados posteriormente por uma classe (APPLE, 2009). O Quadro 8 mostra a definição de um protocolo.

```
@protocol MyProtocol

- (void)requireMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;
```

```
@required
- (void)anotherRequiredMethod;
@end
```

Fonte: adaptado de Apple (2009).

Quadro 8 – Definição de um protocolo

Através da palavra chave `@optional` é possível definir métodos opcionais que podem ou não ser implementados posteriormente. Usando a palavra chave `@required` ou na omissão de qualquer palavra chave, tornam a implementação do método obrigatório por classes que adotem o protocolo (APPLE, 2009).

Uma classe pode adotar mais de um protocolo. Para que ela possa utilizar um protocolo, o mesmo deve ser mencionado na definição da classe como mostra o Quadro 9.

```
@interface MyObject < MyProtocol >
```

Quadro 9 – Utilização do protocolo `MyProtocol`

Protocolos são equivalentes a interfaces em Java conforme mostra o Quadro 10 (DALRYMPLE; KNASTER, 2009, p. 314-315).

```
public interface MyProtocol {
   public void requireMethod();
}
```

Quadro 10 – Definição de interface em Java

2.1.4 Tipagem dinâmica

Em alguns momentos durante a codificação em Objective-C é necessário o uso de variáveis usando um tipo dinâmico. Isso se deve ao fato de que em algumas situações não se sabe exatamente o tipo do objeto que pode estar contido na variável. Em Objective-C, para definir uma variável com tipo dinâmico deve ser usado a palavra chave `id`.

Segundo Dalrymple e Knaster (2009, p. 245), “o tipo `id` representa um ponteiro para qualquer tipo de objeto. Ele é um tipo genérico de objeto”. Esse tipo pode ser utilizado em conjunto com um protocolo quando é necessário aceitar qualquer tipo de objeto, desde que a classe do objeto tenha adotado o protocolo como é demonstrado no Quadro 11.

```
- (void)setObjectValue: (id<NSCopying>) obj;
```

Fonte: Dalrymple e Knaster (2009, p. 245).

Quadro 11 – Uso do tipo dinâmico `id`

Em Java é possível declarar interfaces, que são equivalentes a protocolos conforme

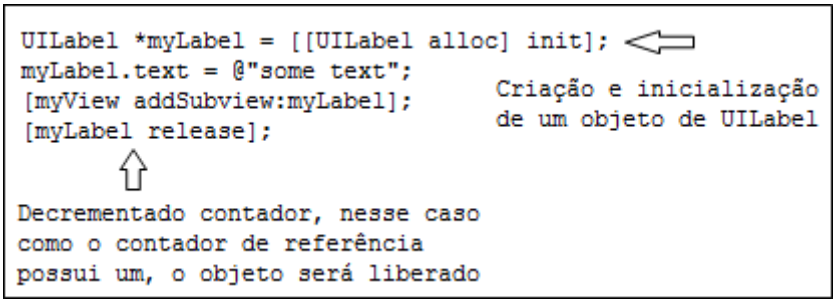
item 2.4.3, em métodos e variáveis sem que seja necessário para isso o uso de um tipo genérico com é o caso de Objective-C.

2.1.5 Gerenciamento de memória

Segundo Goldstein (2009, p. 301), na versão 2.0 do Objective-C foi introduzido “o gerenciamento automático de memória, também chamado de *garbage collector*.”. O gerenciamento automático de memória é responsável por liberar a memória de objetos que não estão sendo mais utilizados (GOLDSTEIN, 2009, p. 301).

Apesar do iPhone SDK permitir o desenvolvimento de aplicações usando Objective-C 2.0 ele não suporta o gerenciamento automático de memória (APPLE, 2009).

Para tratar manualmente o gerenciamento de memória, segundo Campbell (2009, p. 9), “Objective-C oferece um recurso conhecido como contador de referências”. Quando um objeto é instanciado utilizando o método de alocação `alloc` é criado um contador de referência e atribuído ao objeto, onde é acrescentado um a esse contador. Cada vez que o objeto usar o método `retain` é aumentado em um o contador de referências do objeto. Quando o objeto não for mais utilizado e for chamado o método de liberação `release` o contador de referência é decrementado em um. Ao atingir o valor zero, a memória do objeto é liberada do sistema conforme é demonstrado na Figura 1 (CAMPBELL, 2009, p. 9-11).



Fonte: adaptado de Campbell (2009, p. 9-11).

Figura 1 – Exemplo do uso do contador de referência

Segundo Brannan (2009, p. 59), “o gerenciamento manual do contador de referências é cansativo e sujeito a erros”. O método `autorelease` do objeto `NSObject` permite que não seja necessário o controle por parte do desenvolvedor. O método utiliza o conceito de *pool* de liberação para gerenciar as referências de um objeto. Esse *pool* pode ser controlado automaticamente pela aplicação fazendo com que um objeto seja liberado logo após o término da execução do método onde ele foi instanciado conforme é demonstrado no Quadro 12 (BRANNAN, 2009, p. 59-60).

```
- (void) sayHelloTom {
  Simple *objSimple = [[[Simple alloc] init] autorelease];
  [objSimple sayHello:@”Tom”];}
```

Fonte: Brannan (2009, p. 59).

Quadro 12 – `Autorelease` sem controle manual

Em algumas situações faz-se necessário com que o objeto permaneça por mais tempo em memória. Nesse caso o controle é realizado manualmente utilizando-se do objeto `NSAutoreleasePool` (BRANNAN, 2009, p. 59). Assim, o objeto é liberado apenas quando o objeto `NSAutoreleasePool` for liberado conforme Quadro 13.

```
- (void) sayHelloTom {
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  Simple *objSimple = [[[Simple alloc] autorelease] init];
  [objSimple sayHello: @”Tom”];
  [pool release]; }
```

Fonte: Brannan (2009, p. 59).

Quadro 13 – `Autorelease` utilizando `NSAutoreleasePool`

Java também possui o recurso de *garbage collector* porém, diferentemente de Objective-C não é necessário instanciar qualquer objeto para que esse controle seja realizado. Apesar do controle para liberação dos objetos ser realizado automaticamente pelo *garbage collector,* o desenvolvedor pode, em alguns casos, executá-lo de maneira explícita sugerindo que faça a liberação naquele momento. Ainda assim, não é garantido que o *garbage collector* faça a liberação dos objetos no momento em que é chamado (SUN, 2006).

2.1.6 Comparativo entre classes em Objective-C e Java

O Quadro 14 apresenta um comparativo de algumas classes das linguagens Java e Objective-C, fazendo uma breve descrição sobre cada uma delas.

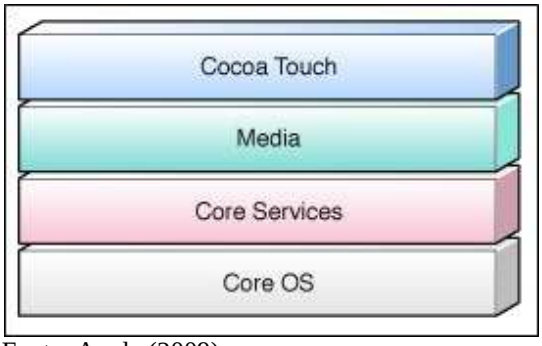
ID	CLASSE	FUNCIONALIDADE
1	**Java:** ArrayList	Classe utilizada para criar lista de objetos
	Objective-C: NSMutableArray, NSArray	
2	**Java:** String, StringBuffer	Classe para armazenar uma sequência de caracteres
	Objective-C: NSString, NSMutableString	
3	**Java:** org.apache.commons.digester.Digester	Classe utilizada para efetuar parser em arquivo XML
	Objective-C: NSXMLParser	
4	**Java:** HashMap	Classe utilizada para criar lista de objetos com chaves identificadas
	Objective-C: NSDictionary, NSMutableDictionary	

5	**Java:** Exception	Classe utilizada para levantar exceções na rotina
	Objective-C: NSException	

Quadro 14 – Relação entre classes em Java e Objective-C

2.2 IPHONE OS

iPhone OS é um sistema operacional baseado no kernel Mach e é utilizado pelos dispositivos iPhone e iPod touch (APPLE, 2009). A plataforma iPhone OS foi construída usando a experiência na criação do Mac OS X e muitas das ferramentas e tecnologias empregadas no desenvolvimento da plataforma tiveram suas raízes no Mac OS X (APPLE, 2009). A implementação das tecnologias contidas no iPhone OS pode ser vista como um conjunto de camadas conforme Figura 2.



Fonte: Apple (2009).

Figura 2 – Camadas que representam o iPhone OS

Nas camadas inferiores do sistema estão os serviços fundamentais em que todas as aplicações dependem, enquanto as camadas de maior nível contêm os mais sofisticados serviços e tecnologias (APPLE, 2009). É recomendado sempre que possível a utilização das camadas de maior nível, pois elas possuem um maior grau de abstração orientado a objeto.

Na camada Cocoa Touch está a maioria das tecnologias que empregam o uso de Objective-C. O *framework* Foundation disponível nessa camada oferece suporte orientado a objetos para coleções, gerenciamento de arquivos, operações de rede entre outros.

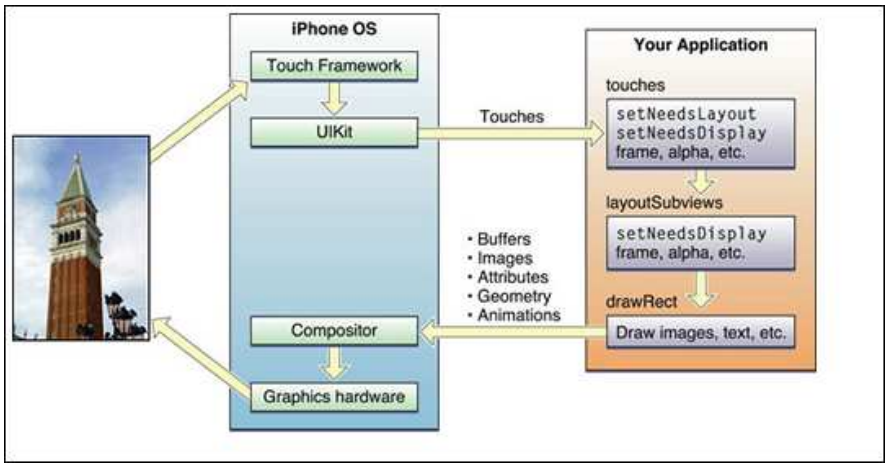
O iPhone SDK comentado no item 2.6 suporta a criação de aplicações orientadas a gráficos que rodam nativamente no iPhone OS. Os aplicativos criados e instalados no sistema são disponibilizados na tela do usuário, juntamente com os demais aplicativos que acompanham o iPhone OS como fotos, tempo e relógio.

2.3 IPHONE SDK

O iPhone SDK versão 3.0 oferece ferramentas e recursos necessários para criação de aplicações nativas para iPhone. Estas aplicações têm acesso a todas as funcionalidades oferecidas pelo dispositivo, como acelerômetro, serviço de localização e interface *multi-touch* (APPLE, 2009). Entre as aplicações disponibilizadas pelo SDK estão o XCode (item 2.3.1), Interface Builder (item 2.3.2), o Instruments (item 2.3.3) e o iPhone Simulator (2.3.4).

Um dos *frameworks* que acompanham o SDK é o Foundation responsável por fornecer várias classes básicas de objetos primitivos, introduzindo paradigmas que definem funcionalidades não abrangidas pela linguagem Objective-C (APPLE, 2009). Este *framework* foi projetado para permitir a persistência de objetos, a independência operacional melhorando a portabilidade e a introdução de convenções coerentes no desenvolvimento de software (APPLE, 2009).

Outro *framework* muito utilizado no desenvolvimento de aplicações para iPhone é a UIKit. Desde o momento que a aplicação é executada pelo usuário, até o término da mesma, a UIKit gerencia a maioria das solicitações de interface. De acordo com Apple (2009), "cada vez que o usuário interage com a interface do dispositivo, uma sequência de eventos é disparada através de chamada de classes contidas dentro da UIKit para executar a solicitação". A Figura 3 mostra a sequência básica de eventos que inicia com o usuário tocando na tela e termina com o sistema de gráficos atualizando a tela com o conteúdo da resposta.



Fonte: Apple (2009).

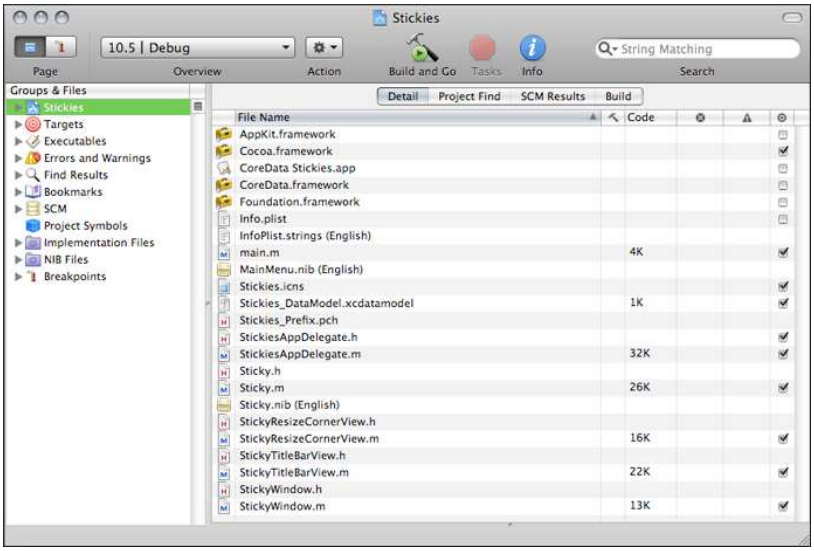
Figura 3 – Sequência de eventos para atualizar tela

As aplicações desenvolvidas para iPhone utilizam o conceito de *Model View Controller* (MVC), pois o próprio desenvolvimento do SDK foi baseado neste conceito. O uso

desta padronização facilita o desenvolvimento e o entendimento das classes do SDK.

2.3.1 XCode

XCode é um *Integrated Development Environment* (IDE) para Mac OS X. No XCode o desenvolvedor tem a disposição ferramentas para criação, depuração e otimização de aplicações, sendo algumas delas: Interface Builder, Instruments (APPLE, 2009). Na Figura 4 é possível visualizar a interface da aplicação.



Fonte: Apple (2009).

Figura 4 – Interface da aplicação XCode

Dentro de um projeto do XCode é possível armazenar grupos de arquivos de fonte, bibliotecas, mídias e qualquer outro recurso que seja necessário para construir a aplicação. Além da criação de aplicações, é possível criar também *frameworks* ou *static libraries* que podem ser utilizados em outros projetos.

2.3.2 Interface Builder

Interface Builder é uma aplicação da Apple para criação de interfaces de usuário

usando tanto Carbon[1] quanto Cocoa[2]. No Interface Builder é possível criar interfaces que seguem o padrão Aqua de interface gráfica do Mac OS X. Para construir uma interface, o desenvolvedor deve arrastar objetos de interface da paleta de componentes para uma janela ou um menu (APPLE, 2009).

Através da integração do Interface Builder com o XCode é possível com o uso de *outlets*[3] conectar objetos da interface com objetos do código da aplicação (MARK; LAMARCHE, 2009, p. 34-35). Além disso também é possível associar *actions* ou eventos dos objetos da interface, com ações do código da aplicação (MARK; LAMARCHE, 2009, p. 36-37).

As interfaces criadas no Interface Builder são armazenadas em arquivos com extensão xib (APPLE, 2009). Nesses arquivos que seguem o padrão *eXtensible Markup Language* (XML) estão contidas informações sobre cada objeto da interface. A Figura 5 demonstra, na imagem A o menu com as propriedades do objeto selecionado, na imagem B a janela onde são colocados os componentes, na imagem C a paleta de componentes e na imagem D a lista de objetos contidos no arquivo com extensão xib.

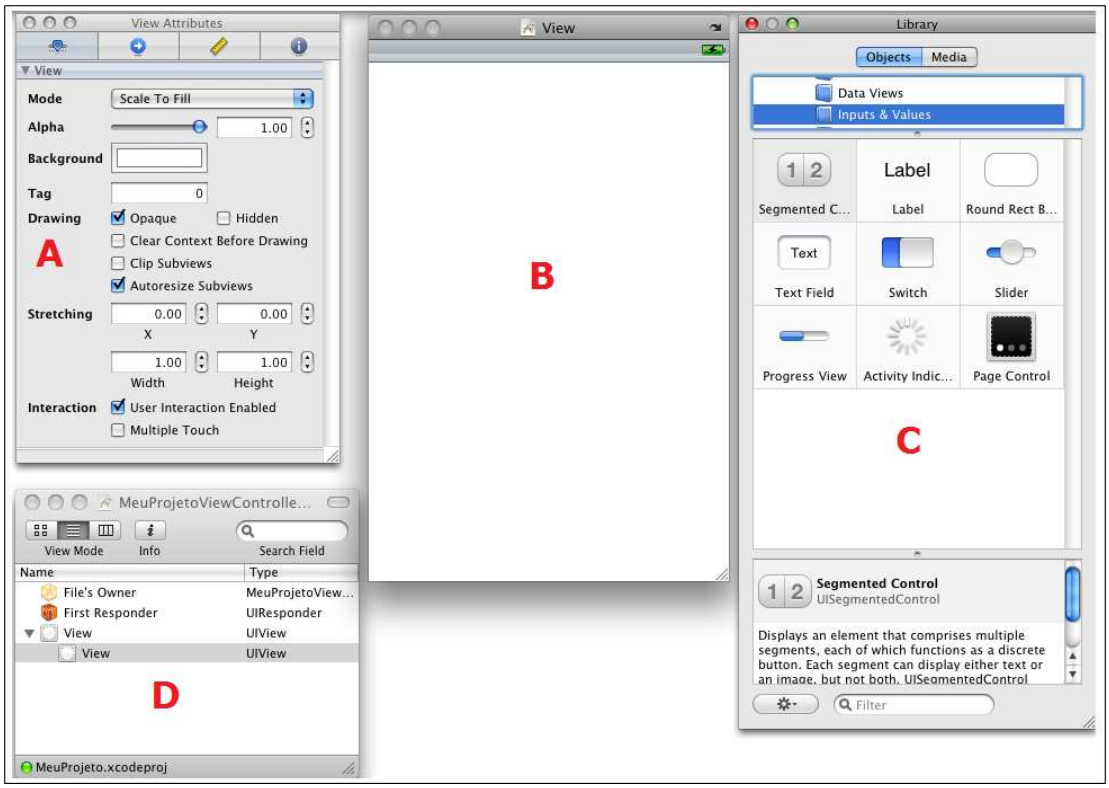


Figura 5 – Janelas do Interface Builder

[1] Carbon é uma API procedural de 32-bit para desenvolvimento de aplicações para Mac OS X (APPLE, 2009).
[2] Cocoa é uma coleção de frameworks e APIs que permitem o desenvolvimento de aplicações em plataforma 32-bit e 64-bit além do uso de Objective-C 2.0 que possui *garbage collector* (APPLE, 2009).
[3] *Outlets* são instâncias de variáveis que referenciam um objeto. Através do Interface Builder é possível conectar um componente visual com uma instância de um objeto definido no código como `IBOutlet` (BUCANEK, 2009, p. 363).

2.3.3 Instruments

Instruments é um conjunto de ferramentas para análise de código no Mac OS X (APPLE, 2009). Com o uso dessas ferramentas é possível:

a) rastrear diferentes aspectos do comportamento de um processo;
b) gravar um sequência de ações do usuário de maneira que possam ser reproduzidas várias vezes, coletando informações de cada execução;
c) automatizar testes do código da aplicação;
d) analisar desempenho da aplicação codificada;
d) efetuar *stress test*[4] em partes da aplicação.

Algumas ferramentas são construídas para a própria aplicação do Instruments. Cada ferramenta tem as suas próprias opções de configuração e formas de exibir informações, de acordo com o tipo de dados que é coletado (APPLE, 2009). As ferramentas são agrupadas em categorias com base no tipo de informação que é coletada por ela.

A ferramenta ObjectAlloc, já embutida na instalação do Instruments, permite acompanhar graficamente a alocação de memória em um determinado aplicativo. Para usar essa ferramenta é necessário que a mesma seja iniciada a partir de um único processo permitindo assim coletar as informações desde o início do processo (APPLE, 2009). A Figura 6 demonstra a aplicação em execução usando a ferramenta ObjectAlloc.

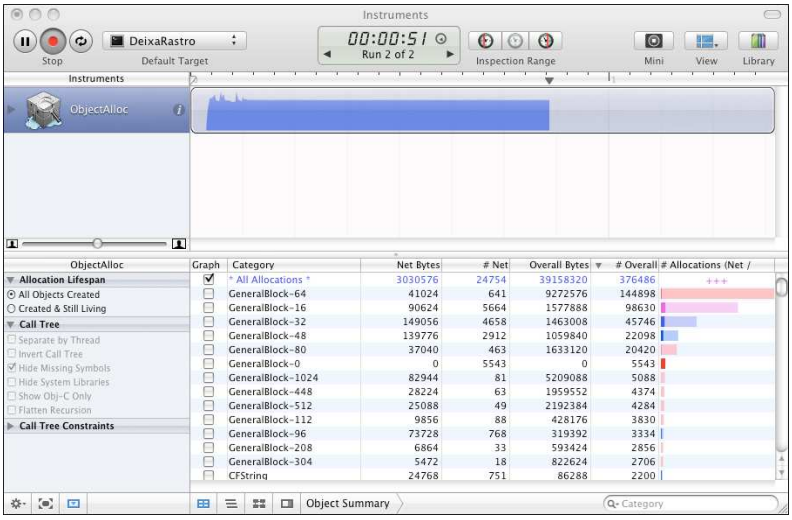


Figura 6 – ObjectAlloc em execução no Instruments

[4] *Stress test* é um teste que vai além da utilização normal da aplicação. São executadas ações repetitivamente com o objetivo de testar limites de carga e volume. (GRAHAM et al., 2008, p. 183).

2.3.4 iPhone Simulator

O iPhone Simulator permite construir e executar aplicações do iPhone em um desktop. O ambiente do simulador pode ser usado para:

a) resolver problemas na aplicação durante o projeto inicial;

b) testar a interface do usuário;

c) medir a utilização de memória do aplicativo.

Ainda, o aplicativo possui diversas maneiras de interação permitindo o uso integrado do teclado, mouse para simular toques na tela, rotação do dispositivo, entre outros (APPLE, 2009). A Figura 7 demonstra a aplicação em execução.

Figura 7 – iPhone Simulator em execução

O iPhone suporta o Objective-C introduzido na versão 10.5 do Mac OS X, porém não permite o acesso a classes de metadados do Objective-C. Isto significa que, se forem utilizadas na aplicação classes de metadados do Objective-C, esta aplicação não conseguirá ser executada no iPhone Simulator (APPLE, 2009).

A instalação de uma aplicação no iPhone Simulator é bastante simples, bastando apenas construir a aplicação usando o XCode incluído no iPhone SDK. No iPhone Simulator é possível instalar apenas aplicações que forem criadas pelo desenvolvedor.

Para enviar uma aplicação para um dispositivo iPhone ou iPod touch é necessário que o dispositivo, além do desenvolvedor, esteja cadastrado no iPhone Dev Center (APPLE, 2009). Além dessa situação, a única forma de instalar aplicações em um dispositivo iPhone ou iPod touch, é através da aquisição de aplicações pela App Store disponibilizada pela Apple.

2.4 COCOS2D

Cocos2d é um *framework* para construção de jogos 2D, demos e outras aplicações gráficas e interativas. É baseado na API desenvolvida anteriormente para linguagem de programação Python (COCOS2D, 2009). Todas as classes criadas em Cocos2d herdam os métodos e atributos da classe `CocosNode`. É nessa classe que estão os principais controles necessários para desenvolvimento da aplicação.

2.4.1 Classe `Scene`

Cena, ou `Scene`, é uma peça independente do fluxo de trabalho da aplicação. Alguns desenvolvedores podem chamá-los de telas ou etapas. Uma aplicação desenvolvida em Cocos2d pode ter muitas cenas, mas apenas uma delas estará ativa em um determinado momento (COCOS2D, 2009). Um exemplo simples utilizando cenas é um jogo com uma cena de introdução, duas cenas de fases e uma cena para o fim do jogo conforme mostra a Figura 8.

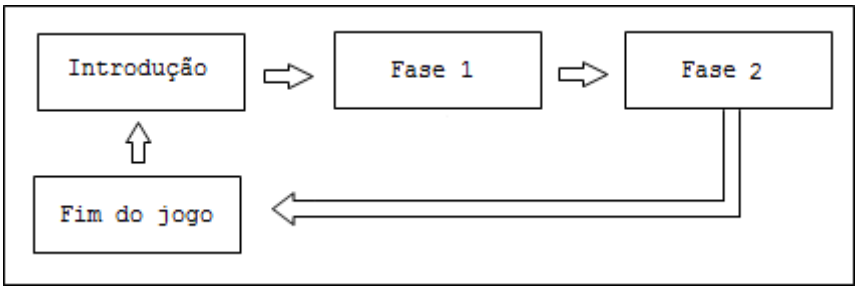


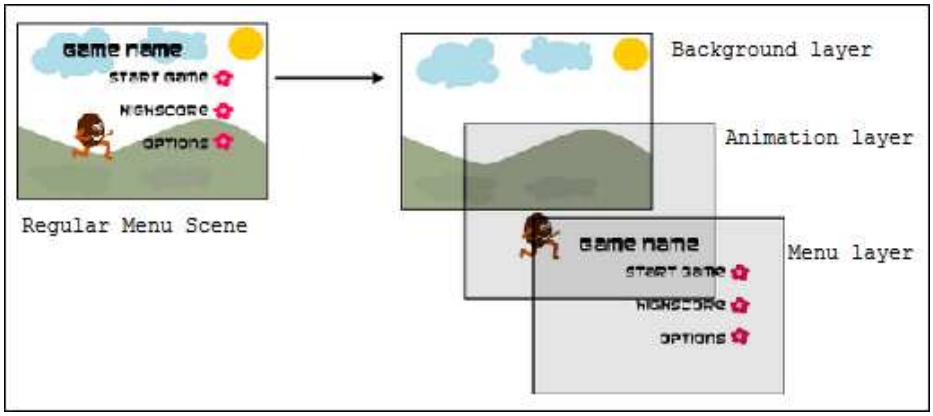
Figura 8 – Cenas de um jogo usando Cocos2d

Através da função `switchScene` da classe `Scene` é possível alternar entre as cenas da aplicação. Cada cena em Cocos2d pode ser composta de uma ou mais camadas.

2.4.2 Classe `Layer`

`Layer` é a camada existente dentro das cenas. A classe `Layer` é responsável por definir a aparência e comportamento da janela. É nela que são incluídos botões, imagens, animações entre outros componentes oferecidos pelo *framework*. Ele pode ser semitransparente, com furos de transparência, parcialmente transparente ou totalmente transparente. A transparência

de um objeto de `Layer` permite que sejam vistos camadas abaixo dele conforme é demonstrado na Figura 9 (COCOS2D, 2009).



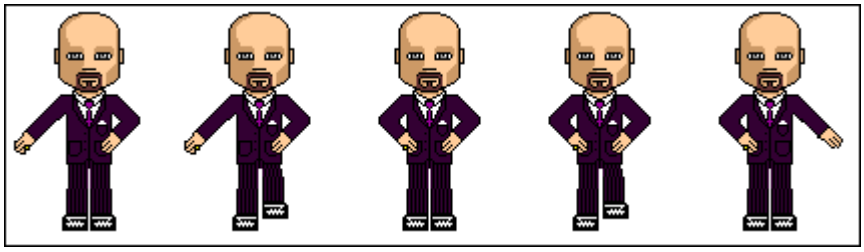
Fonte: Cocos2d (2009).

Figura 9 – Camadas sobrepostas com transparência

É na classe `Layer` que são definidos os manipuladores de eventos. Ao ser disparado um evento, o mesmo é propagado para os objetos de `Layer` de frente para trás, até que algum deles aceite (COCOS2D, 2009). Em algumas situações, onde é necessário mudar a cor de fundo da camada, é utilizada a classe `ColorLayer` que se diferencia da classe `Layer` apenas por essa propriedade.

2.4.3 Classe `Sprite`

A classe `Sprite` é responsável por permitir animações utilizando uma imagem 2D. Através dessa classe é possível mover a imagem de um lado para o outro, ou ainda, rodar, escalar e até mesmo animar utilizando uma sequência de imagens. A Figura 10 demonstra várias imagens que podem ser utilizadas na criação de um animação usando `Sprite`.



Fonte: adaptado de Cocos2d (2009).

Figura 10 – Sequência de imagens de um personagem

Através do método `runAction` é possível enviar ações para serem executadas. O Quadro 15 demonstra a codificação de ações de rotação de uma imagem.

```
1:     id actionTo0 = [RotateTo actionWithDuration: 2 angle:-45];
2:     id actionTo1 = [RotateTo actionWithDuration:2  angle:0];
3:      Sprite *kathia = [Sprite spriteWithFile:@"grossinis_sister2.png"];
4:     [kathia setPosition:ccp(240,160)];
5:     [kathia runAction: [Sequence actions:actionTo20 actionTo2, nil]];
```

Quadro 15 – Código para efetuar rotação de uma imagem

Na linha 3 do Quadro 15 pode ser observado o momento em que o objeto de `Sprite` é criado através da mensagem `spriteWithFile` passando como parâmetro o nome do arquivo contendo a imagem.

Um objeto de `Sprite` pode conter outros objetos associados a ele. Isto permite que, ao efetuar uma transformação no objeto pai, todos os demais objetos associados a ele sejam afetados pelo mesma transformação.

2.5 TRABALHOS CORRELATOS

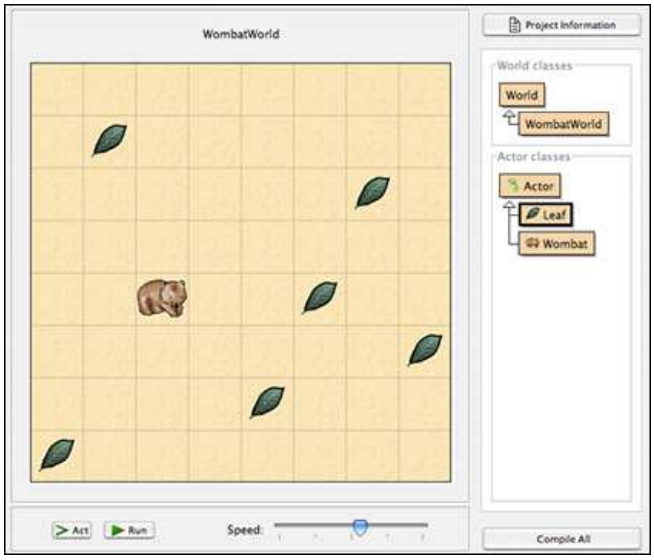
Existem aplicações que possuem características semelhantes ao proposto neste trabalho, tais como o Greenfoot (GREENFOOT, 2008), m-Geo (MARÇAL et al., 2009), Skattjäkt (CELEKT, 2007), Sistema de gerenciamento de objetos de aprendizagem para dispositivos móveis (FRANCISCATO; MEDINA, 2009) e Gorillas (GORILLAS, 2009).

2.5.1 Greenfoot

O Greenfoot foi concebido e implementado na Universidade de Kent na Inglaterra e na Universidade Deakin em Melbourne na Austrália. Ele é uma combinação entre *framework* de criação de mundos 2D em Java e um ambiente de desenvolvimento que integra ferramentas como visualizador de classes, editor, compilador e execução (GREENFOOT, 2008).

O Greenfoot utiliza fortemente o conceito de orientação a objetos através de herança de classes e sobrescrita de métodos. É composto por classes que devem ser herdadas e implementadas na própria interface do Greenfoot.

A Figura 11 demonstra a interface gráfica do Greenfoot. A imagem quadriculada representa o mundo. À direita estão as classes implementados e abaixo do mundo os controles para execução da aplicação.

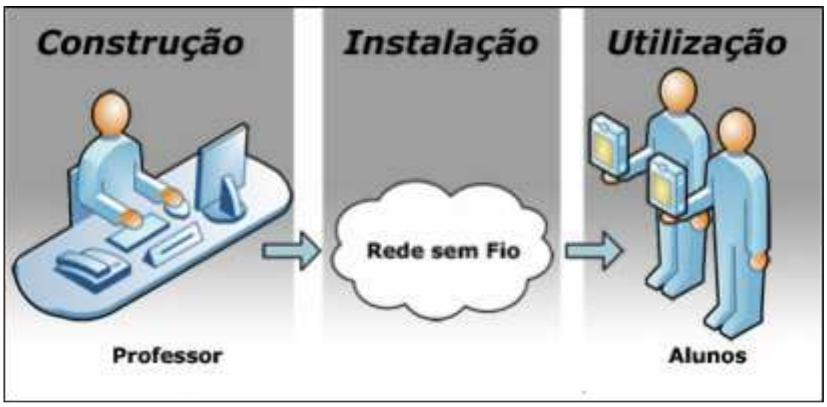


Fonte: Greenfoot (2008).

Figura 11 – Arquivo XML com definição do mundo

2.5.2 m-Geo

A m-Geo é uma ferramenta computacional para elaboração de material de apoio ao ensino de geometria. A ferramenta possibilita que o professor crie o material de apoio envolvendo geometria espacial e converta em aplicações (MARÇAL et al., 2009). Essas aplicações são disponibilizadas através de rede sem fio ou internet, para os alunos baixarem, utilizando os seus dispositivos móveis. A Figura 12 demonstra esse processo em execução.



Fonte: Marçal et al. (2009).

Figura 12 – Processo usando a ferramenta

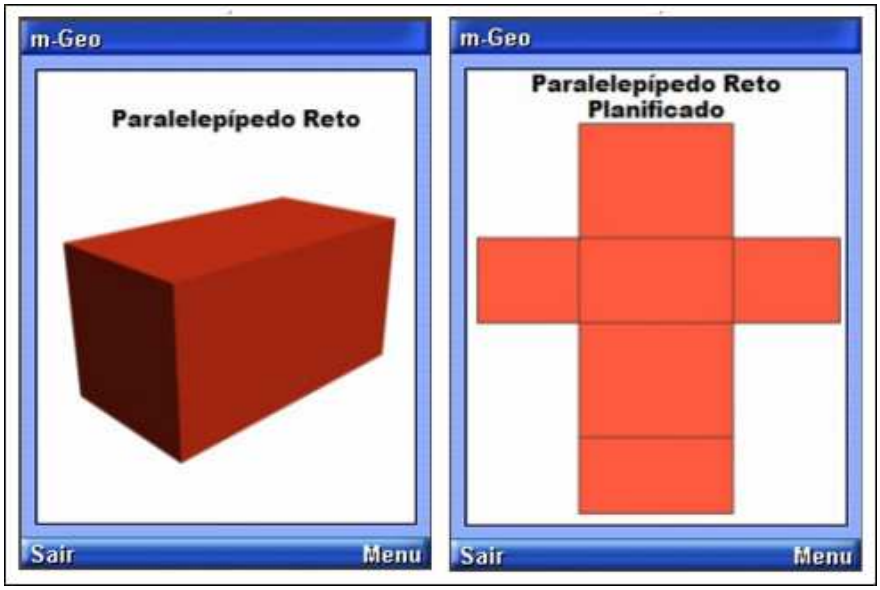
Ao executar a aplicação já no dispositivo móvel, o aluno encontra um texto introdutório cadastrado previamente pelo professor. Com o término da introdução o aluno passa a primeira fase onde é demonstrado um modelo 3D interativo de uma forma geométrica e é necessário que ele efetue interações pré-estabelecidas pelo professor como

reposicionamento, rotação e operações de aproximação e afastamento.

Em sequência às interações, vê-se um questionário que deve ser respondido pelo aluno antes de ir para a próxima fase. Para cada resposta correta é demonstrado um texto adicional cadastrado pelo professor e para cada resposta incorreta, é apresentada uma animação com a resposta correta.

Na fase seguinte o aluno deve realizar novas interações e responder a novos questionamentos, usando como base os conhecimentos obtidos na fase anterior. Na terceira fase será solicitado ao aluno a construção de uma mapa conceitual interligando os conceitos já discutidos e construindo relações entre os mesmos. Esse mapa é armazenado para verificação futura do professor ou discussão cooperativa com colegas.

Na última fase é apresentado ao aluno o cenário e o respectivo problema enunciado pelo professor. Dentro do cenário, o elemento 3D estudado é apresentado com as opções básicas de interação já mencionadas anteriormente e uma nova denominada planificação. A planificação considerada como o desmonte da estrutura 3D em suas respectivas faces bidimensionais pode auxiliar na realização de tarefas específicas, como o cálculo da área da face ou do elemento 3D completo (MARÇAL et al., 2009). A Figura 13 demonstra uma imagem geométrica 3D e em 2D já com a planificação.



Fonte: Marçal et al. (2009).

Figura 13 – Duas telas da aplicação mGeo em execução

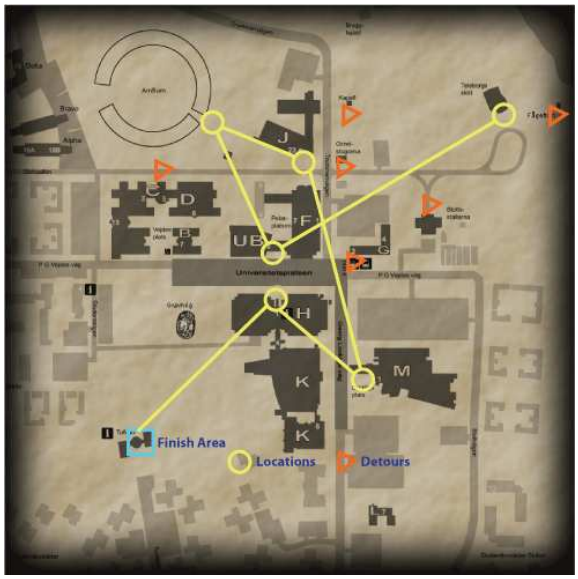
2.5.3 Skattjäkt

Skattjäkt é um jogo para dispositivos móveis projetado para encorajar jovens a

resolverem os mistérios em torno de um castelo construído no campus da Universidade de Växjö na Suécia. O jogo é inspirado no estilo caça ao tesouro e até seis equipes podem participar simultaneamente.

O jogo inicia com um vídeo detalhando o mistério da volta de um fantasma chamado Anna Koskull e os jogadores precisam ajudá-la a resolver o mistério de seu falecido marido Frederico Bonde a fim de libertar seu espírito do limbo. Após o término do vídeo os jogadores são separados em equipes e o jogo inicia.

Cada equipe possui pelo menos um dispositivo móvel onde aparecerá uma marcação em um determinado local no campus. Além da marcação ainda é informado uma dica como, olhar para os pés do leão, permitindo facilitar o encontro de um código de quatro dígitos disponível no local. Ao encontrar o código e digitar o mesmo no dispositivo móvel, irá surgir um questionamento sobre uma determinada informação existente no castelo, como por exemplo, o que está escrito no escudo da família do castelo. Achando o escudo e respondendo corretamente o questionamento, será informado no dispositivo uma nova pista para o próximo local. Caso a resposta esteja incorreta será informado um local alternativo sem demonstrar que a equipe respondeu incorretamente e que levará mais tempo para chegar ao local correto. As equipes precisam resolver os mistérios de cada um dos 6 lugares e quem chegar ao sétimo lugar primeiro ganha. A Figura 14 demonstra o mapa após o término da competição.



Fonte: Celekt (2007).

Figura 14 – Mapa após o término do jogo Skattjäkt

2.5.4 Sistema de gerenciamento de objetos de aprendizagem para dispositivos móveis

O sistema de gerenciamento de objetos de aprendizagem foi desenvolvido com base na dificuldade de encontrar objetos de aprendizagem que possam ser executados em dispositivos móveis, e também na desorganização das informações vinculadas aos objetos (FRANCISCATO; MEDINA, 2009, p. 6-7).

No desenvolvimento do sistema são levadas em consideração as especificidades de cada dispositivo tais como:

a) hardware: memória, processamento, resolução da tela e disponibilidade de som;

b) software: navegador, *plugins* necessários e sistema operacional.

O sistema está baseado nas tecnologias da web 3.0, e foi modelado para ser visualizado em dispositivos móveis. O uso do padrão web 3.0 para organizar informações permite que os objetos de aprendizagem sejam cadastrados com uma grande riqueza de detalhes, favorecendo aos usuários que necessitam de informações adicionais para acessar estes a partir de seus dispositivos móveis (FRANCISCATO; MEDINA, 2009, p. 8). A Figura 15 expõe a tela inicial do sistema.

Fonte: Franciscato e Medina (2009, p. 7).

Figura 15 – Tela inicial do sistema de gerenciamento

Através da opção cadastrar objetos é possível incluir novos objetos com uma grande diversidade de informações como, nome do aluno, disciplina, responsável pelo aluno, sistema operacional recomendável, tipo de linguagem utilizada, navegadores suportados, uso de som e vídeo, dispositivos móveis suportados, processamento, memória ram, resolução da tela e um resumo geral do objeto.

Além do recurso de cadastro é possível pesquisar por objetos já cadastrados através de qualquer dado informado durante o cadastro. A opção também permite efetuar buscas por sequência de escolhas, que seria a junção de um ou mais campos do cadastro.

No desenvolvimento da aplicação são utilizadas as linguagens Javascript, HTML e PHP. Na manipulação, análise e consulta dos dados em formato *Resource Description Framework* (RDF) é utilizada a biblioteca RDF API for PHP que é um conjunto de ferramentas de código aberto (FRANCISCATO; MEDINA, 2009, p. 8).

2.5.5 Gorillas

Gorillas é um jogo para iPhone baseado no antigo jogo Gorilla.bas desenvolvido para MS-DOS. É possível jogar desde um jogador contra a máquina, até multi-jogadores. O jogo foi desenvolvido utilizando o *framework* Cocos2d comentado no item 2.6.

O objetivo do jogo consiste em arremessar um objeto, representado pela imagem de uma banana, no jogador adversário. Diferentemente da versão original, onde havia a necessidade de informar o grau e a velocidade do arremesso, através do iPhone é possível visualizar graficamente o grau do arremesso e ajustar a melhor posição apenas com o movimento do dedo sobre a tela. Além do grau do arremesso, deve ser levado em consideração a velocidade do vento, a gravidade e a silhueta da cidade. A Figura 16 demonstra a interface do jogo.

Fonte: Gorillas (2009).

Figura 16 – Interface do jogo Gorillas

2.6 FURBOT

O Furbot é um *framework* que foi concebido a partir da necessidade de seus criadores de prover um ambiente desafiador, que permitisse aos alunos resolverem problemas de programação utilizando a metáfora de jogos de computador (VAHLDICK; MATTOS, 2009, p. 1). O Furbot foi construído utilizando o conceito MVC, o que permite uma fácil adaptação a mudanças de interface gráfica.

O Furbot tem o seu funcionamento a partir da programação de um robô em um mundo bidimensional. Os movimentos e ações do robô são determinados através da implementação do método `inteligencia`. São utilizados comandos em português como `ehVazio`, `ehFim` e `andarDireita` com o objetivo de facilitar o processo de desenvolvimento. O Quadro 16 demonstra o uso destes métodos no desenvolvimento de um exercício.

```
import br.furb.furbot.Furbot;
public class ExemploFurbot extends Furbot {
   public void inteligencia() {
      while (!ehFim(DIREITA)) { //desloca o furbot até o limite direito do mundo
         if (!ehVazio(DIREITA)) //se nao houver um objeto no caminho...
            andarDireita(); //faz o furbot deslocar-se para a proxima célula
          else   //trata a situacao em que existe um obstaculo na direcao do furbot
            ;
      } //while
   }
}
```

Fonte: Vahldick e Mattos (2009, p. 3).

Quadro 16 – Exemplo de uso do *framework* Furbot

Para criação de um mundo bidimensional no Furbot é necessária a criação de um arquivo de parametrização onde são preenchidas informações como tamanho do mundo e objetos existentes no mundo. Este arquivo deve ser criado usando o padrão XML (Quadro 17).

```
<furbot>
    <enunciado> Exemplo de enunciado... </enunciado>
    <mundo> <qtdadeLin>7</qtdadeLin> <qtdadeCol>4</qtdadeCol> </mundo>
    <robo>  <x>0</x> <y>0</y> </robo>
    <objeto class="br.furb.furbot.Alien"> <x>2</x><y>1</y>  </objeto>
    <objeto class="br.furb.furbot.Alien"> <x>2</x><y>2</y>  </objeto>
    <objeto class="br.furb.furbot.Alien"> <x>2</x><y>5</y>  </objeto>
</furbot>
```

Fonte: Vahldick e Mattos (2009, p. 3).

Quadro 17 – Arquivo XML com definição do mundo

2.6.1 Visão geral

O classes que compõe o *framework* Furbot podem são agrupadas em 3 pacotes e são melhor detalhadas no Quadro 18.

PACOTE	DESCRIÇÃO
furbot	Classes de interface e que representam os objetos do mundo.
exceptions	Classes responsáveis pelo tratamento de exceções específicas do *framework*.
suporte	Classes com funcionalidades diversas utilizadas quando programa é executado ou reiniciado.

Quadro 18 – Detalhamento dos pacotes do *framework* Furbot

Na Figura 17 é apresentado um diagrama de pacotes do Furbot demonstrando o pacote furbot e suas dependências.

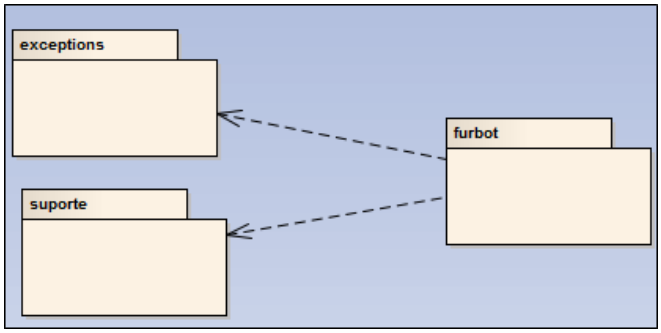


Figura 17 – Diagrama de pacotes do Furbot

Na Figura 18 é apresentado um diagrama de classes do pacote `furbot`.

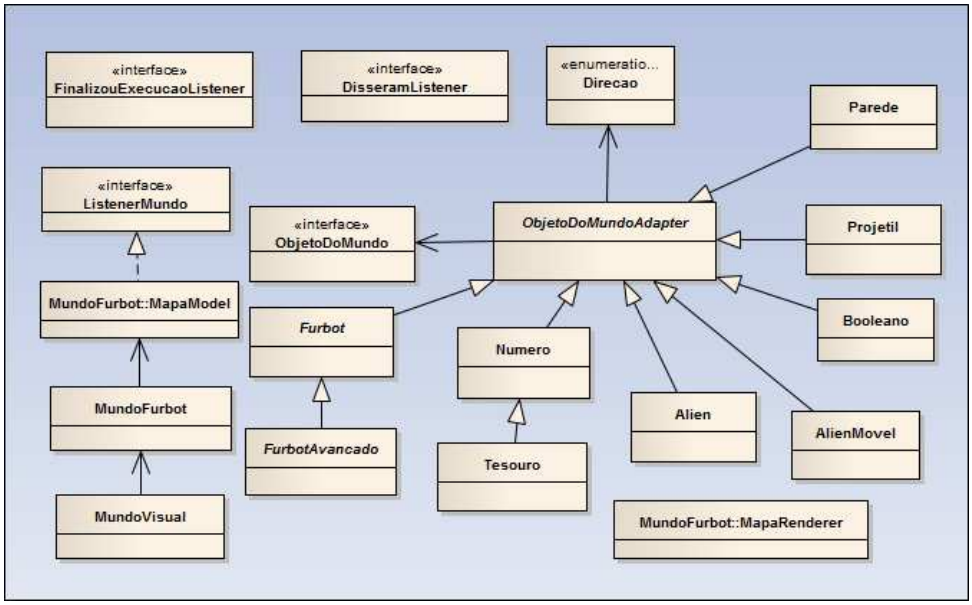


Figura 18 – Diagrama de classes do pacote `furbot`

As principais classes contidas no pacote furbot são: `ObjetoDoMundoAdapter`, `MundoFurbot` e `MundoVisual`. Os métodos dessas classes são detalhados no apêndice A, sendo o Quadro 36 para a classe `ObjetoDoMundoAdapter` , o Quadro 37 para `MundoFurbot` e Quadro 38 para o `MundoVisual`.

Na Figura 19 é demonstrado um diagrama de classes do pacote `suporte`.

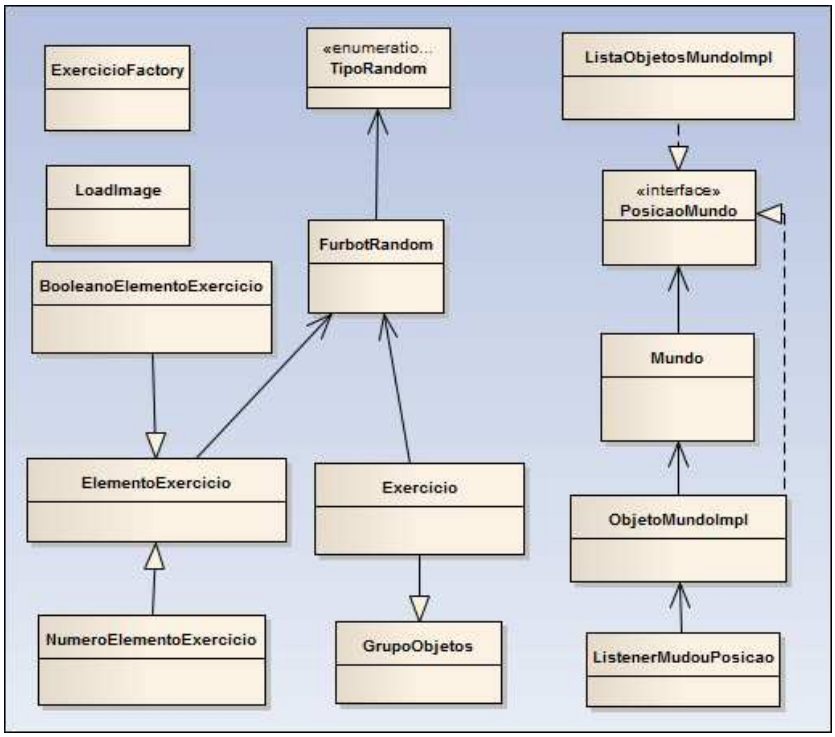


Figura 19 – Diagrama de classes do pacote `suporte`

As principais classes contidas no pacote suporte são: `Mundo`, `ObjetoMundoImpl` e `Exercicio`. Os métodos dessas classes são detalhados no apêndice A, sendo o Quadro 39 para a classe `Mundo`, Quadro 40 para a classe `ObjetoMundoImpl` e o Quadro 41 para a classe `Exercicio`.

Na Figura 20 é apresentado o diagrama de classes do pacote `exception`.

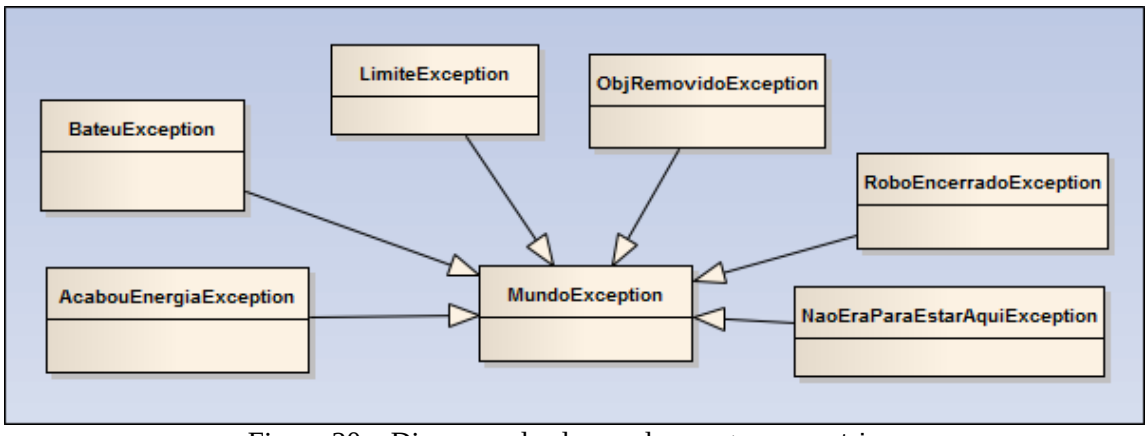


Figura 20 – Diagrama de classes do pacote `exception`

Na Figura 21 são apresentadas todas as classes do *framework* Furbot e seus relacionamentos.

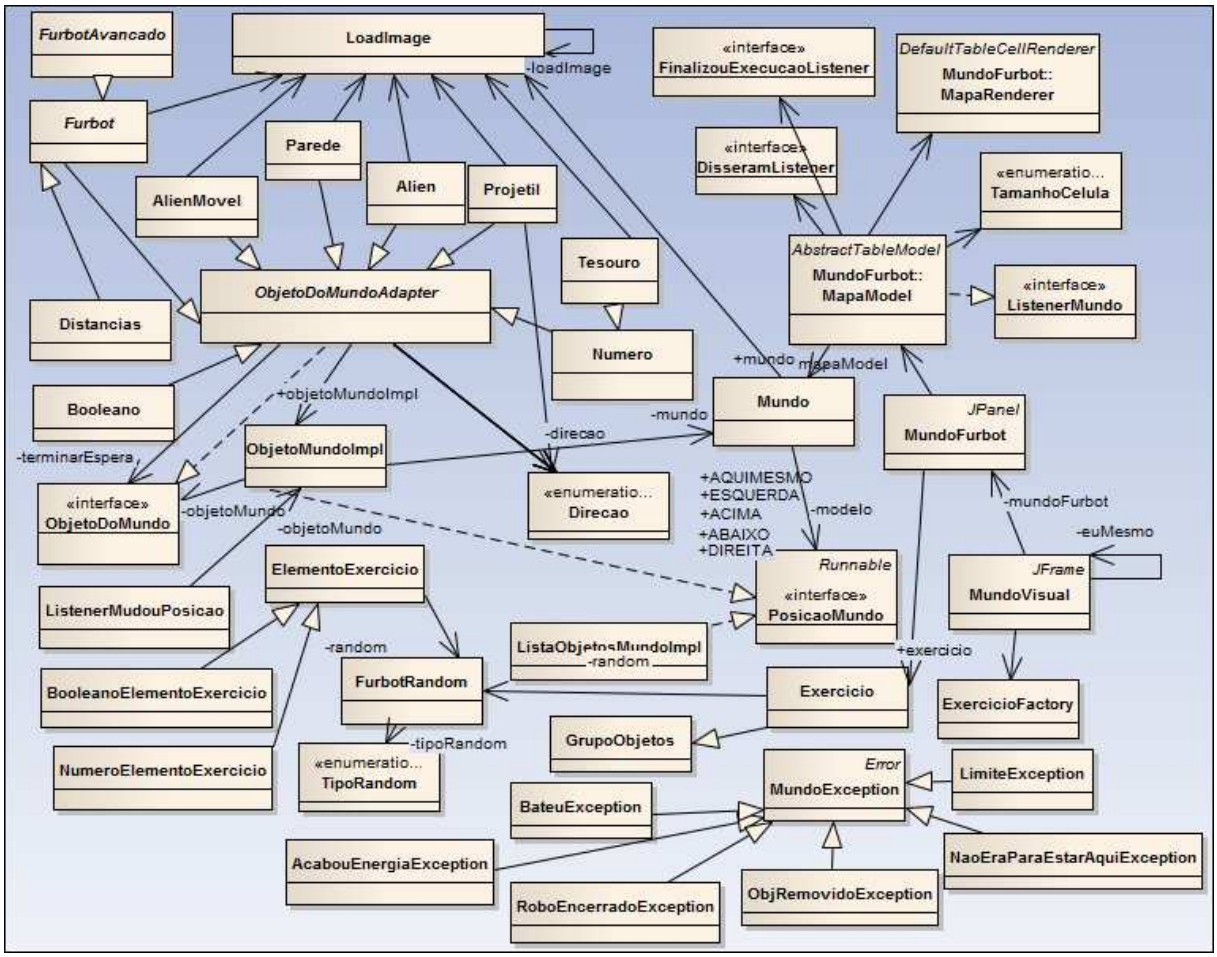


Figura 21 – Classes que compõe o framework Furbot

3 DESENVOLVIMENTO

Para detalhar o processo de desenvolvimento serão abordados a análise e especificação dos requisitos no item 4.1, especificação através de diagramas no item 4.2 e conversão da ferramenta Furbot em Java para Objective-C no item 4.3.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O ambiente proposto deverá:

a) permitir a criação de aplicações bidimensionais para iPhone, usando uma estrutura similar a do Furbot *desktop* (Requisito Funcional - RF);
b) utilizar arquivos do tipo XML para armazenar propriedades de configuração do mundo e seus personagens (RF);
c) permitir a criação de aplicação bidimensionais, considerando as limitações de tela do iPhone (RF);
d) permitir o mapeamento de interação via teclado para interação de interface *multi-touch* e acelerômetro (RF);
e) converter os controles e objetos do mundo do Furbot para a linguagem Objective-C (RF);
f) considerar o uso do iPhone SDK para desenvolvimento das classes do iFurbot (Requisito Não Funcional - RNF);
g) permitir executar aplicações desenvolvidas utilizando o *framework* convertido em Objective-C no dispositivo iPhone (RNF).

3.2 ESPECIFICAÇÃO

Nesta seção será apresentado o diagrama de caso de uso, diagrama de atividades, diagrama de pacotes e diagrama de classes criados a partir da ferramenta Enterprise Architect.

3.2.1 Diagrama de caso de uso

O diagrama de caso de uso apresentado nesta seção demonstra as funcionalidades existentes na ferramenta na visão de um usuário comum, que não possui conhecimento de programação representado no diagrama como ator usuário. Além deste, o diagrama também demonstra a existência de outro ator definido no diagrama como programador, que possui além das habilidades do usuário, habilidades para modificar o robô e o mundo furbot. Na Figura 22 são demonstradas todas as ações possíveis de serem realizadas pelos atores.

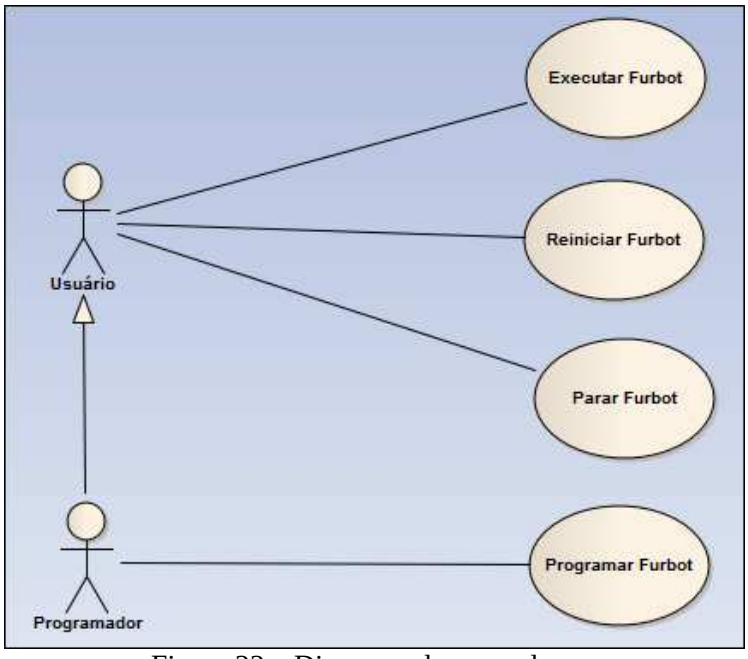


Figura 22 – Diagrama de casos de uso

O Quadro 19 demonstra os cenários para o caso de uso `Executar Furbot`.

Descrição: Execução do exercício desenvolvido
Cenário principal
1: Usuário executa a aplicação.
2: Sistema apresenta tela de introdução com o enunciado do exercício.
3: Usuário indica para ir a tela principal.
4: Sistema apresenta tela principal com o mundo furbot.
5: Usuário indica para executar.
6: Sistema apresenta mundo furbot em execução.
Cenário alternativo (passo 6)
7: Usuário indica para ir a tela de log
8: Sistema apresenta tela de log com as mensagens enviadas pelos objetos do mundo furbot.
9: Usuário indica para ir a tela principal
10: Retorna ao passo 6.

Quadro 19 – Cenário do caso de uso `Executar Furbot`

O Quadro 20 demonstra os cenários para o caso de uso `Reiniciar Furbot`.

Descrição: Reinício do exercício em execução.
Cenário principal
1: Usuário executa a aplicação.
2: Sistema apresenta tela de introdução com o enunciado do exercício.
3: Usuário indica para ir para a tela principal
4: Sistema apresenta tela principal com o mundo furbot.
5: Usuário indica para executar.
6: Sistema apresenta mundo furbot em execução.
7: Usuário indica para reiniciar.
8: Sistema para a execução do exercício.
9: Sistema reinicia o mundo furbot.
Cenário alternativo (passo 6)
10: Usuário indica para ir a tela de log
11: Sistema apresenta tela de log com as mensagens enviadas pelos objetos do mundo furbot
12: Usuário indica para ir a tela principal
13: Retorna ao passo 6.

Quadro 20 – Cenário do caso de uso `Reiniciar Furbot`

O Quadro 21 demonstra os cenários para o caso de uso `Parar Furbot`.

Descrição: Parando exercício em execução
Cenário principal
1: Usuário executa a aplicação.
2: Sistema apresenta tela de introdução com o enunciado do exercício.
3: Usuário indica para ir para a tela principal
4: Sistema apresenta tela principal com o mundo furbot.
5: Usuário indica para executar.
6: Sistema apresenta mundo furbot em execução.
7: Usuário indica para parar a execução.
8: Sistema para a execução do exercício.
Cenário alternativo (passo 6)
9: Usuário indica para ir a tela de log
10: Sistema apresenta tela de log com as mensagens enviadas pelos objetos do mundo furbot
11: Usuário indica para ir a tela principal
12: Retorna ao passo 6.

Quadro 21 – Cenário do caso de uso `Parar Furbot`

O Quadro 22 apresenta os cenários para o caso de uso `Programar Furbot`.

Descrição: Programando exercício
Cenário principal
1: Programador inicia IDE de desenvolvimento.
2: IDE é apresentada.
3: Programador indica para criar um novo projeto de iFurbot.
4: IDE cria um novo projeto de iFurbot usando o template disponível na IDE.
5: Programador associa arquivo XML ao projeto.
6: Programador codifica método inteligencia.
7: Programa executa aplicação.
8: Sistema apresenta tela de introdução com o enunciado do exercício.
Cenário exceção (passo 8) – XML de configuração inválido
a) Sistema apresenta mensagem de erro e retorna ao passo 5.

Quadro 22 – Cenário do caso de uso `Programar Furbot`

3.2.2 Diagrama de atividades

O diagrama de atividade apresentado na Figura 23 mostra o processo inicial ao entrar na aplicação desenvolvida utilizando o *framework* iFurbot.

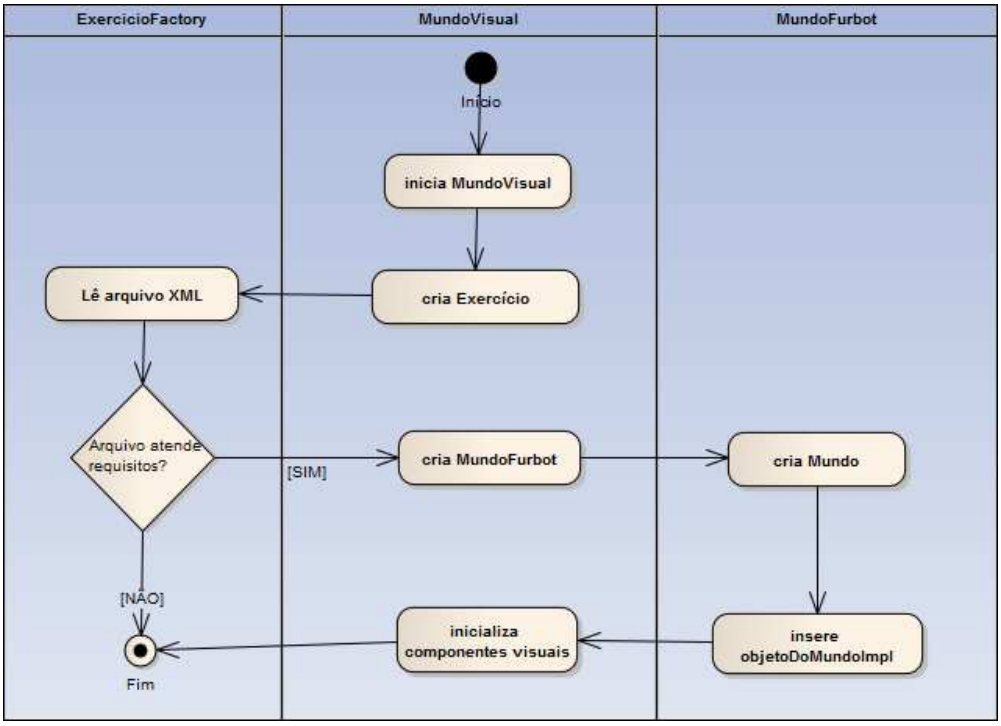


Figura 23 – Diagrama de atividades do início da aplicação

3.2.3 Diagrama de pacotes

As classes criadas em Objective-C foram agrupadas em pacotes da mesma forma que a versão original do Furbot. Os quatro pacotes contidos na versão do iFurbot são melhor detalhados no Quadro 23.

PACOTE	**DESCRIÇÃO**
furbot	Classes de interface e que representam os objetos do mundo.
exceptions	Classes responsáveis pelo tratamento de exceções específicas do *framework*.
suporte	Classes com funcionalidades diversas utilizadas quando programa é executado ou reiniciado.
adicionais	Classes que foram criadas onde não havia equivalência de classes em Java e Objective-C.

Quadro 23 – Detalhamento dos pacotes do *framework* iFurbot

A Figura 24 demonstra através de um diagrama de pacotes, os pacotes de classes contidos no iFurbot.

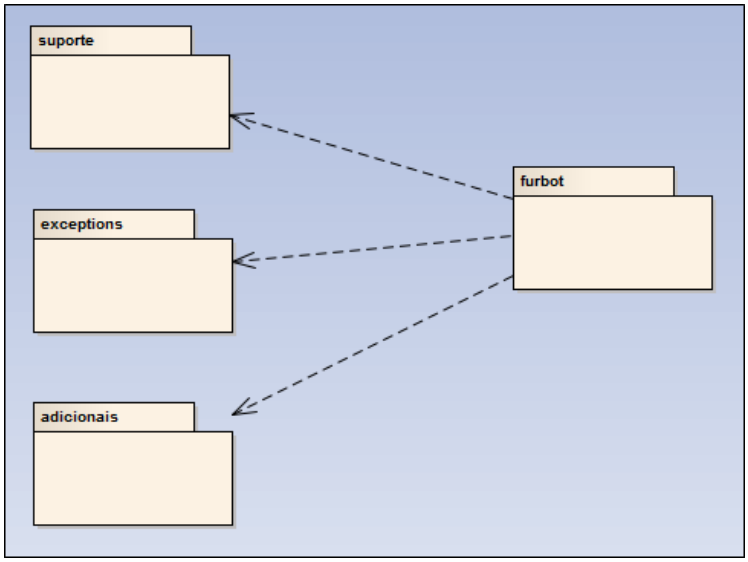


Figura 24 – Representação dos pacotes contidos no iFurbot

3.2.4 Diagrama de classes

A Figura 25 demonstra um diagrama de classes com as principais classes contidas no pacote `furbot` já convertidas para Objective-C.

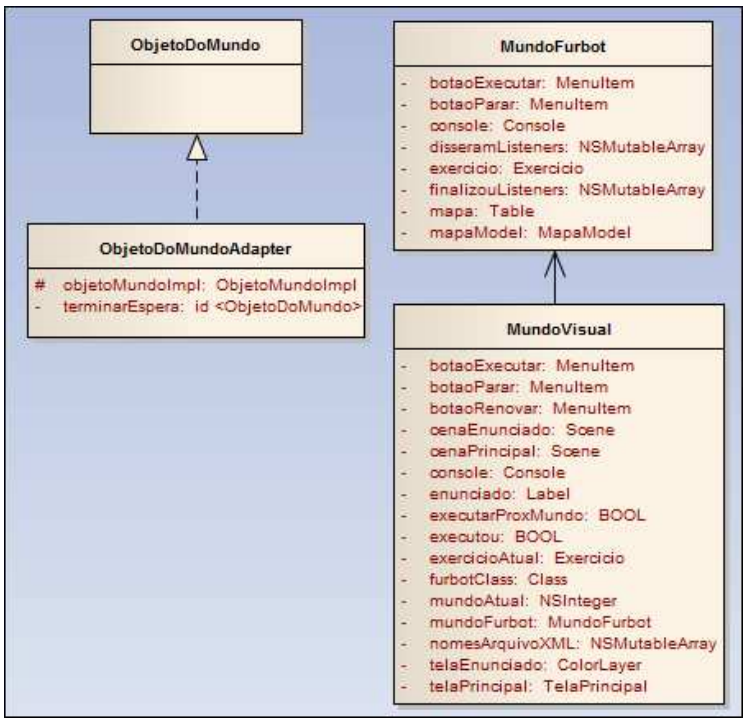


Figura 25 – Diagrama de classes do pacote `furbot` em Objective-C

Os métodos em Java e os mesmos já convertidos em Objective-C das classes `ObjetoDoMundoAdapter`, `MundoFurbot` e `MundoVisual` são detalhados no apêndice B, sendo o Quadro 42 para a classe `ObjetoDoMundoAdapter` , o Quadro 43 para `MundoFurbot` e o Quadro 44 para o `MundoVisual`.

A Figura 26 demonstra um diagrama de classes com as principais classes contidas no pacote `suporte` já convertidas para Objective-C.

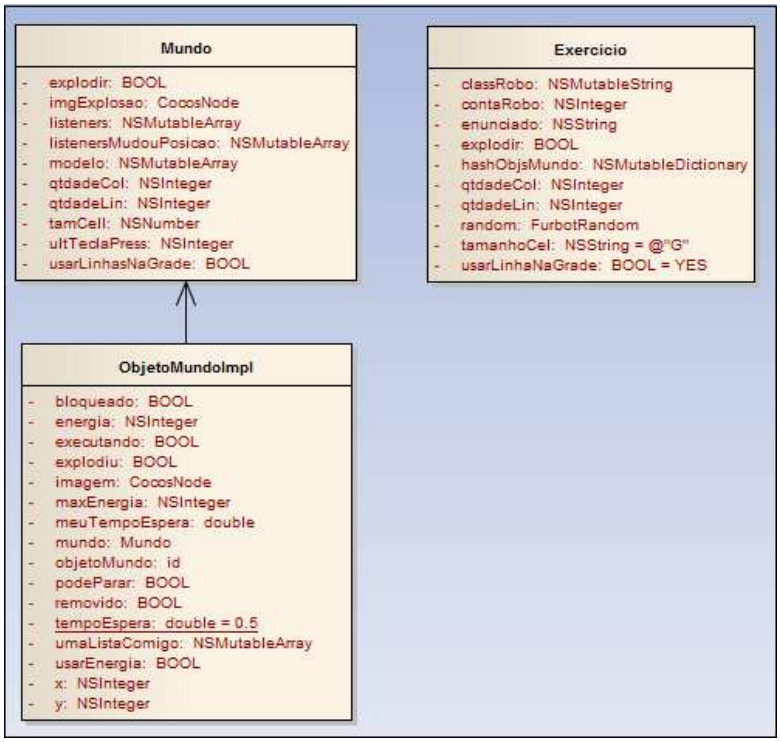


Figura 26 - Diagrama de classes do pacote `suporte` em Objective-C

Assim como as principais classes do pacote suporte, as classes `Mundo`, `ObjetoMundoImpl` e `Exercicio` são detalhadas no Quadro 45, Quadro 46 e Quadro 47 nessa ordem no apêndice B.

A Figura 27 demonstra um diagrama de classes com as principais classes contidas no pacote `adicionais` em Objective-C.

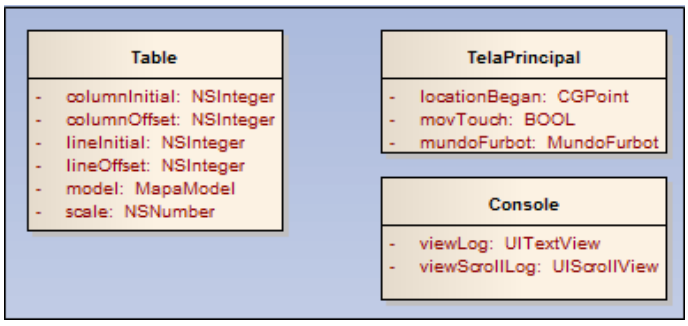


Figura 27 - Diagrama de classes do pacote `adicionais` em Objective-C

3.3 IMPLEMENTAÇÃO

A seguir é demonstrada a utilização do *framework* em projetos usando XCode e mais adiante a operacionalidade da implementação.

3.3.1 Usando o *framework* em projetos no XCode

Considerando que o *framework* será utilizado por desenvolvedores que estão iniciando na área relacionada à programação, foi desenvolvido um template para criação de projetos na aplicação XCode com o intuito de automatizar as tarefas básicas de associação da *framework* iFurbot, além de algumas configurações relacionados ao projeto.

O processo é iniciado com a solicitação do usuário para criação de um novo projeto conforme é demonstrado na Figura 28.

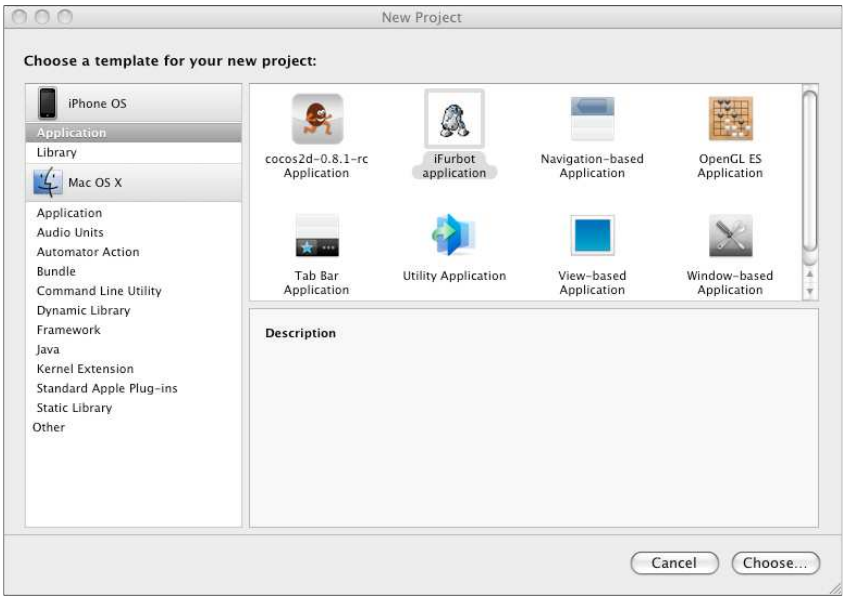


Figura 28 – Criação de uma nova aplicação usando iFurbot

Após selecionado o tipo de projeto `iFurbot application`, o usuário avança para a próxima tela onde deverá ser informado o nome do projeto. O nome do projeto deve ser igual ao nome do arquivo de configuração, do exercício passado pelo professor. Finalizando esse passo, o projeto é criado e demonstrado para o usuário conforme Figura 29.

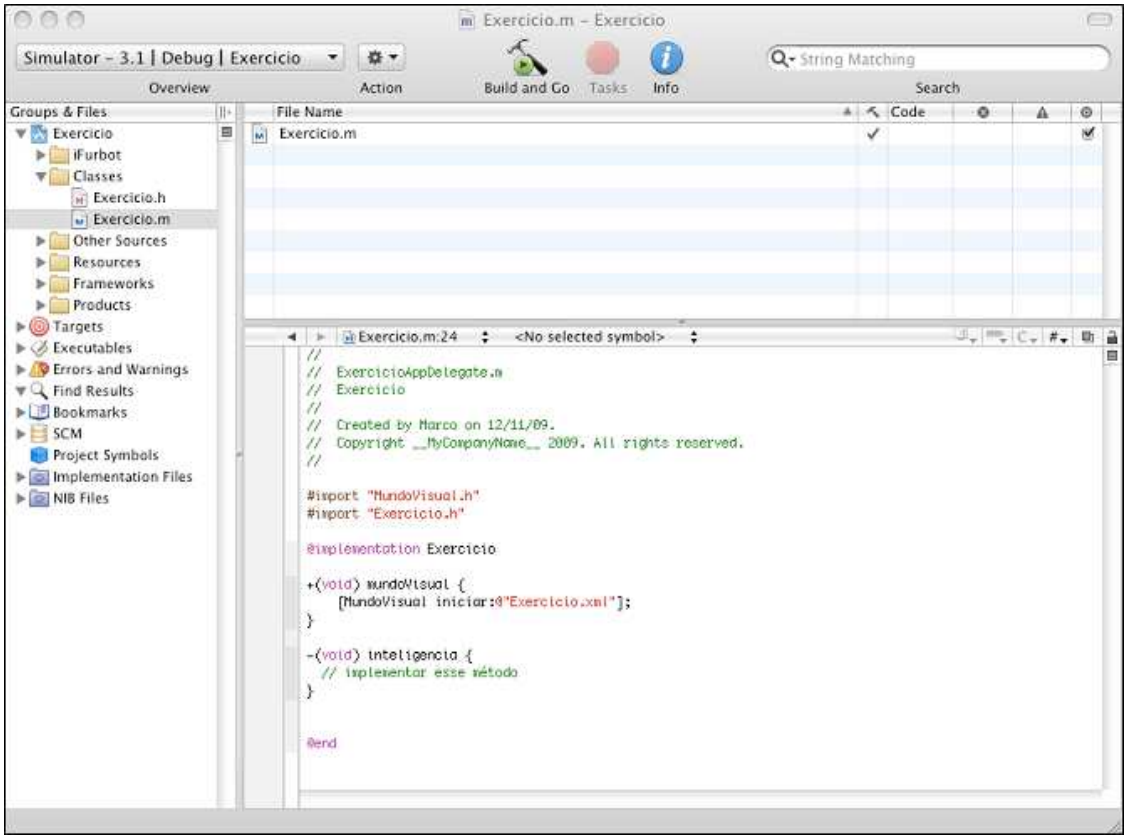


Figura 29 – Projeto `Exercicio` no XCode

Para prosseguir na implementação o desenvolvedor deve associar o arquivo de configuração XML ao projeto e codificar o método `inteligencia`.

3.3.2 Operacionalidade da implementação

Nessa seção é explicitado os recursos utilizados e soluções adotadas para atender os requisitos propostos.

3.3.2.1 Interface com o usuário

O iFurbot foi criado com o objetivo de demonstrar todas as informações existentes na sua versão em Java. A sua versão criada para o iPhone é dividida em 3 telas, sendo elas: tela de introdução, tela principal e tela de *log*. Na tela de introdução é apresentado o enunciado do exercício repassado pelo professor no arquivo XML e um botão que permite avançar para a tela principal. Na tela principal é apresentado o mundo Furbot e botões de ação que permitem:

a) botão E – executar a aplicação;
b) botão R – reiniciar a aplicação;
c) botão P – parar a execução;
d) botão I – ir para a tela de introdução;
e) botão L – ir para a tela de *log*.

Ainda nessa tela é possível simular a funcionalidade do componente `Slider` do Furbot em Java. Tocando na tela, e arrastando o dedo para a esquerda faz com que a velocidade de execução do mundo aumente e para a direita que diminua.

Na tela de *log* é possível visualizar as mensagens enviadas pelos objetos utilizando o método `diga` do `objetoDoMundoAdapter` e voltar para a tela principal através do botão principal.

3.3.2.2 Interface *multi-touch*

Para atender o requisito de mapeamento de interação via teclado para interação de interface *multi-touch*, foi necessária a criação da imagem de um direcional, onde através de regiões mapeadas nessa imagem, são simuladas as teclas de direita, esquerda, acima e abaixo de um teclado. Essa imagem foi dividida em regiões de toque e quando pressionadas, acionam um evento de acordo com a região. A imagem a da Figura 30 demonstra o direcional utilizado no desenvolvimento e duas outras imagens destacando regiões do direcional.

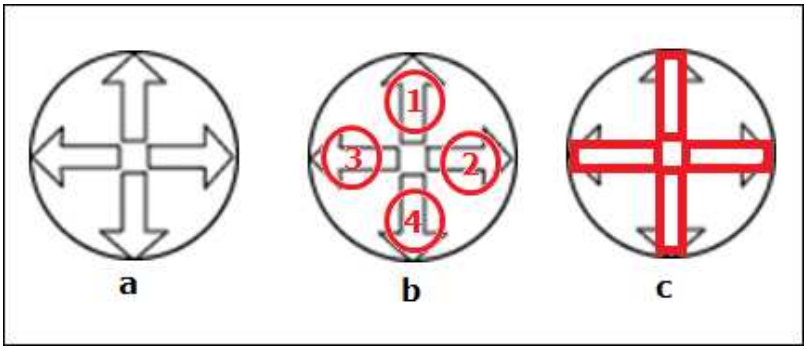


Figura 30 – Direcional utilizado no iFurbot

Referente a imagem b da Figura 30, tem-se que, ao ocorrer o evento de toque na região de número 1 é executado o método `andarAcima`, ao ocorrer na região 2 executa-se o método `andarDireita`, na região 3 o método `andarEsquerda` e na região 4 o método `andarAbaixo`. Na imagem c estão as possíveis regiões mapeadas de toque. Ao ser acionado o evento `ccTouchesBegan` indicando um toque na interface do dispositivo pelo usuário, é analisado se

o ponto de toque está dentro de alguma dessas regiões. O Quadro 24 demonstra a rotina que faz essa verificação.

```
-(BOOL) verificaTeclaPressionada:(NSInteger) tecla
pontoPressionado:(CGPoint) ponto {
	CGRect rect;

	switch (tecla) {
		case ACIMA:
			rect = CGRectMake(47, 55, 9, 30);
			break;
		case ABAIXO:
			rect = CGRectMake(47, 15, 9, 30);
			break;
		case ESQUERDA:
			rect = CGRectMake(16, 45, 30, 9);
			break;
		case DIREITA:
			rect = CGRectMake(56, 45, 30, 9);
			break;
		default:
			break;
	}
	return CGRectContainsPoint(rect, ponto);
}
```

Quadro 24 – Código para verificação de toque no direcional

Durante a implementação do direcional levou-se em conta que o mesmo não seria utilizado em todas as situações. Por isso, criou-se o método estático `usarTeclas` da classe `MundoVisual` para adicionar o mesmo à tela principal do iFurbot, além de efetuar algumas configurações como o bloqueio do recurso de rotação do dispositivo, obrigando o usuário a utilizá-lo apenas em modo paisagem. O modo paisagem foi escolhido por permitir uma maior facilidade de uso da aplicação que requer o direcional. A Figura 31 demonstra o direcional sendo utilizado em um exercício do iFurbot.

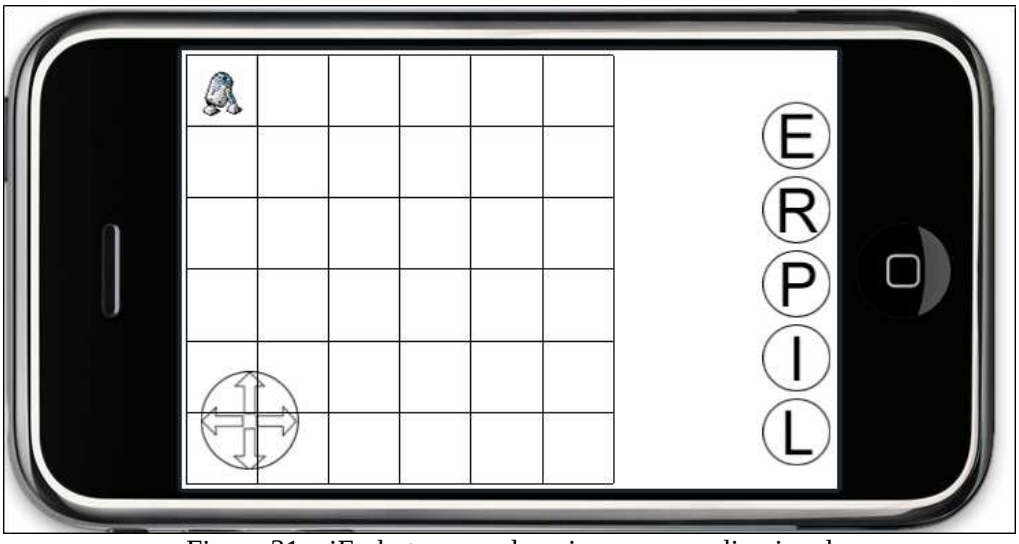


Figura 31 – iFurbot em modo paisagem com direcional

3.3.2.3 Acelerômetro

A utilização do acelerômetro permite obter a posição exata que o dispositivo se encontra em um determinado momento, além de receber avisos de alteração de posição através da execução do método `addObserver` do objeto de `NSNotificationCenter`.

Ao receber o aviso de alteração de posição do dispositivo, o iFurbot reposiciona os componentes de acordo com sua nova posição. No caso dos componentes da tela de introdução e da tela principal é necessário apenas esse reposicionamento já que ambas foram construídas utilizando apenas recursos do Cocos2d como `Layer`, `Scene` e `Sprite` deixando boa parte do controle de rotação por conta do *framework*. O Quadro 25 demonstra o código da tela principal responsável por reposicionar cada componente.

```
 1- -(void) atualizaPosicoesTelaPrincipal {
 2-
 3-  CGSize dimensoes = [[Director sharedDirector] winSize];
 4-  if ([[Director sharedDirector] deviceOrientation] ==
                                                CCDeviceOrientationPortrait){
 5-   [menuTelaPrincipal alignItemsHorizontallyWithPadding: 4];
 6-   [menuTelaPrincipal setPosition: ccp(dimensoes.width-180,50)];
 7-   [telaPrincipal changeWidth: dimensoes.width height: dimensoes.height];
 8-  } else {
 9-   [menuTelaPrincipal alignItemsVerticallyWithPadding: 4];
10-   [menuTelaPrincipal setPosition: ccp(dimensoes.width-30,150)];
11-   [telaPrincipal changeWidth: dimensoes.width height: dimensoes.height];
12-
13-  }
14- }
```

Quadro 25 – Método `atualizaPosicoesTelaPrincipal`

Antes de fazer o reposicionamento dos componentes, é verificada a posição que o dispositivo está conforme demonstrado na linha 4 do Quadro 25. Após essa verificação são aplicadas as novas posições, além da mudança de largura da tela já que, no modo retrato a altura é maior do que a largura e no modo paisagem ocorre o contrário.

A tela de console foi construída utilizando componentes nativos oferecidos pela camada Cocoa touch. No desenvolvimento dessa interface, foi utilizado o componente `UITextView` para permitir que o texto fosse adicionado dinamicamente sem limites de dimensões, já que o componente oferece um recurso de *scroll*. Além desse, foi adicionado também um componente `UIButton` para permitir voltar a tela principal. A classe utilizada responsável pela interface foi a `UIView`.

Como o desenvolvimento desse projeto tomou como base a utilização do framework Cocos2d para gerenciar as interfaces visuais, foi necessário que a tela de console funcionasse

a partir desse framework. O Cocos2d ao receber aviso de que houve rotação do dispositivo, propaga esse aviso aos componentes de sua framework além de rotacionar a sua interface base. Essa propagação não é submetida aos componentes da Cocoa touch fazendo assim com que a interface da tela de console não seja rotacionada. Para permitir a rotação da tela de console foi necessária a criação de um algoritmo conforme é demonstrado no Quadro 26.

```
 1- -(void)atualizarPosicoes {
 2-
 3-  CATransform3D rotationTransform = CATransform3DIdentity;
 4-  [console.layer removeAllAnimations];
 5-  UIDeviceOrientation interfaceOrientation = [[UIDevice currentDevice]
                                                              orientation];
 6-  CGSize dimensoes = [[Director sharedDirector] winSize];
 7-
 8-  CGRect contentRect = CGRectMake(0, 0, dimensoes.width, dimensoes.height);
 9-  console.bounds = contentRect;
10-
11-  if (interfaceOrientation == UIInterfaceOrientationPortrait) {
12-     rotationTransform = CATransform3DRotate(rotationTransform, 3.14159*2,
                                          0.0,0.0, 1);
13-     console.layer.transform = rotationTransform;
14-     botaoTelaPrincipal.frame = CGRectMake(dimensoes.width/2-80,
                                         dimensoes.height-40, 150, 30);
15-     viewLog.frame = CGRectMake(0, 0, dimensoes.width, dimensoes.height-50);
16-  } else {
17-     if (interfaceOrientation == UIInterfaceOrientationLandscapeRight)
18-       rotationTransform = CATransform3DRotate(rotationTransform, 3.14159/2,
                                           0.0, 0.0, 1);
19-     else
20-       rotationTransform = CATransform3DRotate(rotationTransform,
                             3.14159+(3.14159/2), 0.0, 0.0, 1);
21-
22-     console.layer.transform = rotationTransform;
23-     botaoTelaPrincipal.frame = CGRectMake(dimensoes.width/2-80,
                                         dimensoes.height-40, 150, 30);
24-     viewLog.frame = CGRectMake(0, 0, dimensoes.width, dimensoes.height-50);
25-  }
26- }
```

Quadro 26 – Método `atualizarPosicoes`

Nas linhas, 12, 18 e 20 é instanciado um objeto do tipo `CATransform3D` que é utilizado na rotação do dispositivo através da chamada da função `CATransform3Drotate`. No segundo parâmetro dessa função deve ser informado o ângulo para o qual a interface deve ser rotacionada. Após a execução da rotação a tela é redimensionada de acordo com a nova posição do dispositivo.

3.3.2.4 Multi-threads

Na conversão do *framework* levou-se em consideração a necessidade do uso de *threads*, uma vez que esse recurso estava disponível na versão em Java. A linguagem Objective-C oferece recursos nativos que permitem essa funcionalidade, não obstante, o

framework Cocos2d não oferece suporte a *multi-threads*. Para resolver essas duas situações é utilizado o método `performSelectorOnMainThread` da classe `NSThread` para centralizar funções relacionadas ao *framework* Cocos2d na *thread* principal, não havendo assim possibilidade de ocorrer erro na interface gráfica ocasionados pela Cocos2d e permitindo o recurso de *multi-threads* em exercícios utilizando o *framework*.

3.3.2.5 *Exercício* `DeixaRastro`

Parar explicitar o funcionamento do *framework* Furbot é realizado um comparativo entre as duas implementações do *framework* usando o exercício `DeixaRastro` como exemplo. O arquivo XML do exercício pode ser visto no Quadro 27.

```
<furbot>
  <enunciado>
    Faça o furbot andar pelo mundo em ziguezague horizontal
e deixar um rastro com o número do passo que ele executou.
  </enunciado>
  <mundo>
   <qtdadeLin>6</qtdadeLin>
   <qtdadeCol>6</qtdadeCol>
   <tamanhoCel>P</tamanhoCel>
   <explodir>false</explodir>
  </mundo>
  <robo>
   <x>0</x>
   <y>0</y>
   <explodir>false</explodir>
  </robo>
</furbot>
```

Quadro 27 – Arquivo `DeixaRastro.xml`

Para resolver o exercício foi necessário criar atributos conforme é possível visualizar no Quadro 28.

`private int contaPassos;`
`private Numero n;`
	`@private`
	`  NSInteger contaPassos;`
	`  Numbero *n;`

Quadro 28 – Declaração dos atributos utilizados no exercício `DeixaRastro`

Ainda, foi necessário a criação de uma nova classe que herda da classe `Furbot` e a codificação do método `inteligencia` como pode ser visto no Quadro 29.

```
public void inteligencia() {
  boolean repetir = true;

  while (repetir) {
    while (!ehFim(DIREITA)) {
      contar();
      andarDireita();
    }
    if (!ehFim(ABAIXO)) {
      contar();
      andarAbaixo();
      while (!ehFim(ESQUERDA)) {
        contar();
        andarEsquerda();
      }
      if (!ehFim(ABAIXO)) {
        contar();
        andarAbaixo();
      } else {
        repetir = false;
      }
    } else {
        repetir = false;
    }
  }
}
```

```
- (void)inteligencia {
  BOOL repetir = YES;

  while (repetir) {
    while (![self ehFim: DIREITA]) {
       [self contar];
       [self andarDireita];
    }
    if (![self ehFim: ABAIXO]) {
       [self contar];
       [self andarAbaixo];
       while (![self ehFim: ESQUERDA]) {
         [self contar];
         [self andarEsquerda];
       }
       if (![self ehFim: ABAIXO]) {
         [self contar];
         [self andarAbaixo];
       } else {
         repetir = NO;
       }
    } else {
       repetir = NO;
    }
  }
}
```

Quadro 29 – Método `inteligencia` em Java e Objective-C

Para permitir que o robô deixasse rastro a cada novo passo, foi criado um novo método chamado `contar` que é responsável por instanciar um objeto de `Numero` e incluí-lo no mundo, toda vez que o robô mudar de posição. O Quadro 30 demonstra as duas versões do método `contar`.

```
public void contar() {
  contaPassos++;
  n = new Numero();
  n.setValor(contaPassos);
  this.adicionarObjetoNoMundo(n,
AQUIMESMO);
}
```

```
- (void)contar {
  contaPassos++;
  n = [[[Numero alloc] init] autorelease];
  [n setValor: contaPassos];
  [self adicionarObjetoNoMundo: n pDirecao:
AQUIMESMO];
}
```

Quadro 30 – Método `contar` em Java e Objective-C

Para que, ao iniciar a aplicação, seja carregada a interface visual seguindo os critérios definidos no arquivo de configuração, é necessário a execução do método `iniciar` passando o arquivo XML como parâmetro como mostra o Quadro 31.

```
public static void main(String args[]){
 MundoVisual.iniciar(“DeixaRastro.xml”);
}
```

```
+(void) mundoVisual {
  [MundoVisual iniciar:
              @”DeixaRastro.xml”];
}
```

Quadro 31 – Iniciando `MundoVisual` em Java e Objective-C

Para que fosse possível a inicialização da interface visual em Objective-C sem que houvesse a necessidade da criação de um objeto para isso e também para que não fosse necessária a alteração de várias classes para fazer apenas um exercício, criou-se um método padrão estático `mundoVisual` que sempre é chamado ao iniciar a aplicação. Ao final do desenvolvimento do exercício, tem-se como resultado a aplicação desenvolvida, conforme

mostra a Figura 32 e a Figura 33.

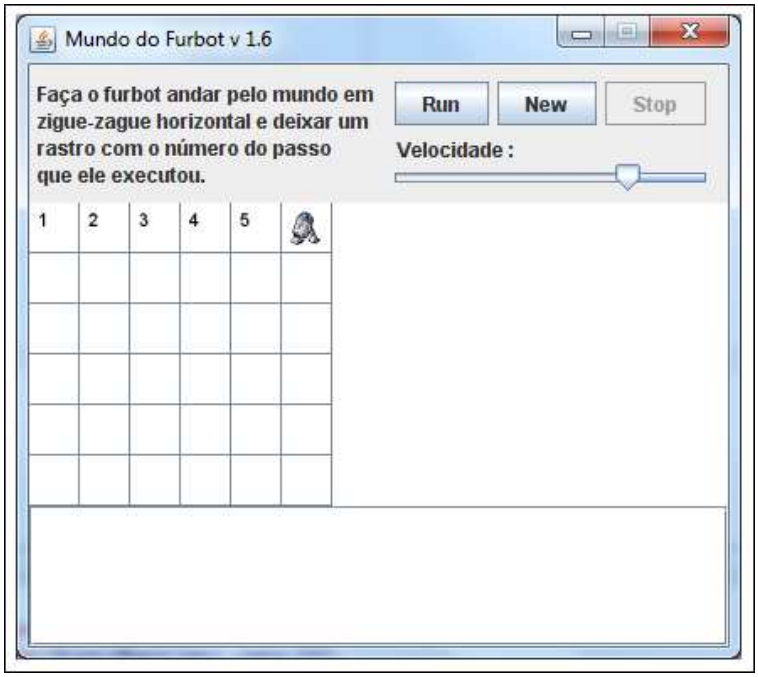


Figura 32 – `DeixaRastro` executando em Java

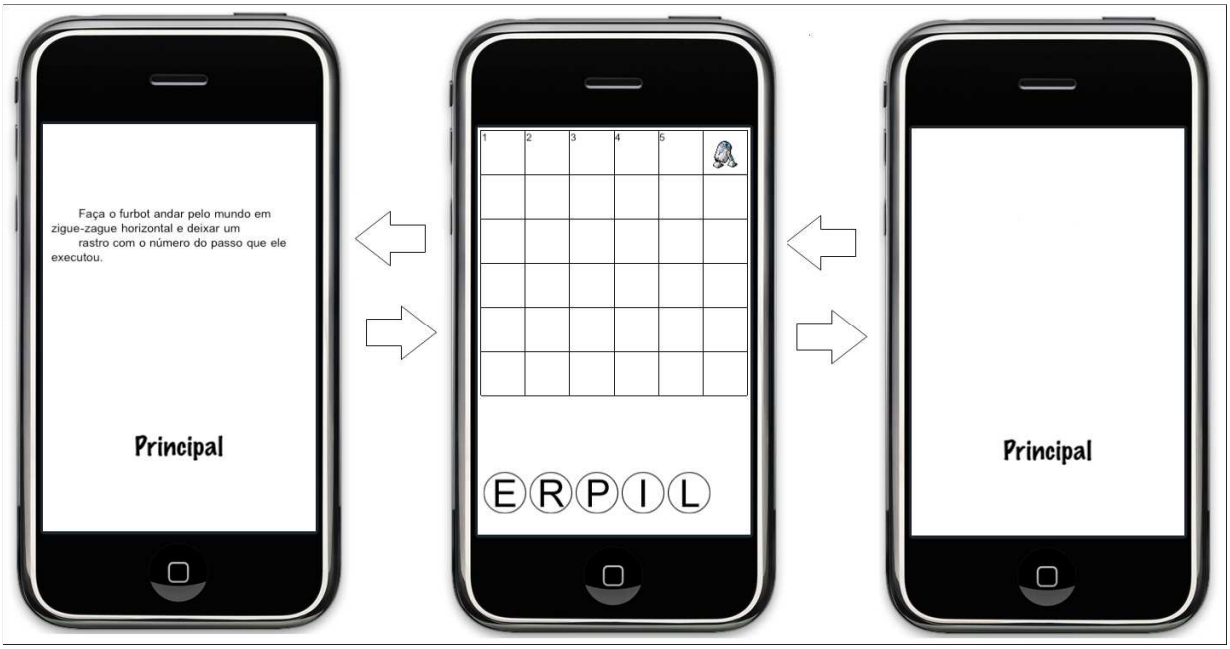


Figura 33 – `DeixaRastro` em execução no iPhone

Na Figura 33, a tela apresentada mais a direita é a tela de console que no caso desse exercício não foi utilizada, pois o robô não precisou executar o método `diga`.

3.3.2.6 Exercício Space Invader

Para demonstrar a utilização do direcional e do uso de *multi-threads* no iFurbot foi utilizado um exercício já desenvolvido anteriormente na versão em Java para a versão em Objective-C. O exercício `Space Invader` propõe que o aluno desenvolva um mundo onde o robô é representado pela imagem de um foguete que pode ser movimentado durante a sua execução. Além do robô são incluídos no mundo inimigos em um tempo determinado, que se em contato com robô finalizam a execução do exercício. É possível através do robô disparar tiros em direção aos inimigos que quando atingidos, tem sua imagem substituída pela imagem de uma explosão. Logo após isso o inimigo é retirado do mundo.

Para resolver o exercício, assim como na sua versão em Java, foi necessário a criação de três classes sendo elas: `Interacao`, `Inimigo` e `Tiro`. Na classe `Interacao` foi codificada a inteligência do robô e pode ser visualizada no Quadro 32.

```
01- -(void) inteligencia {
02- [self setTempoEspera: 0];
03- NSInteger nascAlien = 0;
04- Direcao ultDirecao = DIREITA;
05- while (vivo) {
06-   NSInteger tecla = [self ultimaTeclaPress];
07-
08-   // os inimigos nascem a cada 1 milhão de vezes
09-   if (nascAlien == 1000000) {
10-     [self nascerAlien];
11-     nascAlien = 0; // reinicia a contagem
12-   }
13-
14-   nascAlien++;
15-
16-   if (tecla != 0) {
17-     indImag = tecla;
18-     [self repintar];
19-   }
20-
21-   switch (tecla) {
22-    case ACIMA: ultDirecao = [self andar: ACIMA]; break;
23-    case ABAIXO: ultDirecao = [self andar: ABAIXO]; break;
24-    case DIREITA: ultDirecao = [self andar: DIREITA];break;
25-    case ESQUERDA: ultDirecao = [self andar: ESQUERDA]; break;
26-    case ESPACO:
27-    [self adicionarObjetoNoMundo: [[[Tiro alloc] initWithParams: ultDirecao]
autorelease] pDirecao: AQUIMESMO];
28-    break;
29-   }
30- }
31-
32- indImag = 4;
33- [self pararInimigos];
34-}
```

Quadro 32 – Classe `Interacao`

A execução do método `inteligencia` consiste em verificar se alguma tecla foi pressionada e interagir com a mesma quando isso ocorrer. Para que fosse possível fazer com o que robô atirasse, foi necessário adaptar a diretiva `ESPACO`, onde na versão em Java está atrelada a tecla espaço do teclado, para o *touch* do dispositivo. Ao pressionar a tela enquanto o exercício estiver em execução, e a região pressionada não estiver dentro das regiões reservadas para o direcional, será registrado que a última tecla pressionada foi espaço, permitindo assim que a verificação seja realizada no desenvolvimento do exercício e seja aplicada uma funcionalidade a isso.

A classe `Inimigo` é responsável pela codificação do objeto que será incluído no mundo quando o contador `nascAlien` atingir um milhão conforme linha 09 do Quadro29. A codificação do método `executar` dessa classe pode ser visualizada no Quadro 33.

```
01- -(void) executar {
02-  [self esperar: 1];
03-  while (!matou && ![self ehFim: direcao]) {
04-    ObjetoDoMundoAdapter *obj = (ObjetoDoMundoAdapter *)[self objeto:
direcao];
05-    if (obj != nil) {
06-      if ([[obj souDoTipo] isEqualToString:@"Tiro"]) {
07-        [self setTempoEspera: 2];
08-        break;
09-      }
10-      if ([[obj souDoTipo] isEqualToString:@"Inimigo"]) {
11-        continue;
12-      }
13-      Interacao *i = (Interacao *)obj;
14-      [i matar];
15-    }
16-
17-    [self setTempoEspera: tempoEspera];
18-    switch (direcao) {
19-      case ACIMA:    [self andarAcima];  break;
20-      case ABAIXO:   [self andarAbaixo]; break;
21-      case ESQUERDA: [self andarEsquerda]; break;
22-      case DIREITA:  [self andarDireita]; break;
23-      case AQUIMESMO: break;
24-      default: break;
25-    }
26-    [self setTempoEspera: 1];
27-  }
28-  if (matou) {
29-    [self repintar];
30-    [self esperar: 1];
31-  }
32-  [Interacao retirarInimigo: self];
33-  [self removerMe];
34-  vivo = NO;
35- }
```

Quadro 33 – Método executar da classe `Inimigo`

O objetivo da classe `Inimigo` é fazer com que o objeto caminhe pelo mundo do iFurbot verificando se foi atingido por um tiro, outro inimigo, ou ainda o robô. Essas verificações podem ser observadas da linha 04 à 15 do Quadro 33.

A classe `Tiro` faz com que o objeto caminhe até o final do mundo sempre utilizando a direção no qual foi iniciado. A cada nova posição do objeto é realizada uma verificação por objetos do tipo `Inimigo` que estejam na mesma posição. Essa verificação pode ser visualizada das linhas 04 à 10 do Quadro 34.

```
01- -(void) executar {
02-
03-  while (![self ehFim: direcao]) {
04-   id<ObjetoDoMundo> obj = [self objeto: direcao];
05-   if (obj != nil) {
06-     if ([[obj souDoTipo] isEqualToString:@"Inimigo"]) {
07-    Inimigo *iv = (Inimigo *)obj;
08-    [iv matar];
09-    break;
10-     }
11-   }
12-   switch (direcao) {
13-     case ABAIXO:
14-    [self andarAbaixo];
15-    break;
16-     case ACIMA:
17-    [self andarAcima];
18-    break;
19-     case DIREITA:
20-    [self andarDireita];
21-    break;
22-     case ESQUERDA:
23-    [self andarEsquerda];
24-    break;
25-     default:
26-    break;
27-   }
28-  }
29-  [self removerMe];
30- }
```

Quadro 34 - Método executar da classe `Tiro`

Para atender esse exercício levando em consideração a sua funcionalidade baseada em *multi-threads* foi necessário retirar a adição de novas imagens dos métodos `initWithParams` e `executar` e colocá-los no método `buildImage` para que fosse executado sempre pela *thread* principal. A Figura 34 demonstra a interface do exercício.

Figura 34 – Interface do exercício `Space Invader`

3.4 RESULTADOS E DISCUSSÃO

As linguagens Java e Objective-C possuem várias semelhanças, principalmente devido a sua origem ser a linguagem C e C++. A complexidade encontrada em Objective-C deve-se a sua sintaxe diferenciada. Enquanto em Java é utilizada uma sintaxe de declarações de métodos que é muito semelhante às funções C, Objective-C, por outro lado, utiliza uma sintaxe que é muito mais próxima de Smalltalk utilizando o conceito de mensagens.

Ainda, em Objective-C há a necessidade de um controle mais rígido da memória alocada com o uso das diretivas, `retain` e `release`, o que dificulta a conversão já que em Java esse tratamento não é necessário por utilizar *garbage collector*. Em algumas situações onde não houve interação *multi-thread*, foi utilizado o objeto `NSAutoReleasePool` para gerenciamento da memória.

Durante a conversão do Furbot levou-se em consideração que haveriam manutenções em ambas as versões além de futuras implementações. Sendo assim, foi mantida a nomenclatura das classes, métodos e variáveis tentando minimizar a complexidade das alterações deixando apenas a sintaxe e os objetos oferecidos pelas linguagens como diferencial.

Com a migração do Furbot para Objective-C foram criadas novas classes que possuem funcionalidades não existentes nos *frameworks* oferecidos pelo iPhone SDK. Essas classes foram separadas das demais através do grupo adicionais. Além disso, devido a integração do iFurbot com o *framework* Cocos2d, foram necessárias alterações em algumas classes que antes eram consideradas como *views* no conceito de MVC para *controllers* como é o caso da classe `MundoVisual` que agora serve como gerenciador de janelas e eventos.

O uso do *framework* Cocos2d permitiu um alto grau de abstração no desenvolvimento das interfaces visuais com o uso das classes `Layer` e `Scene`. Porém, em algumas situações onde foi necessária a utilização de *threads*, ouve a necessidade de direcionar as interações da Cocos2d à *thread* principal já que o *framework* não permite o recurso *multi-thread*.

O desenvolvimento do acelerômetro teve como principal dificuldade a adaptação da janela de *log* para a sua utilização uma vez que, a janela utiliza o componente `UIScrollView` que não é compatível com o tratamento de rotação efetuado pelo framework Cocos2d. Para tornar funcional a rotação dessa janela foi necessária a criação de uma rotina específica.

Para permitir a mudança de velocidade da execução do exercício, assim como era possível através do `Slider` em Java, vinculou-se a funcionalidade ao movimento do dedo

sobre a tela na direção direita para aumentar a velocidade e esquerda para diminuir.

No desenvolvimento do direcional foi necessário o mapeamento das regiões onde estavam as setas da imagem mesmo. Durante os testes percebeu-se que em alguns momentos além da chamada do evento que registrava a seta pressionada era modificada a velocidade da execução do exercício. Como a idéia principal era permitir a mudança de velocidade em qualquer região da tela, mas que isso poderia se tornar prejudicial para os testes dos exercícios, conclui-se que a solução mais adequada seria aumentar o desvio necessário para a esquerda ou direita para ocorrer a mudança de velocidade.

O Quadro 35 demonstra um comparativo dos trabalhos correlatos com o iFurbot.

Aplicação	**Voltado para o ensino**	**Aprendizado de algoritmos**	**iPhone**	**Jogo**
Greenfoot	X	X		
Gorillas			X	X
M-Geo	X		X	
Skattjäkt	X			X
Sist. de gerenc. de objetos de aprendi. para dispositivos móveis	X			
iFurbot	X	X	X	

Quadro 35 – Comparativo do iFurbot com os trabalhos correlatos

4 CONCLUSÕES

O objetivo desse trabalho de converter o *framework* Furbot desenvolvido utilizando a tecnologia Java para Objective-C foi concluído com êxito. Com o *framework* já convertido, é possível a criação de aplicações para iPhone através da codificação do método `inteligencia` do robô e o uso do iPhone Simulator para efetuar testes na aplicação.

Para que fosse possível a execução e implementação de exercícios considerando o baixo conhecimento de programação do aluno, foi desenvolvido um template no XCode com o intuito de diminuir a complexidade na sua utilização.

Tendo em vista que o foco desse trabalho implica na utilização direta do iPhone SDK, não foi considerado o uso de *garbage collector* oferecido pelo Objective 2.0.

A aplicação Gorillas desenvolvida em Cocos2d, foi utilizada como fonte de informações para dúvidas relacionadas ao uso do *framework*. Os demais trabalhos correlatos foram utilizados como base de conhecimento de experiências já vivenciadas com alunos em trabalhos educacionais envolvendo dispositivos móveis.

Com o uso da ferramenta ObjectAlloc disponível na aplicação Instruments foi possível diminuir consideravelmente os problemas de vazamento de memória. Ainda assim, apesar da conversão ter sido finalizada e os vazamentos terem sido tratados, em alguns pontos do *framework*, o controle de memória ainda pode ser aperfeiçoado.

O Interface Builder foi utilizado no desenvolvimento da interface de *log* onde houve a necessidade da criação de uma tela com recurso *scroll* enquanto nas demais interfaces foi utilizado o *framework* Cocos2d.

Um dos aspectos que difere o desenvolvimento para iPhone de Java é que a Apple só permite o envio de aplicações para o dispositivo, se o usuário estiver cadastrado no programa de desenvolvimento da Apple ou ainda através da aquisição do mesmo na App Store. Isso limita a possibilidade de testar aplicações desenvolvidas com o iFurbot usando funcionalidades dependentes de dispositivo com o acelerômetro.

Um aspecto importante a destacar é que um resumo do trabalho foi publicado em um evento científico com banca de avaliação interinstitucional onde foram submetidos 80 trabalhos e selecionados apenas 17.

4.1 EXTENSÕES

Apesar do iFurbot usar funcionalidades oferecidas pelo *framework* Cocos2d, existe ainda uma grande gama de funções que podem ser adicionadas no iFurbot como o uso de animações de `sprites`, ações de rotação, escalonamento ou ainda o uso de extensões do Cocos2d como o Cocoslive que oferece recursos de pontuação on-line.

O recurso de acelerômetro utilizado no trabalho pode ser melhorado uma vez que sua funcionalidade hoje consiste apenas em redesenhar a tela quando é movimentado o dispositivo. Como sugestão pode-se considerar o uso do recurso para criar novos eventos no mundo, como terremotos ou até mesmo usá-lo como direcional para movimentar o robô.

A funcionalidade *multi-touch* não foi muito utilizada nesse trabalho tendo em vista as limitações existentes na versão do Furbot em Java. Através da aplicação de melhorias no *framework* iFurbot, é possível com o uso do *multi-touch* permitir a movimentação de dois robôs ou mais separando a tela do dispositivo em regiões.

A interface do iFurbot foi criada apenas com o objetivo de atender os requisitos do trabalho. Por esse motivo, sugere-se que a mesma seja melhorada, uma vez que, o público que utiliza o dispositivo iPhone e iPod touch é exigente quanto aos recursos visuais e o ambiente de desenvolvimento disponibiliza ferramentas e recursos para isso.

Como sugestão para trabalhos futuros, é possível ainda estender a versão do Furbot que agora está disponível em Java e Objective-C para a linguagem LUA que também é baseada na linguagem de programação C e possui suporte a programação orientada a objetos.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE. **iPhone Dev Center – Apple developer connection.** [S.l.], 2009. Disponível em: <http://developer.apple.com/iphone/>. Acesso em: 20 jul. 2009.

BRANNAN, James A. **iPhone SDK programming**: a beginner's guide. New York: McGraw Hill Professional, 2009.

BUCANEK, James. **Learn Objective-C for Java developers**. New York: Apress, 2009.

CAMPBELL, Duncan. **iPhone SDK 3**: visual quickstart guide. Berykeley: Peachpit Press, 2009.

CELEKT. **Center for learning and knowledge technologies**. Växjö, 2007. Disponível em: <http://www.celekt.info/max/>. Acesso em: 15 out. 2009.

COCOS2D. **Cocos2d for iPhone**. [S.l.], 2009. Disponível em: <http://www.cocos2d-iphone.org>. Acesso em: 30 ago. 2009.

DALRYMPLE, Mark; KNASTER, Scott. **Learn Objective-C on the Mac**. New York: Apress, 2009.

FRANCISCATO, Fabio T.; MEDINA Reseclea D. Sistema de gerenciamento de objetos de aprendizagem para dispositivos móveis. **RENOTE – Revista Novas Tecnologias na Educação**, Porto Alegre, v. 7, n. 1, não paginado, jul. 2009.

GOLDSTEIN, Neal. **Objective-C for dummies**. Indiana: Wiley Publishing, 2009.

GONDIM, Haller W. A. S.; AMBRÓSIO, Ana P. Esboço de fluxogramas no ensino de algoritmos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 28., 2008, Belém. **Anais...** Belém: SBC, 2008. p. 1-9.

GORILLAS. **Gorillas - the iPhone game**. [S.l.], 2009. Disponível em: <http://gorillas.lhunath.com/>. Acesso em: 30 ago. 2009.

GRAHAM, Dorothy et al. **Foundations of software testing**: ISTQB certification. 2th ed. London: Cengage Learning EMEA, 2008.

GREENFOOT. **Greenfoot**. Melbourne, 2008. Disponível em: <http://www.greenfoot.org/index.html>. Acesso em: 20 jul. 2009.

MARÇAL, Edgar et al. O uso de dispositivos móveis para auxiliar a aprendizagem significativa na geometria espacial. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 29., 2009, Bento Gonçalves. **Anais...** Bento Gonçalves: CSBC, 2009. p. 1-10.

MARK, Dave.; LAMARCHE, Jeff. **Beginning iPhone 3 development**: exploring the iPhone SDK. New York: Apress, 2009.

MENEZES, Célia M. C. A. V. Q. **Utilização de dispositivos móveis na escola do séc. XXI**: o impacto do podcast no processo ensino-aprendizagem da língua ingles no 7º ano do 3º ciclo do ensino básico. 2009. 119 f. Dissertação (Mestrado em Informática Educacional) - Universidade Portucalense Infante D. Henrique, Porto.

MEYERS, Scott.; LEE, Mike. **MAC OS X Leopard**: beyond the manual. New York: Apress, 2007.

MILUZZO, James M. H et al. Evaluating the iPhone as a mobile platform for people-centric sensing applications. In: INTERNATIONAL WORKSHOP ON URBAN, COMMUNITY, AND SOCIAL APPLICATIONS OF NETWORKED SENSING SYSTEMS - URBANSENSE08, 1., 2008, Raleigh. **Proceedings...** Hanover: Sensor Networks Group, 2008. p. 41-45.

PHELPS, Andrew M.; EGERT, Christopher A.; BAYLISS Jessica D. Media impact games in the classroom: using games as a motivator for studying computing: part 1. **IEEE MultiMedia**, [S.l.], v. 12, n. 2, p. 4-8, apr./june 2009.

SUN, Microsystems. **Memory management in the Java HotSpot™ Virtual Machine**. [S.l.], 2006. Disponível em: <http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf>. Acesso em: 10 nov. 2009.

VAHLDICK, Adilson; MATTOS, Mauro M. Aprendendo programação de computadores com experiências lúdicas. Artigo aceito mas não publicado. In: INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION, 6., 2009, Buenos Aires. **Anais…** Buenos Aires: ICECE, 2009. p. 1-6.

ZDZIARSKI, Jonathan. **iPhone SDK application development**: building applications for the AppStore. Sebastopol: O'Reilly Media, 2009.

APÊNDICE A – Principais classes do Furbot

O Quadro 36 demonstra os métodos da classe `ObjetoDoMundoAdapter` na linguagem Java.

ObjetoDoMundoAdapter	
MÉTODO	**DESCRIÇÃO**
public void adicionarObjetoNoMundo(ObjetoDoMundo objeto, Direcao direcao)	Chama doAdicionarObj com parâmetro embaixo = false
public void adicionarObjetoNoMundoEmbaixo(ObjetoDoMundo objeto, Direcao direcao)	Chama doAdicionarObj com parâmetro embaixo = true
public void adicionarObjetoNoMundoXY(ObjetoDoMundo objeto, int x, int y)	Chama doAdicionarObjXY com parâmetro embaixo = false
public void andarAbaixo()	Faz com que o objeto ande para baixo no mundo
public void andarAcima()	Faz com que o objeto ande para cima no mundo
public void andarDireita()	Faz com que o objeto ande para a direita no mundo
public void andarEsquerda()	Faz com que o objeto ande para a esquerda no mundo
public void bloquear()	Bloqueia o objeto
public void chegouUmObjetoNaPosicao(ObjetoDoMundo objetoChegou)	Informa que um objeto chegou e deve terminar espera
public void desbloquear()	Desbloqueia o objeto
public void diga(String texto)	Escreve na console a string passada
public void diga(Object object)	Escreve na console o método toString do objeto passado
private void doAdicionarObj(ObjetoDoMundo objeto, Direcao direcao, boolean embaixo)	Chama doAdicionarObjXY atualizando X e Y de acordo com a Direcao
private void doAdicionarObjXY(ObjetoDoMundo objeto, int x, int y, boolean embaixo)	Insere o objeto no mundo na posição passada como parâmetro
public boolean ehBloqueado()	Verifica se está bloqueado
public boolean ehDependenteEnergia()	Verifica se depende de energia para se mover
public boolean ehFim(Direcao direcao)	Verifica se está em algum canto do mundo de acordo com a Direcao
public boolean ehObjetoDoMundoTipo(String classe, Direcao direcao)	Verifica se a classe passada como parâmetro é do mesmo tipo do objeto
public boolean ehVazio(Direcao direcao)	Verifica se a posição passada como parâmetro está vazia
public void esperar(int segundos)	Espera para prosseguir a quantidade de segundos passados
public ObjetoDoMundo esperarAlguem()	Espera até aparecer um objeto
public int getEnergia()	Retorna quantidade de energia do objeto
public int getMaxEnergia()	Retorna a quantidade máxima de energia
public <T extends ObjetoDoMundo> getObjeto(Direcao direcao)	Retorna o objeto de acordo com a Direção
public ObjetoMundoImpl getObjetoMundoImpl()	Retorna o objetoDoMundoImpl
public <T extends ObjetoDoMundo> getObjetoXY(int x, int y)	Retorna o objeto de acordo com a posição X e Y passada
public String getSouDoTipo()	Retorna a classe do objeto
public int getTempoEspera()	Retorna o tempo definido que deve esperar
public int getUltimaTeclaPress()	Retorna a última tecla pressionada
public int getVelocidade()	Retorna a velocidade do objeto
public int getX()	Retorna a posição X que se encontra o objeto no mundo
public int getY()	Retorna a posição Y que se encontra o objeto no mundo
public boolean jahExplodiu()	Verifica se o objeto explodiu

public void limparConsole()	Limpa a console
public void mudarPosicao(int x, int y)	Muda a posição do objeto para a posição passada
public ObjetoDoMundoAdapter()	Construtor da classe
public void removerMe()	Remove o objeto do mundo
public void removerObjetoDoMundo(ObjetoDoMundo objeto)	Remove o objeto passado do mundo
public void repintar()	Repinta os ListernerMundo associados ao objeto
public void setObjetoMundoImpl(ObjetoMundoImpl objetoMundoImpl)	Define o objetoDoMundoImpl do objeto
public void setTempoEspera(int milisegundos)	Define o tempo de espera
public void setVelocidade(int velocidade)	Define a velocidade
public int sortear()	Retorna um número randômico

Quadro 36 – Detalhamento da classe `ObjetoDoMundoAdapter`

O Quadro 37 demonstra os métodos da classe `MundoFurbot` na linguagem Java.

MundoFurbot	
MÉTODO	**DESCRIÇÃO**
public void addDisseramListener(DisseramListener disseramListener)	Adiciona o objeto passado a lista de aviso Disseram
public void addFinalizouExecucaoListener(FinalizouExecucaoListener finalizouExecucaoListener)	Adiciona o objeto passado a lista de aviso FinalizouExecucao
public void addObjeto(ObjetoDoMundo objetoDoMundo, int x, int y)	Adiciona o objeto no mundo na posição passada
public void addObjeto(ObjetoDoMundo objetoDoMundo)	Adiciona o objeto no mundo na posição que ele está no momento
private void configKeyListener()	Cria evento para aguardar teclas
private void constroiGrade(MapaModel mapaModel)	Cria objeto de Tabela e associa ao mapaModel
public void executar()	Executa mundo
public void executar(final ObjetoDoMundo objetoDoMundo)	Chama método executar do objeto passado
public JTextArea getConsole()	Retorna a console
public int getTempoEspera()	Retorna o tempo de espera global
public MundoFurbot(int qtasColunas, int qtasLinhas, TamanhoCelula tamanhoCelula)	Construtor da classe que recebe número de linhas e colunas do mundo
public MundoFurbot(Exercicio exercicio)	Construtor da classe que recebe um objeto de exercício
public void parar()	Para a execução do mundo
public void pressionadaTecla(int keyCode)	Avisa que foi pressionado tecla
public void reiniciar()	Reinicia o mundo
public void removeDisseramListener(DisseramListener disseramListener)	Remove objeto da lista de Disseram
public void removeFinalizouExecucaoListener(FinalizouExecucaoListener finalizouExecucaoListener)	Remove objeto da lista de FinalizouExecucao
public void removerObjeto(int x, int y)	Remove objeto do mundo que está na posição passada
public void setBotaoExecutar(JComponent botaoExecutar)	Define o objeto do botão Executar
public void setBotaoParar(JComponent botaoParar)	Define o objeto do botão Parar
public void setConsole(JTextArea console	Define o objeto de Console
public void setExercicio(Exercicio exercicio)	Define o exercício atual
public void setTempoEspera(int milisegundos)	Define o tempo de espera global

Quadro 37 – Detalhamento da classe `MundoFurbot`

O Quadro 38 demonstra os métodos da classe `MundoVisual` na linguagem Java.

MundoVisual	
MÉTODO	**DESCRIÇÃO**
private void executar(Exercicio exercicio)	Seta exercicio no mundoFurbot e reinicia o mundo
public static <T> T getAtributo(String nome)	Retorna o atributo do MundoVisual
private JPanel getControle(final Exercicio exercicio	Cria objetos de controle, Executar, Parar, Reiniciar
public static int getMundo()	Retorna um inteiro que indica o mundo atual
private void habilitarBotoesExecucao()	Habilita botões para depois executar
public static void iniciar(String nomeArquivoXML)	Chama o iniciar com a classe e o autor nulo
public static void iniciar(String nomeArquivoXML, String autor)	Chama o iniciar com a classe nula
public static void iniciar(String nomeArquivoXML, Class<?> furbotClass, final String autor)	Lê arquivo de parâmetros e cria um novo objeto de MundoVisual passando o arquivo xml
public static void iniciar(String nomeArquivoXML, Class<?> furbotClass)	Chama o iniciar com o autor nulo
public static void iniciarSequencia(Class<?> furbotClass, String... nomesArquivosXML)	Chama o iniciarSequencia com o autor nulo
public static void iniciarSequencia(final String autor, Class<?> furbotClass, String... nomesArquivosXML)	Lê arquivo de parâmetros e cria um novo objeto de MundoVisual passando vários arquivos xml
private void initComponents(Exercicio exercicio, String autor)	Cria os componentes visuais
public static void main(String args[])	Permite que a classe mundoVisual seja executada
private MundoVisual(Exercicio exercicio, String autor)	Construtor da classe que executa um exercício
public MundoVisual(String autor, Exercicio exercicio, Class<?> furbotClass, String nomesArquivosXML[])	Construtor da classe que executa vários exercícios
private void novaSequencia(Exercicio exercicio)	Atualiza o anunciado e zera os atributos do MundoVisual
public static void proxMundo(int indiceMundo)	Executa próximo mundo caso exista o mundo solicitado
public static <T> void setAtributo(String nome, T valor)	Define os atributos do MundoVisual
public static boolean temAtributo(String nome)	Verifica se o MundoVisual possui o atributo solicitado

Quadro 38 – Detalhamento da classe `MundoVisual`

O Quadro 39 demonstra os métodos da classe `Mundo` na linguagem Java.

Mundo	
MÉTODO	**DESCRIÇÃO**
public synchronized void addListener(ListenerMundo listener)	Adiciona objeto passado a Lista que é avisada quando ocorre alguma iteração no mundo
public synchronized void addListenerCelula(ObjetoMundoImpl euMesmo)	Adiciona objeto passado a Lista de objetos que estão em uma determinada posição
public synchronized void addObjetoMundoImpl(ObjetoMundoImpl objMundo)	Chama addObjetoMundoImpl com parâmetro embaixo = false
public synchronized void addObjetoMundoImpl(ObjetoMundoImpl objMundo, boolean embaixo)	Adiciona objetoMundoImpl no modelo de dados
public synchronized void andar(ObjetoMundoImpl objMundo, Direcao direcao)	Faz com que o objetoMundoImpl ande na direção passada
private void atribuirModelo(int y, int x, PosicaoMundo posicaoMundo)	Avisa listenerMundo que um objeto de PosicaoMundo está sendo adicionado e adiciona o objeto de PosicaoMundo ao modelo de dados
public synchronized void disse(ObjetoMundoImpl objetoMundo, String texto)	Avisa listenerMundo que o objeto passado mandou uma mensagem
private void doAndar(ObjetoMundoImpl objMundo, Direcao direcao, int novoX, int novoY)	Faz checagens de colisão e caso seja possível, movimenta objeto pelo mundo de acordo com a Direcao passada
public synchronized boolean ehFim(ObjetoMundoImpl objetoMundo, Direcao direcao)	Verifica se o objetoMundoImpl estará no fim caso ande na Direcao

public synchronized boolean ehVazio(ObjetoMundoImpl objetoMundo, Direcao direcao)	Verifica se existe o objetoMundoImpl não está na Direcao passada
public synchronized boolean ehVazio(ObjetoMundoImpl objetoMundoImpl, int x, int y)	Verifica se existe algum objeto no modelo nas posições passadas
public synchronized void fimExecucao()	Avisa listeners do mundo que a execução foi terminada
public Object getImagem(int lin, int col)	Retorna imagem do objeto que está localizado nas posições passadas
public synchronized ObjetoMundoImpl getObjeto(ObjetoMundoImpl objetoMundo, int x, int y)	Retorna o objeto passado caso encontre ele nas posições passadas
+(ObjetoMundoImpl) getObjeto(ObjetoMundoImpl, Direcao)	Retorna o objeto caso encontre ele indo na Direcao passada
public synchronized List<ObjetoMundoImpl> getObjetos()	Retorna lista de objetos que fazem parte do modelo de dados
public int getQtdadeCol()	Retorna quantidade de colunas do modelo
public int getQtdadeLin()	Retorna quantidade de linhas do modelo
public int getTamCell()	Retorna o tamanho da célula do modelo
public synchronized int getUltimaTeclaPress()	Retorna última tecla pressionada
public boolean isExplodir()	Retorna se deve explodir
public boolean isUsarLinhasNaGrade()	Retorna se deve usar linhas de grade no modelo
public synchronized void limparConsole()	Avisa listeners mundo para limpar a console
private boolean movimentar(ObjetoMundoImpl objMundo, int x, int y)	Tenta movimentar objeto nas posições passadas
public synchronized void mudarPosicao(ObjetoMundoImpl objMundo, int x, int y)	Chama doAndar passando os parâmetros passados
public Mundo(int qtdadeLin, int qtdadeCol)	Cria mundo com a quantidade de linhas e colunas passadas
public synchronized void parar()	Avisa objetos do modelo que devem parar a execução
public synchronized void pressionadaTecla(int keyCode)	Registra última tecla pressionada
public synchronized void removeListener(ListenerMundo listener)	Remove objeto passado da lista de Listeners do mundo
private void removerObjeto(ObjetoMundoImpl objMundo, int yAnt, int xAnt)	Remove objeto passado do modelo se encontrado nas posições x/y
public synchronized void removerObjeto(ObjetoMundoImpl objMundo)	Remove objeto do modelo
protected synchronized void repintar()	Repinta a tela
public synchronized void run()	Executa os objetos que fazem parte do modelo
public void setExplodir(boolean explodir)	Define se é pra explodir
public void setTamCell(String tamanhoCel)	Define o tamanho da célula
public void setUsarLinhasNaGrade(boolean usarLinhasNaGrade)	Define se deve usar linhas de grade no modelo de dados

Quadro 39 – Detalhamento da classe `Mundo`

O Quadro 40 demonstra os métodos da classe `ObjetoMundoImpl` na linguagem Java.

ObjetoMundoImpl	
MÉTODO	**DESCRIÇÃO**
public void adicionarObjetoNoMundo(ObjetoMundoImpl obj)	Chama doAdicionarObjetoNoMundo com o parâmetro embaixo = false
public final void andarAbaixo()	Chama doAndar passando a Direcao Abaixo
public final void andarAcima()	Chama doAndar passando a Direcao Acima
public final void andarDireita()	Chama doAndar passando a Direcao Direita
public final void andarEsquerda()	Chama doAndar passando a Direcao Esquerda
public final void diga(Object object)	Chama diga passando o método ToString do objeto passado
public final void diga(final String texto)	Envia mensagem para o mundo dizer o parâmetro passado
private void doAdicionarObjetoNoMundo(final ObjetoMundoImpl objetoMundoImpl, final boolean embaixo))	Chama método do objeto mundo para adicionar objeto no modelo

private void doAndar(final Direcao direcao)	Chama método andar(ObjetoMundoImpl, Direcao) do objeto mundo
public boolean ehFim(final Direcao direcao)	Chama método ehFim(ObjetoMundoImpl, Direcao) do objeto mundo
public boolean ehVazio(final Direcao direcao)	Chama método ehVazio(ObjetoMundoImpl, Direcao) do objeto mundo
public boolean ehVazio(final int x, final int y)	Chama método ehVazio(ObjetoMundoImpl, int, int) do objeto mundo
public void esperarAlguem()	Chama método addListenerCelula(ObjetoMundoImpl) do objeto mundo
private void gastarEnergia(int qtde)	Gasta a quantia de energia passada do objetoMundoImpl
public int getEnergia()	Retorna quantidade de energia do objetoMundoImpl
public final ImageIcon getImage()	Retorna imagem associada ao objetoMundoImpl
public int getMaxEnergia()	Retorna quantidade máxima de energia
public Mundo getMundo()	Retorna o mundo associado ao objetoMundoImpl
public ObjetoMundoImpl getObjeto(final Direcao direcao)	Chama método getObjeto(ObjetoMundoImpl, Direcao) do objeto mundo
public ObjetoMundoImpl getObjeto(final int x, final int y)	Chama método getObjeto(ObjetoMundoImpl, int, int) do objeto mundo
public <T extends ObjetoDoMundo> T getObjetoMundo()	Retorna o objeto mundo associado ao objetoMundoImpl
public <T extends ObjetoDoMundo> List<T> getObjetos(Class<T> clazz)	Retorna lista com objetos do mundo que possuem a classe passada
public List<ObjetoMundoImpl> getObjetos()	Retorna uma lista com o objetoMundoImpl
public int getQtdadeCol()	Chama método getQtdadeCol() do objeto mundo
public int getQtdadeLin()	Chama método getQtdadeLin() do objeto mundo
public int getTempoEspera()	Retorna tempo geral de espera
public int getUltimaTeclaPress()	Chama método getUltimaTeclaPress() do objeto mundo
public int getX()	Retorna posição X que o objeto se encontra
public int getY()	Retorna posição Y que o objeto se encontra
public void inserirObjetoNoMundo(ObjetoMundoImpl obj)	Chama método doAdicionarNoMundo com o parâmetro embaixo=true
public void inteligencia()	Método que deve ser sobrescrito pelo programador
public boolean isBloqueado()	Verifica se o objeto está bloqueado na posição dele
public boolean isExplodiu()	Verifica se o objeto explodiu
public boolean isParar()	Verifica se o objeto pode parar
public boolean isRemovido()	Verifica se objeto foi removido do modelo do mundo
public boolean isUsarEnergia()	Verifica se objeto usa energia
public void limparConsole()	Chama o método limparConsole() do objeto mundo
public void mudarPosicao(final int x, final int y)	Chama o método mudarPosicao(ObjetoMundoImpl, int, int) do objeto mundo
public ObjetoMundoImpl(ObjetoDoMundo objetoMundo)	Cria objetoMundoImpl e associa ao objetoDoMundo
public void parar()	Para a execução do objeto
public void pararMundo()	Para a execução do mundo
public void removerMe()	Remove-se do mundo
public void repintar()	Repinta o objeto no mundo
public void run()	Chama o método inteligencia()
public void setBloqueado(boolean bloqueado)	Define se o objeto está bloqueado ou não na célula
public void setEnergia(int energia)	Define quantidade de energia do objeto
public void setExplodiu(boolean explodiu)	Define se o objeto explodiu
public final void setImage(ImageIcon imagem)	Define a imagem do objeto
public void setMaxEnergia(int maxEnergia)	Define o máximo de energia do objeto

public void setMundo(Mundo mundo)	Associa o mundo ao objeto
public void setRemovido(boolean removido)	Define se o objeto foi removido do mundo
public void setTempoEspera(int milisegundos)	Define o tempo de espera padrão do objeto
public void setUsarEnergia(boolean usarEnergia)	Define se objeto usa energia
public void setX(int x)	Define a posição X onde o objeto se encontra
public void setY(int y)	Define a posição Y onde o objeto se encontra

Quadro 40 – Detalhamento da classe `ObjetoMundoImpl`

O Quadro 41 demonstra os métodos da classe `Exercicio` na linguagem Java.

Exercicio	
MÉTODO	**DESCRIÇÃO**
private void addElementsOfGroups(List<ElementoExercicio> elemsAUsar, List<GrupoObjetos> grupos)	Carrega todos os objetos participantes de grupos que devem ser incluidos no mundo
public void addRobo(ElementoExercicio robo)	Adiciona robô a lista de objetos a serem adicionados ao mundo
private void adicionarObjetosMundo(boolean random, boolean dependentes, Mundo m, List<String> posUsadas, List<ElementoExercicio> elems)	Cria objetoMundoImpl e depois adiciona ao mundo
private void calcularQtdadeLinCol()	Calcula quantidade de linhas e colunas de acordo com os parâmetros do XML de configuração
public Mundo criarMundo()	Cria objeto do mundo e carrega objetos nele
private ObjetoMundoImpl criarObjetoMundoImpl(Mundo mundo, List<String> posUsadas, ElementoExercicio elemento, String clazz)	Cria objetoMundoImpl
public Exercicio(String nomeArquivoXML)	Construtor da classe
private ArrayList<ElementoExercicio> extrairElementosPosicao(boolean random, boolean dependentes, List<ElementoExercicio> elemsOrigem)	Carrega elementos com as características dos parâmetros passados e retorna em uma lista
public void finalizar()	Bloqueia robôs do mundo
public String getEnunciado()	Retorna enunciado definido no XML de configuração
private ObjetoMundoImpl getObjetoMundoImpl(Mundo mundo, String id)	Retorna objeto do mundo que possui o id passado como parâmetro
public int getQtdadeCol()	Retorna quantidade de colunas que deve ter o mundo
public int getQtdadeLin()	Retorna quantidade de linhas que deve ter o mundo
public FurbotRandom getRandom()	Retorna objeto de random
public String getTamanhoCel()	Retorna o tamanho da célula que deve ter o modelo de dados
private int getX(Mundo mundo, ElementoExercicio elemento)	Retorna posição X do objeto ElementoExercicio
private int getY(Mundo mundo, ElementoExercicio elemento)	Retorna posição Y do objeto ElementoExercicio
public boolean isExplodir()	Retorna se foi definido no XML para o mundo explodir
public boolean isUsarLinhasNaGrade()	Retorna se foi definido no XML para usar linhas de grade
private void refazerPosicaoDependente(java.awt.Point p, Mundo mundo, ElementoExercicio elemento)	Calcula posição X e Y do objeto do mundo caso no elementoExercicio esteja definido que o objeto é dependente
public void setClassRobo(String classRobo)	Define a classe dos robos
public void setEnunciado(String enunciado)	Define o enunciado
public void setExplodir(boolean explodir)	Define se o mundo pode explodir
public void setQtdadeCol(int qtdadeCol)	Define a quantidade de colunas
public void setQtdadeLin(int qtdadeLin)	Define a quantidade de linhas
public void setRandom(FurbotRandom random)	Define o objeto de random
public void setTamanhoCel(String tamanhoCel)	Define o tamanho da célula
public void setUsarLinhasNaGrade(boolean usarLinhasNaGrade)	Define se deve usar linhas de grade no modelo

Quadro 41 – Detalhamento da classe `Exercicio`

APÊNDICE B – Principais classes do iFurbot

O Quadro 42 demonstra os métodos da classe `ObjetoDoMundoAdapter` comparando a sua versão em Java com Objective-C.

ObjetoDoMundoAdapter		
ID	**MÉTODO**	**DESCRIÇÃO**
1	public void adicionarObjetoNoMundo(ObjetoDoMundo objeto, Direcao direcao)	Chama doAdicionarObj com parâmetro embaixo = false
	-(void) adicionarObjetoNoMundo:(id <ObjetoDoMundo>) objeto pDirecao:(Direcao) direcao	
2	public void adicionarObjetoNoMundoEmbaixo(ObjetoDoMundo objeto, Direcao direcao)	Chama doAdicionarObj com parâmetro embaixo = true
	-(void) adicionarObjetoNoMundoEmbaixo:(id <ObjetoDoMundo>) objeto pDirecao:(Direcao) direcao	
3	public void adicionarObjetoNoMundoXY(ObjetoDoMundo objeto, int x, int y)	Chama doAdicionarObjXY com parâmetro embaixo = false
	-(void) adicionarObjetoNoMundoXY:(id <ObjetoDoMundo>) objeto pi:(NSInteger) i pj:(NSInteger) ij	
4	public void andarAbaixo()	Faz com que o objeto ande para baixo no mundo
	-(void) andarAbaixo	
5	public void andarAcima()	Faz com que o objeto ande para cima no mundo
	-(void) andarAcima	
6	public void andarDireita()	Faz com que o objeto ande para a direita no mundo
	-(void) andarDireita	
7	public void andarEsquerda()	Faz com que o objeto ande para a esquerda no mundo
	-(void) andarEsquerda	
8	public void bloquear()	Bloqueia o objeto
	-(void) bloquear	
9	public void chegouUmObjetoNaPosicao(ObjetoDoMundo objetoChegou)	Informa que um objeto chegou e deve terminar espera
	-(void) chegouUmObjetoNaPosicao:(id <ObjetoDoMundo>) objetoChegou	
10	public void desbloquear()	Desbloqueia o objeto
	-(void) desbloquear	
11	public void diga(String texto)	Escreve na console a string passada
	-(void) diga:(NSObject *) objeto	
12	public void diga(Object object)	Escreve na console o método toString do objeto passado
	-(void) diga:(NSObject *) objeto	
13	private void doAdicionarObj(ObjetoDoMundo objeto, Direcao direcao, boolean embaixo)	Chama doAdicionarObjXY atualizando X e Y de acordo com a Direcao
	-(void) doAdicionarObj:(id <ObjetoDoMundo>) objeto pDirecao:(Direcao) direcao pEmbaixo:(BOOL) embaixo	
14	private void doAdicionarObjXY(ObjetoDoMundo objeto, int x, int y, boolean embaixo)	Insere o objeto no mundo na posição passada como parâmetro
	-(void) doAdicionarObjXY:(id <ObjetoDoMundo>) objeto pX:(NSInteger) ix pY:(NSInteger) iy pEmbaixo:(BOOL) embaixo	

15	public boolean ehBloqueado()	Verifica se está bloqueado
	-(BOOL) ehBloqueado	
16	public boolean ehDependenteEnergia()	Verifica se depende de energia para se mover
	-(BOOL) ehDependenteEnergia	
17	public boolean ehFim(Direcao direcao)	Verifica se está em algum canto do mundo de acordo com a Direcao
	-(BOOL) ehFim:(Direcao) direcao	
18	public boolean ehObjetoDoMundoTipo(String classe, Direcao direcao)	Verifica se a classe passada como parâmetro é do mesmo tipo do objeto
	-(BOOL) ehObjetoDoMundoTipo:(NSString *) classe pDirecao:(Direcao) direcao	
19	public boolean ehVazio(Direcao direcao)	Verifica se a posição passada como parâmetro está vazia
	-(BOOL) ehVazio:(Direcao) direcao	
20	public void esperar(int segundos)	Espera para prosseguir a quantidade de segundos passados
	-(void) esperar:(NSInteger) segundos	
21	public ObjetoDoMundo esperarAlguem()	Espera até aparecer um objeto
	-(id <ObjetoDoMundo>) esperarAlguem	
22	public int getEnergia()	Retorna quantidade de energia do objeto
	-(NSInteger) energia	
23	public int getMaxEnergia()	Retorna a quantidade máxima de energia
	-(NSInteger) maxEnergia	
24	public <T extends ObjetoDoMundo> getObjeto(Direcao direcao)	Retorna o objeto de acordo com a Direção
	-(id <ObjetoDoMundo>) objeto:(Direcao) direcao	
25	public ObjetoMundoImpl getObjetoMundoImpl()	Retorna o objetoDoMundoImpl
	@property (nonatomic, retain) ObjetoMundoImpl *objetoMundoImpl;	
26	public <T extends ObjetoDoMundo> getObjetoXY(int x, int y)	Retorna o objeto de acordo com a posição X e Y passada
	-(id <ObjetoDoMundo>) objetoXY:(NSInteger) i pj:(NSInteger) j	
27	public String getSouDoTipo()	Retorna a classe do objeto
	-(NSString *) souDoTipo	
28	public int getTempoEspera()	Retorna o tempo definido que deve esperar
	-(double) tempoEspera	
29	public int getUltimaTeclaPress()	Retorna a última tecla pressionada
	-(NSInteger) ultimaTeclaPress	
30	public int getVelocidade()	Retorna a velocidade do objeto
	-(double) velocidade	
31	public int getX()	Retorna a posição X que se encontra o objeto no mundo
	-(NSInteger) x	
32	public int getY()	Retorna a posição Y que se encontra o objeto no mundo
	-(NSInteger) y	
33	public boolean jahExplodiu()	Verifica se o objeto explodiu
	-(BOOL) jahExplodiu	
34	public void limparConsole()	Limpa a console
	-(void) limparConsole	
35	public void mudarPosicao(int x, int y)	Muda a posição do objeto para a posição passada
	-(void) mudarPosicao:(NSInteger) x pY:(NSInteger) y	
36	public ObjetoDoMundoAdapter()	Construtor da classe
	-(id) init	
37	public void removerMe()	Remove o objeto do mundo
	-(void) removerMe	

38	public void removerObjetoDoMundo(ObjetoDoMundo objeto)	Remove o objeto passado do mundo
	-(void) removerObjetoDoMundo:(id <ObjetoDoMundo>) objeto	
39	public void repintar()	Repinta o próprio objeto
	-(void) repintar	
40	public void setObjetoMundoImpl(ObjetoMundoImpl objetoMundoImpl)	Define o objetoDoMundoImpl do objeto
	@property (nonatomic, retain) ObjetoMundoImpl *objetoMundoImpl;	
41	public void setTempoEspera(int milisegundos)	Define o tempo de espera
	-(void) setTempoEspera:(double) segundos	
42	public void setVelocidade(int velocidade)	Define a velocidade
	-(void) setVelocidade:(double) velocidade	
43	public int sortear()	Retorna um número randômico
	-(NSInteger) sortear	

Quadro 42 – Detalhamento da classe `ObjetoDoMundoAdapter` em Objective-C

O Quadro 43 demonstra os métodos da classe `MundoFurbot` comparando a sua versão em Java com Objective-C.

MundoFurbot		
ID	**MÉTODO**	**DESCRIÇÃO**
1	public void addDisseramListener(DisseramListener disseramListener)	Adiciona o objeto passado a lista de aviso Disseram
	-(void) addDisseramListener:(id <DisseramListener>) disseramListener	
2	public void addFinalizouExecucaoListener(FinalizouExecucaoListener finalizouExecucaoListener)	Adiciona o objeto passado a lista de aviso FinalizouExecucao
	-(void) addFinalizouExecucaoListener:(id <FinalizouExecucaoListener>) finalizouExecucaoListener	
3	public void addObjeto(ObjetoDoMundo objetoDoMundo, int x, int y)	Adiciona o objeto no mundo na posição passada
	-(void) addObjeto:(id <ObjetoDoMundo>) objetoDoMundo px:(NSInteger) x py:(NSInteger) y	
4	public void addObjeto(ObjetoDoMundo objetoDoMundo)	Adiciona o objeto no mundo na posição que ele está no momento
	-(void) addObjeto:(id <ObjetoDoMundo>) objetoDoMundo	
5	private void configKeyListener()	Cria evento para aguardar teclas
	-(void) configKeyListener	
6	private void constroiGrade(MapaModel mapaModel)	Cria objeto de Tabela e associa ao mapaModel
	-(void) constroiGrade	
7	public void executar()	Executa mundo
	-(void) executar	
8	public void executar(final ObjetoDoMundo objetoDoMundo)	Chama método executar do objeto passado
	-(void) executar:(id <ObjetoDoMundo>) objetoDoMundo	
9	public JTextArea getConsole()	Retorna a console
	@property (nonatomic, retain) Console *console	
10	public int getTempoEspera()	Retorna o tempo de espera global
	-(double) tempoEspera	
11	public MundoFurbot(int qtasColunas, int qtasLinhas, TamanhoCelula tamanhoCelula)	Construtor da classe que recebe número de linhas e colunas do mundo
	-(id) initWithLinAndCol:(NSInteger) qtasColunas pQtasLinhas:(NSInteger) qtasLinhas pTamanhoCelula:(NSString *) tamanhoCelula	
12	public MundoFurbot(Exercicio exercicio)	Construtor da classe que recebe um objeto de exercício

	-(id) initWithExercicio:(Exercicio *) exer	
13	public void parar()	Para a execução do mundo
	-(void) parar	
14	public void pressionadaTecla(int keyCode)	Avisa que foi pressionado tecla
	-(void) pressionadaTecla:(NSInteger) keyCode	
15	public void reiniciar()	Reinicia o mundo
	-(void) reiniciar	
16	public void removeDisseramListener(DisseramListener disseramListener)	Remove objeto da lista de Disseram
	-(void) removeDisseramListener:(id <DisseramListener>) disseramListener	
17	public void removeFinalizouExecucaoListener(FinalizouExecucaoListener finalizouExecucaoListener)	Remove objeto da lista de FinalizouExecucao
	-(void) removeFinalizouExecucaoListener:(id <FinalizouExecucaoListener>) finalizouExecucaoListener	
18	public void removerObjeto(int x, int y)	Remove objeto do mundo que está na posição passada
	-(void) removerObjeto:(NSInteger) x pY:(NSInteger) y	
19	public void setBotaoExecutar(JComponent botaoExecutar)	Define o objeto do botão Executar
	-(void) setBotaoExecutar:(MenuItemImage *) botaoE	
20	public void setBotaoParar(JComponent botaoParar)	Define o objeto do botão Parar
	-(void) setBotaoParar:(MenuItemImage *) botaoP	
21	public void setConsole(JTextArea console	Define o objeto de Console
	@property (nonatomic, retain) Console *console	
22	public void setExercicio(Exercicio exercicio)	Define o exercício atual
	-(void) setExercicio:(Exercicio *) exer	
23	public void setTempoEspera(int milisegundos)	Define o tempo de espera global
	-(void) setTempoEspera:(double) segundos	

Quadro 43 – Detalhamento da classe `MundoFurbot` em Objective-C

O Quadro 44 demonstra os métodos da classe `MundoVisual` comparando a sua versão em Java com Objective-C.

	MundoVisual	
ID	**MÉTODO**	**DESCRIÇÃO**
1	private void executar(Exercicio exercicio)	Seta exercicio no mundoFurbot e reinicia o mundo
2	public static <T> T getAtributo(String nome)	Retorna o atributo do MundoVisual
	+(id) atributo:(NSString *) nome	
3	private JPanel getControle(final Exercicio exercicio	Cria objetos de controle, Executar, Parar, Reiniciar
	-(void) initTelaEnunciado / -(void) initTelaPrincipal / -(void) initTelaConsole	
4	public static int getMundo()	Retorna um inteiro que indica o mundo atual
	+(NSInteger) mundo	
5	private void habilitarBotoesExecucao()	Habilita botões para depois executar
	-(void) habilitarBotoesExecucao	
6	public static void iniciar(String nomeArquivoXML)	Chama o iniciar com a classe e o autor nulo
	+(void) iniciar:(NSString *) nomeArquivoXML	
7	public static void iniciar(String nomeArquivoXML, String autor)	Chama o iniciar com a classe nula
	+(void) iniciar:(NSString *) nomeArquivoXML pAutor:(NSString *) autor	
8	public static void iniciar(String nomeArquivoXML, Class<?>	Lê arquivo de parâmetros e cria um novo objeto de

	furbotClass, final String autor)	MundoVisual
	+(void) iniciar:(NSString *) nomeArquivoXML pFurbotClass:(Class) cfurbotClass pAutor:(NSString *) autor	passando o arquivo xml
9	public static void iniciar(String nomeArquivoXML, Class<?> furbotClass)	Chama o iniciar com o autor nulo
	+(void) iniciar:(NSString *) nomeArquivoXML pFurbotClass:(Class) cfurbotClass	
10	public static void iniciarSequencia(Class<?> furbotClass, String... nomesArquivosXML)	Chama o iniciarSequencia com o autor nulo
	+(void) iniciarSequencia:(Class) cfurbotClass pNomesArquivosXML:(NSString *) anomesArquivosXML, ... NS_REQUIRES_NIL_TERMINATION	
11	public static void iniciarSequencia(final String autor, Class<?> furbotClass, String... nomesArquivosXML)	Lê arquivo de parâmetros e cria um novo objeto de MundoVisual passando vários arquivos xml
	+(void) iniciarSequencia:(NSString *) autor pFurbotClass:(Class) furbotClass pNomesArquivosXML:anomesArquivosXML, ... NS_REQUIRES_NIL_TERMINATION	
12	private void initComponents(Exercicio exercicio, String autor)	Cria os componentes visuais
	-(void) initComponents:(Exercicio *) exercicio pAutor:(NSString *) autor	
13	public static void main(String args[])	Permite que a classe mundoVisual seja executada
14	private MundoVisual(Exercicio exercicio, String autor)	Construtor da classe que executa um exercício
	-(id) initWithExercicio:(Exercicio *) exercicio pAutor:(NSString *) autor	
15	public MundoVisual(String autor, Exercicio exercicio, Class<?> furbotClass, String nomesArquivosXML[])	Construtor da classe que executa vários exercícios
	-(id) initWithParams:(NSString *) autor pExercicio:(Exercicio *) exercicio pFurbotClass:(Class) cfurbotClass pNomesArquivosXML:(NSMutableArray *) anomesArquivosXML	
16	private void novaSequencia(Exercicio exercicio)	Atualiza o anunciado e zera os atributos do MundoVisual
	-(void) novaSequencia:(Exercicio *) exercicio	
17	public static void proxMundo(int indiceMundo)	Executa próximo mundo caso exista o mundo solicitado
	+(void) proxMundo:(NSInteger) indiceMundo	
18	public static <T> void setAtributo(String nome, T valor)	Define os atributos do MundoVisual
	+(void) setAtributo:(NSString *) nome pValor:(id) valor	
19	public static boolean temAtributo(String nome)	Verifica se o MundoVisual possui o atributo solicitado
	+(BOOL) temAtributo:(NSString *) nome	

Quadro 44 – Detalhamento da classe `MundoVisual` em Objective-C

O Quadro 45 demonstra os métodos da classe `Mundo` comparando a sua versão em Java com Objective-C.

Mundo		
ID	**MÉTODO**	**DESCRIÇÃO**
1	public synchronized void addListener(ListenerMundo listener)	Adiciona objeto passado a Lista que é avisada quando ocorre alguma iteração no mundo
	-(void) addListener:(id <ListenerMundo>) listener	
2	public synchronized void addListenerCelula(ObjetoMundoImpl euMesmo)	Adiciona objeto passado a Lista de objetos que estão em uma determinada posição

	-(void) addListenerCelula:(ObjetoMundoImpl *) euMesmo	
3	public synchronized void addObjetoMundoImpl(ObjetoMundoImpl objMundo)	Chama addObjetoMundoImpl com parâmetro embaixo = false
	-(void) addObjetoMundoImpl:(ObjetoMundoImpl *) objMundo	
4	public synchronized void addObjetoMundoImpl(ObjetoMundoImpl objMundo, boolean embaixo)	Adiciona objetoMundoImpl no modelo de dados
	-(void) addObjetoMundoImpl:(ObjetoMundoImpl *) objMundo pembaixo:(BOOL) embaixo	
5	public synchronized void andar(ObjetoMundoImpl objMundo, Direcao direcao)	Faz com que o objetoMundoImpl ande na direção passada
	-(void) andar:(ObjetoMundoImpl *) objMundo pDirecao:(Direcao) direcao	
6	private void atribuirModelo(int y, int x, PosicaoMundo posicaoMundo)	Avisa listenerMundo que um objeto de PosicaoMundo está sendo adicionado e adiciona o objeto de PosicaoMundo ao modelo de dados
	-(void) atribuirModelo: (NSInteger) y px:(NSInteger) x posicao:(id <PosicaoMundo>) posicaoMundo	
7	public synchronized void disse(ObjetoMundoImpl objetoMundo, String texto)	Avisa listenerMundo que o objeto passado mandou uma mensagem
	-(void) disse:(ObjetoMundoImpl *) objetoMundo ptexto:(NSString *) texto	
8	private void doAndar(ObjetoMundoImpl objMundo, Direcao direcao, int novoX, int novoY)	Faz checagens de colisão e caso seja possível, movimenta objeto pelo mundo de acordo com a Direcao passada
	-(void) doAndar:(ObjetoMundoImpl *) objMundo pDirecao:(Direcao) direcao pNovoX:(NSInteger) novoX pNovoY:(NSInteger) novoY	
9	public synchronized boolean ehFim(ObjetoMundoImpl objetoMundo, Direcao direcao)	Verifica se o objetoMundoImpl estará no fim caso ande na Direcao
	-(BOOL) ehFim:(ObjetoMundoImpl *) objetoMundo pDirecao:(Direcao) direcao	
10	public synchronized boolean ehVazio(ObjetoMundoImpl objetoMundo, Direcao direcao)	Verifica se existe o objetoMundoImpl não está na Direcao passada
	-(BOOL) ehVazio:(ObjetoMundoImpl *) objetoMundo pDirecao:(Direcao) direcao	
11	public synchronized boolean ehVazio(ObjetoMundoImpl objetoMundoImpl, int x, int y)	Verifica se existe algum objeto no modelo nas posições passadas
	-(BOOL) ehVazio:(ObjetoMundoImpl *) objetoMundoImpl px:(NSInteger) x py:(NSInteger)y	
12	public synchronized void fimExecucao()	Avisa listeners do mundo que a execução foi terminada
	-(void) fimExecucao	
13	public Object getImagem(int lin, int col)	Retorna imagem do objeto que está localizado nas posições passadas
	-(NSObject *) imagem:(NSInteger) lin pcol:(NSInteger) col	
14	public synchronized ObjetoMundoImpl getObjeto(ObjetoMundoImpl objetoMundo, int x, int y)	Retorna o objeto passado caso encontre ele nas posições passadas
	-(ObjetoMundoImpl *) objeto:(ObjetoMundoImpl *) objetoMundo px:(NSInteger) x py:(NSInteger) y	
15	+(ObjetoMundoImpl) getObjeto(ObjetoMundoImpl, Direcao)	Retorna o objeto caso encontre ele indo na Direcao passada
	-(ObjetoMundoImpl *) objeto:(ObjetoMundoImpl *) objetoMundo pDirecao: (Direcao) direcao	
16	public synchronized List<ObjetoMundoImpl> getObjetos()	Retorna lista de objetos que fazem parte do modelo de dados
	-(NSMutableArray *) objetos	
17	public int getQtdadeCol()	Retorna quantidade de colunas do modelo

	@property (nonatomic, readonly) NSInteger qtdadeCol	
18	public int getQtdadeLin()	Retorna quantidade de linhas do modelo
	@property (nonatomic, readonly) NSInteger qtdadeLin	
19	public int getTamCell()	Retorna o tamanho da célula do modelo
	-(NSNumber *) tamCell	
20	public synchronized int getUltimaTeclaPress()	Retorna última tecla pressionada
	-(NSInteger) ultimaTeclaPress	
21	public boolean isExplodir()	Retorna se deve explodir
	@property (nonatomic, assign, getter=isExplodir) BOOL explodir	
22	public boolean isUsarLinhasNaGrade()	Retorna se deve usar linhas de grade no modelo
	@property (nonatomic, assign, getter=isUsarLinhasNaGrade) BOOL usarLinhasNaGrade	
23	public synchronized void limparConsole()	Avisa listeners mundo para limpar a console
	-(void) limparConsole	
24	private boolean movimentar(ObjetoMundoImpl objMundo, int x, int y)	Tenta movimentar objeto nas posições passadas
	-(BOOL) movimentar:(ObjetoMundoImpl *) objMundo px:(NSInteger) x py:(NSInteger) y	
25	public synchronized void mudarPosicao(ObjetoMundoImpl objMundo, int x, int y)	Chama doAndar passando os parâmetros passados
	-(void) mudarPosicao:(ObjetoMundoImpl *) objMundo px:(NSInteger)x py:(NSInteger)y	
26	public Mundo(int qtdadeLin, int qtdadeCol)	Cria mundo com a quantidade de linhas e colunas passadas
	-(id) initWithLinCol:(NSInteger) qtdadeLin pqtdadeCol:(NSInteger) qtdadeCol	
27	public synchronized void parar()	Avisa objetos do modelo que devem parar a execução
	-(void) parar	
28	public synchronized void pressionadaTecla(int keyCode)	Registra última tecla pressionada
	-(void) pressionadaTecla:(NSInteger) keyCode	
29	public synchronized void removeListener(ListenerMundo listener)	Remove objeto passado da lista de Listeners do mundo
	-(void) removeListener:(id <ListenerMundo>) listener	
30	private void removerObjeto(ObjetoMundoImpl objMundo, int yAnt, int xAnt)	Remove objeto passado do modelo se encontrado nas posições x/y
	-(void) removerObjeto:(ObjetoMundoImpl *) objMundo pyAnt:(NSInteger) yAnt pxAnt:(NSInteger) xAnt	
31	public synchronized void removerObjeto(ObjetoMundoImpl objMundo)	Remove objeto do modelo
	-(void) removerObjeto:(ObjetoMundoImpl *) objMundo	
32	protected synchronized void repintar()	Repinta a tela
	-(void) repintar	
33	public synchronized void run()	Executa os objetos que fazem parte do modelo
	-(void) run	
34	public void setExplodir(boolean explodir)	Define se é pra explodir
	@property (nonatomic, assign, getter=isExplodir) BOOL explodir	
35	public void setTamCell(String tamanhoCel)	Define o tamanho da célula
	-(void) setTamCell:(NSString *) tamanhoCel	
36	public void setUsarLinhasNaGrade(boolean usarLinhasNaGrade)	Define se deve usar linhas de grade no modelo de dados
	@property (nonatomic, assign, getter=isUsarLinhasNaGrade) BOOL usarLinhasNaGrade	

Quadro 45 – Detalhamento da classe `Mundo` em Objective-C

O Quadro 46 demonstra os métodos da classe `ObjetoMundoImpl` comparando a sua versão em Java com Objective-C.

ObjetoMundoImpl		
ID	**MÉTODO**	**DESCRIÇÃO**
1	public void adicionarObjetoNoMundo(ObjetoMundoImpl obj)	Chama doAdicionarObjetoNoMundo com o parâmetro embaixo = false
	-(void) adicionarObjetoNoMundo:(ObjetoMundoImpl *) obj	
2	public final void andarAbaixo()	Chama doAndar passando a Direcao Abaixo
	-(void) andarAbaixo	
3	public final void andarAcima()	Chama doAndar passando a Direcao Acima
	-(void) andarAcima	
4	public final void andarDireita()	Chama doAndar passando a Direcao Direita
	-(void) andarDireita	
5	public final void andarEsquerda()	Chama doAndar passando a Direcao Esquerda
	-(void) andarEsquerda	
6	public final void diga(Object object)	Chama diga passando o método ToString do objeto passado
	-(void) diga:(NSObject *) object	
7	public final void diga(final String texto)	Envia mensagem para o mundo dizer o parâmetro passado
	-(void) diga:(NSObject *) object	
8	private void doAdicionarObjetoNoMundo(final ObjetoMundoImpl objetoMundoImpl, final boolean embaixo))	Chama método do objeto mundo para adicionar objeto no modelo
	-(void) doAdicionarObjetoNoMundo:(ObjetoMundoImpl *) objetoMundoImpl pEmbaixo:(BOOL) embaixo	
9	private void doAndar(final Direcao direcao)	Chama método andar(ObjetoMundoImpl, Direcao) do objeto mundo
	-(void) doAndar:(Direcao) direcao	
10	public boolean ehFim(final Direcao direcao)	Chama método ehFim(ObjetoMundoImpl, Direcao) do objeto mundo
	-(BOOL) ehFim:(Direcao) direcao	
11	public boolean ehVazio(final Direcao direcao)	Chama método ehVazio(ObjetoMundoImpl, Direcao) do objeto mundo
	-(BOOL) ehVazio:(Direcao) direcao	
12	public boolean ehVazio(final int x, final int y)	Chama método ehVazio(ObjetoMundoImpl, int, int) do objeto mundo
	-(BOOL) ehVazio:(NSInteger) ix py:(NSInteger) iy	
13	public void esperarAlguem()	Chama método addListenerCelula(ObjetoMundoImpl) do objeto mundo
	-(void) esperarAlguem	
14	private void gastarEnergia(int qtde)	Gasta a quantia de energia passada do objetoMundoImpl
	-(void) gastarEnergia:(NSInteger) qtde	
15	public int getEnergia()	Retorna quantidade de energia do objetoMundoImpl
	@property (nonatomic, assign) NSInteger energia	
16	public final ImageIcon getImage()	Retorna imagem associada ao objetoMundoImpl
	-(CocosNode *) image	
17	public int getMaxEnergia()	Retorna quantidade máxima de energia
	@property (nonatomic, assign) NSInteger maxEnergia	
18	public Mundo getMundo()	Retorna o mundo associado ao objetoMundoImpl
	@property (nonatomic, retain) Mundo *mundo	
19	public ObjetoMundoImpl getObjeto(final Direcao direcao)	Chama método getObjeto(ObjetoMundoImpl, Direcao) do objeto mundo
	-(ObjetoMundoImpl *) objeto:(Direcao) direcao	
20	public ObjetoMundoImpl getObjeto(final int x, final int y)	Chama método getObjeto(ObjetoMundoImpl, int, int) do objeto mundo
	-(ObjetoMundoImpl *) objeto:(NSInteger) ix py:(NSInteger) iy	
21	public <T extends ObjetoDoMundo> T getObjetoMundo()	Retorna o objeto mundo associado ao objetoMundoImpl
	@property (nonatomic, retain) id objetoMundo	

22	public <T extends ObjetoDoMundo> List<T> getObjetos(Class<T> clazz)	Retorna lista com objetos do mundo que possuem a classe passada
	@property (nonatomic, retain) id objetoMundo	
23	public List<ObjetoMundoImpl> getObjetos()	Retorna uma lista com o objetoMundoImpl
	@property (nonatomic, readonly, getter=objetos) NSMutableArray *umaListaComigo	
24	public int getQtdadeCol()	Chama método getQtdadeCol() do objeto mundo
	-(NSInteger) qtdadeCol	
25	public int getQtdadeLin()	Chama método getQtdadeLin() do objeto mundo
	-(NSInteger) qtdadeLin	
26	public int getTempoEspera()	Retorna tempo geral de espera
	-(double) tempoEspera	
27	public int getUltimaTeclaPress()	Chama método getUltimaTeclaPress() do objeto mundo
	-(NSInteger) ultimaTeclaPress	
28	public int getX()	Retorna posição X que o objeto se encontra
	@property (nonatomic, assign) NSInteger x	
29	public int getY()	Retorna posição Y que o objeto se encontra
	@property (nonatomic, assign) NSInteger y	
30	public void inserirObjetoNoMundo(ObjetoMundoImpl obj)	Chama método doAdicionarNoMundo com o parâmetro embaixo=true
	-(void) inserirObjetoNoMundo:(ObjetoMundoImpl *) obj	
31	public void inteligencia()	Método que deve ser sobrescrito pelo programador
	-(void) inteligencia	
32	public boolean isBloqueado()	Verifica se o objeto está bloqueado na posição dele
	@property (nonatomic, assign, getter=isBloqueado) BOOL bloqueado	
33	public boolean isExplodiu()	Verifica se o objeto explodiu
	@property (nonatomic, assign, getter=isExplodiu, setter=setExplodir) BOOL explodiu	
34	public boolean isParar()	Verifica se o objeto pode parar
	-(BOOL) isParar	
35	public boolean isRemovido()	Verifica se objeto foi removido do modelo do mundo
	@property (nonatomic, assign, getter=isRemovido) BOOL removido	
36	public boolean isUsarEnergia()	Verifica se objeto usa energia
	@property (nonatomic, assign, getter=isUsarEnergia) BOOL usarEnergia	
37	public void limparConsole()	Chama o método limparConsole() do objeto mundo
	-(void) limparConsole	
38	public void mudarPosicao(final int x, final int y)	Chama o método mudarPosicao(ObjetoMundoImpl, int, int) do objeto mundo
	-(void) mudarPosicao:(NSInteger) ix py:(NSInteger) iy	
39	public ObjetoMundoImpl(ObjetoDoMundo objetoMundo)	Cria objetoMundoImpl e associa ao objetoDoMundo
	-(id) init	
40	public void parar()	Para a execução do objeto
	-(void) parar	
41	public void pararMundo()	Para a execução do mundo
	-(void) pararMundo	
42	public void removerMe()	Remove-se do mundo
	-(void) removerMe	
43	public void repintar()	Repinta o objeto no mundo

	-(void) repintar	
44	public void run()	Chama o método inteligencia()
	-(void) run	
45	public void setBloqueado(boolean bloqueado)	Define se o objeto está bloqueado ou não na célula
	@property (nonatomic, assign, getter=isBloqueado) BOOL bloqueado	
46	public void setEnergia(int energia)	Define quantidade de energia do objeto
	@property (nonatomic, assign) NSInteger energia	
47	public void setExplodiu(boolean explodiu)	Define se o objeto explodiu
	@property (nonatomic, assign, getter=isExplodiu, setter=setExplodir) BOOL explodiu	
48	public final void setImage(ImageIcon imagem)	Define a imagem do objeto
	@property (nonatomic, retain) CocosNode *image	
49	public void setMaxEnergia(int maxEnergia)	Define o máximo de energia do objeto
	@property (nonatomic, assign) NSInteger maxEnergia	
50	public void setMundo(Mundo mundo)	Associa o mundo ao objeto
	@property (nonatomic, retain) Mundo *mundo	
51	public void setRemovido(boolean removido)	Define se o objeto foi removido do mundo
	@property (nonatomic, assign, getter=isRemovido) BOOL removido	
52	public void setTempoEspera(int milisegundos)	Define o tempo de espera padrão do objeto
	-(void) setTempoEspera:(double) segundos	
53	public void setUsarEnergia(boolean usarEnergia)	Define se objeto usa energia
	@property (nonatomic, assign, getter=isUsarEnergia) BOOL usarEnergia	
54	public void setX(int x)	Define a posição X onde o objeto se encontra
	@property (nonatomic, assign) NSInteger x	
55	public void setY(int y)	Define a posição Y onde o objeto se encontra
	@property (nonatomic, assign) NSInteger y	

Quadro 46 – Detalhamento da classe `ObjetoMundoImpl` em Objective-C

O Quadro 47 demonstra os métodos da classe `Exercicio` comparando a sua versão em Java com Objective-C.

Exercicio		
ID	**MÉTODO**	**DESCRIÇÃO**
1	private void addElementsOfGroups(List<ElementoExercicio> elemsAUsar, List<GrupoObjetos> grupos)	Carrega todos os objetos participantes de grupos que devem ser incluidos no mundo
	-(void) addElementsOfGroups:(NSMutableArray *) elemsAUsar pGrupos:(NSMutableArray *) grupos	
2	public void addRobo(ElementoExercicio robo)	Adiciona robô a lista de objetos a serem adicionados ao mundo
	-(void) addRobo:(ElementoExercicio *) robo	
3	private void adicionarObjetosMundo(boolean random, boolean dependentes, Mundo m, List<String> posUsadas, List<ElementoExercicio> elems)	Cria objetoMundoImpl e depois adiciona ao mundo
	-(void) adicionarObjetosMundo:(BOOL) pRandom pDependentes:(BOOL) dependentes pM:(Mundo *) m pPosUsadas:(NSMutableArray *) posUsadas pElems:(NSMutableArray *) elems	

4	private void calcularQtdadeLinCol() -(void) calcularQtdadeLinCol	Calcula quantidade de linhas e colunas de acordo com os parâmetros do XML de configuração
5	public Mundo criarMundo() -(Mundo *) criarMundo	Cria objeto do mundo e carrega objetos nele
6	private ObjetoMundoImpl criarObjetoMundoImpl(Mundo mundo, List<String> posUsadas, ElementoExercicio elemento, String clazz) -(ObjetoMundoImpl *) criarObjetoMundoImpl:(Mundo *) mundo pPosUsadas: (NSMutableArray *) posUsadas pElemento:(ElementoExercicio *) elemento pClazz:(NSString *) clazz	Cria objetoMundoImpl
7	public Exercicio(String nomeArquivoXML) -(id) initWithXMLFile:(NSString *) nomeArquivoXML	Construtor da classe
8	private ArrayList<ElementoExercicio> extrairElementosPosicao(boolean random, boolean dependentes, List<ElementoExercicio> elemsOrigem) -(NSMutableArray *) extrairElementosPosicao:(BOOL) pRandom pDependentes:(BOOL) dependentes pElemsOrigem:(NSMutableArray *) elemsOrigem	Carrega elementos com as características dos parâmetros passados e retorna em uma lista
9	public void finalizar() -(void) finalizar	Bloqueia robô do mundo
10	public String getEnunciado() @property (nonatomic, retain) NSString *enunciado	Retorna enunciado definido no XML de configuração
11	private ObjetoMundoImpl getObjetoMundoImpl(Mundo mundo, String id) -(ObjetoMundoImpl *) objetoMundoImpl:(Mundo *) mundo pId:(NSString *) idz	Retorna objeto do mundo que possui o id passado como parâmetro
12	public int getQtdadeCol() @property (nonatomic, assign) NSInteger qtdadeCol	Retorna quantidade de colunas que deve ter o mundo
13	public int getQtdadeLin() @property (nonatomic, assign) NSInteger qtdadeLin	Retorna quantidade de linhas que deve ter o mundo
14	public FurbotRandom getRandom() @property (nonatomic, retain) FurbotRandom *random	Retorna objeto de random
15	public String getTamanhoCel() @property (nonatomic, retain) NSString *tamanhoCel	Retorna o tamanho da célula que deve ter o modelo de dados
16	private int getX(Mundo mundo, ElementoExercicio elemento) -(NSInteger) x:(Mundo *) mundo pElemento:(ElementoExercicio *) elemento	Retorna posição X do objeto ElementoExercicio
17	private int getY(Mundo mundo, ElementoExercicio elemento) -(NSInteger) y:(Mundo *) mundo pElemento:(ElementoExercicio *) elemento	Retorna posição Y do objeto ElementoExercicio
18	public boolean isExplodir() @property (nonatomic, assign, getter=isExplodir) BOOL explodir	Retorna se foi definido no XML para o mundo explodir
19	public boolean isUsarLinhasNaGrade() @property (nonatomic, assign, getter=isUsarLinhasNaGrade) BOOL usarLinhasNaGrade	Retorna se foi definido no XML para usar linhas de grade
20	private void refazerPosicaoDependente(java.awt.Point p, Mundo mundo, ElementoExercicio elemento)	Calcula posição X e Y do objeto do mundo caso no elementoExercicio esteja definido que o objeto é dependente

	-(void) refazerPosicaoDependente:(Ponto *) p pMundo:(Mundo *) mundo pElemento:(ElementoExercicio *) elemento	
21	public void setClassRobo(String classRobo)	Define a classe dos robos
	-(void) setClassRobo:(NSString *) clasRobo	
22	public void setEnunciado(String enunciado)	Define o enunciado
	@property (nonatomic, retain) NSString *enunciado	
23	public void setExplodir(boolean explodir)	Define se o mundo pode explodir
	@property (nonatomic, assign, getter=isExplodir) BOOL explodir	
24	public void setQtdadeCol(int qtdadeCol)	Define a quantidade de colunas
	@property (nonatomic, assign) NSInteger qtdadeCol	
25	public void setQtdadeLin(int qtdadeLin)	Define a quantidade de linhas
	@property (nonatomic, assign) NSInteger qtdadeLin	
26	public void setRandom(FurbotRandom random)	Define o objeto de random
	@property (nonatomic, retain) FurbotRandom *random	
27	public void setTamanhoCel(String tamanhoCel)	Define o tamanho da célula
	@property (nonatomic, retain) NSString *tamanhoCel	
28	public void setUsarLinhasNaGrade(boolean usarLinhasNaGrade)	Define se deve usar linhas de grade no modelo
	@property (nonatomic, assign, getter=isUsarLinhasNaGrade) BOOL usarLinhasNaGrade	

Quadro 47 – Detalhamento da classe `Exercicio` em Objective-C