INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

EDUARDO BACKES

DESENVOLVIMENTO DE JOGO DIGITAL EM DUAS DIMENSÕES PARA ARQUITETURA ANDROID

Fernando Afonso Abrahão

PASSO FUNDO, 2014

EDUARDO BACKES

DESENVOLVIMENTO DE JOGO DIGITAL EM DUAS DIMENSÕES PARA ARQUITETURA ANDROID

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Fernando Abrahão Afonso

PASSO FUNDO, 2014

EDUARDO BACKES

DESENVOLVIMENTO DE JOGO DIGITAL EM DUAS DIMENSÕES PARA ARQUITETURA ANDROID

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Nome do Professor(a) Orientador(a)

_____

Nome do Professor(a) Convidado(a)

_____

Nome do Professor(a) Convidado(a)

_____

Coordenação do Curso

PASSO FUNDO, 2014

AGRADECIMENTOS

Aos meus pais que sempre estiveram ao meu lado me apoiando e incentivando, aos meus amigos, professores e colegas do curso e especialmente aos professores Fernando Abrahão Afonso e Silvani Lopes Lima pelo tempo despendido comigo e com este trabalho, conselhos e oportunidades de aprendizagem.

EPÍGRAFE

“Se você pensa que pode ou se você pensa que não pode,
de qualquer forma, você tem toda a razão.”
Henry Ford

RESUMO

Os jogos eletrônicos vêm evoluindo ao passo que a tecnologia evolui, portanto, à medida que novos aparelhos vão aparecendo no mercado, os jogos ganham inovações que se ajustam a esses aparelhos. Nesse contexto, uma categoria de jogos que vem crescendo exponencialmente no mercado são os jogos para dispositivos móveis, com ênfase para o sistema operacional Android. Logo, este trabalho consiste em estudar quais são as facilidades e dificuldades encontradas quando se deseja desenvolver jogos para essa plataforma, utilizando a *engine* AndEngine, que é código aberto, mantida pela comunidade, e foi desenvolvida exclusivamente para Android.

Palavras-chave: Android; AndEngine; engine; motor de jogo; jogos; jogo.

ABSTRACT

The electronic games has been evolving in the same way that the technology come evolving, the new devices appear in the market and the games come to evolving together. The games category that is growing up exponentially in the market are the games for mobile devices with more emphasis for the Android operational system. That work is about study what are the facilities and the difficulties founded when we want to develop games for this platform, using the AndEngine that is an engine for developing games, its open-source, maintained for the community and it developed only for Android.

Key words: Android, AndEngine, engine, games, game.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS (OPCIONAL)

2D – Duas Dimensões

3D – Três Dimensões

ASF – Apache Software Foundation

API – Application Programming Interface

IDC – International Data Corporation

MIT – Massachusetts Institute of Technology

RPG – Role-playing Game

SO – Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

Fazer um personagem ganhar “vida”, dar “asas” à imaginação e criar obras de arte. Melhor ainda, fazer uma obra na qual as pessoas possam interagir com a arte, fazer um filme em que os telespectadores são os diretores da história, fazer com que as pessoas em questão de minutos se tornem poderosos líderes da máfia italiana ou, se preferirem, pilotos de Fórmula 1, tudo isso sem sair de casa.

Isso pode ser conquistado através dos jogos. Um jogo em si não é somente uma forma de entretenimento, é arte, uma forma de expressão de ideias ou de visões do mundo e até mesmo expressões de indignação, tudo criado por designers, roteiristas e programadores. Além das expressões artísticas que podem ser transformadas em jogos, pode-se também conscientizar pessoas através dos jogos. Em um mundo onde grande parte da população tem acesso às novas tecnologias e à informação, pouco se pensa em qualidade de vida e meio ambiente, por exemplo, algo que pode ser trabalhado através de jogos. Nesse sentido, podem-se usar essas tecnologias em prol da educação, da conscientização e, com isso, atingir uma gama de pessoas das mais variadas idades e países.

Devido à popularidade dos dispositivos móveis, criou-se um mercado de jogos eletrônicos especifico para esse tipo de aparelho. Voltando os olhares para a plataforma Android, que é largamente utilizada nos aparelhos móveis, porque é uma plataforma *open-source*, o que diminui o custo dos aparelhos para os usuários e facilita o acesso à tecnologia para os desenvolvedores. O fato é que existe aí um nicho de mercado em ascensão, no Brasil, não é diferente.

De acordo com uma pesquisa realizada no ano de 2012, pela empresa Newzoo, o Brasil é o país que mais cresce no mundo em relação a jogos eletrônicos. No país, já existem mais de 40 milhões de jogadores ativos (Figura 1), destes, 54% gastam dinheiro com jogos eletrônicos. O infográfico da pesquisa aponta que a estimativa de dinheiro gasto com jogos foi de 2.645.000.000 dólares, ou seja, 32% a mais do que no ano de 2011, sendo 285 milhões destes gastos com jogos para dispositivos móveis (NEWZOO, 2012).

Figura 1 - Número de jogadores ativos no Brasil

Fonte: Newzoo, 2012.

De acordo com esta mesma pesquisa, são gastos cerca de 73.000.000 de horas com jogos eletrônicos no Brasil. Sendo que 15% deste tempo é gasto com jogos para dispositivos móveis.

Ainda de acordo com o escritório de pesquisas IDC, o sistema operacional Android dominou o mercado dos smartphones no terceiro trimestre no ano de 2012. O SO Android estava presente em três de cada quatro telefones neste mesmo trimestre, o que representa 75% do mercado mundial (ABRIL, 2012).

No Brasil, ainda são poucas as empresas que trabalham especificamente para este ramo da computação, porém, a cada dia se faz mais necessário um número maior de profissionais para este mercado em ascensão.

Considerando todas essas boas estatísticas e prospecções sobre o mercado de jogos eletrônicos, é importante que se pense novas tecnologias, através de novas plataformas, para o

desenvolvimento de jogos. Nesse sentido, o desenvolvimento de jogos 2D para a plataforma Android proposto neste trabalho atende a uma demanda do mercado.

Estas pesquisas são de grande valia para as gerações futuras, pois, por muitos momentos, os profissionais deparam-se com a pouca ou nenhuma informação sobre essas tecnologias, tendendo assim a desistirem e partirem para outra parte do mercado onde existe um maior *know-how* sobre a área.

Este trabalho será divido da seguinte forma, primeiro serão mostrados como os jogos são projetados e gerenciados, técnicas utilizadas na produção de jogos em geral, conceitos sobre aplicações e programação para a plataforma Android, estudo sobre o motor de jogo AndEngine, estudo sobre a extensão Box2D Extension que trata da física dos objetos no jogo e por fim o protótipo e como foi o projeto.

1.1 MOTIVAÇÃO

A motivação para esse trabalho é a ascendência da computação móvel e do grande mercado envolvido exclusivamente em torno dos jogos eletrônicos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um protótipo de um jogo em 2D para a plataforma Android usando o motor de jogo *AndEngine*.

1.2.2 Objetivos específicos

- Pesquisar sobre o desenvolvimento de jogos eletrônicos, assim como sobre as técnicas e tecnologias utilizadas no desenvolvimento de jogos em 2D.
- Estudar sobre o desenvolvimento de aplicativos para dispositivos móveis, em específico para a plataforma Android.
- Fazer um estudo do motor de jogo *AndEngine* para utilizá-lo posteriormente no desenvolvimento do protótipo.
- Desenvolver um protótipo de jogo em 2D, utilizando o motor de jogo *AndEngine* para a plataforma Android.
- Testar e analisar os resultados do protótipo desenvolvido.

2 DESENVOLVIMENTO

Nesta seção serão apresentados os conceitos sobre o desenvolvimento de jogos eletrônicos, como são desenvolvidos e planejados. Também serão apresentados conceitos sobre desenvolvimento de aplicativos para a plataforma Android e o desenvolvimento de jogos 2D com a utilização do motor de jogo AndEngine.

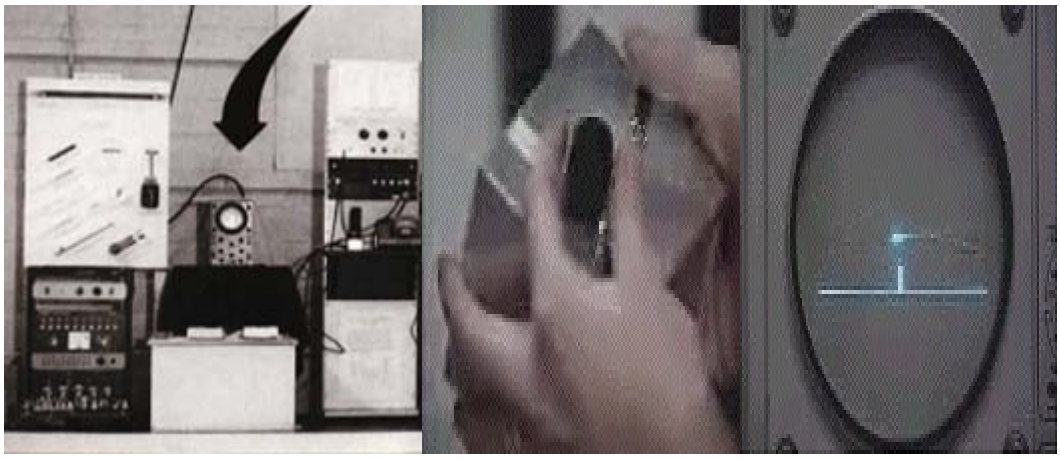
2.1 PRIMÓRDIOS

É sempre bom saber de onde veio todo o *know-how* sobre jogos eletrônicos, e como todas as invenções, os jogos eletrônicos começaram com ideias simples e sem presunções e as gerações posteriores encarregaram-se de aperfeiçoar essas ideias e criar um mercado bilionário.

De acordo com Terra(2005), há muitas contradições, porém, há um consenso hoje de que o primeiro jogo eletrônico da história foi criado em 1958 pelo físico Willy Higinbotham. A principal finalidade do jogo era atrair visitantes para o Brookhaven National Laboratories, no estado de New York. O jogo foi um sucesso e por muitos meses foi a atração mais cobiçada pelo público que visitava o laboratório.

O jogo era um simples jogo de tênis que era mostrado em um osciloscópio e processado por um computador analógico. Mais tarde o cientista aperfeiçoou o jogo que recebeu o nome de Tennis Programming, também conhecido como Tennis for Two (Figura 2), adaptando-o para ser mostrado em um monitor de 15 polegadas

Figura 2 – Tennis for Two.



Fonte: TERRA, 2005.

Em 1961, alguns estudantes do MIT (Massachusetts Institute of Technology) desenvolveram o SpaceWar em Assembly. O objetivo dos estudantes Martin Graetz, Sthephen Russell, Peter Samson, Dan Edwards, Alan Kotok, Steve Piner e Robert A. Saunders era, mais uma vez, chamar atenção do público que visitava as instalações do MIT.

A primeira versão do Spacewar(Figura 3) rodou em um computador DEV PDP-1 em 1962, o jogo era basicamente duas espaçonaves que dois jogadores controlavam em um ambiente escuro e o objetivo era abater o adversário, nesse jogo foram utilizados conceitos de física real, como aceleração e gravidade. Esse jogo foi feito para ser jogado e por isso acabou sendo considerado o primeiro jogo interativo de computador, mas hoje se sabe que o pioneiro nos jogos eletrônicos foi Willy Higinbotham com o Tennis Programming.

Figura 3 – SpaceWar.

Fonte: TERRA, 2005.

A Brown Box foi criada em 1968 por Ralph Baer, essa máquina podia rodar jogos eletrônicos utilizando uma TV. Custava pouco e podia ser usada por qualquer pessoa que quisesse se divertir. Em 1967 surgiu o primeiro esboço de sua ideia, o *chasing game* (Figura 4), um rudimentar jogo de Ping Pong que Baer patenteou e apresentou em 1968 o protótipo do videogame chamado de Brown Box, que rodava jogos de futebol, voleibol e de tiro.

Figura 4 – *Chasing game.*

Fonte: TERRA, 2005.

2.2 COMO OS JOGOS SÃO FEITOS

A produção de jogos eletrônicos não consiste somente em programação, para o desenvolvimento de bons jogos, demanda-se tempo e muita pesquisa. O desenvolvimento de jogos é uma tarefa que envolve profissionais de diversas áreas de formação, em jogos grandes como, por exemplo, GTA IV, são necessários profissionais que tenham conhecimento para desenvolver o design, o roteiro, a arte e a mídia eletrônica do jogo, além de definir música, efeitos sonoros, narração, programação, administração de recursos, coordenação de grupos, entre outros.

Antes de começar a desenvolver um jogo, é necessário ter conhecimento de como eles são feitos, desde a ideia inicial até o modelo final. Antes de tudo, deve-se conhecer algumas terminologias usadas entre os desenvolvedores de jogos. Como o enfoque deste trabalho são os jogos em 2D, serão explanados conceitos sobre desenvolvimento para esta categoria de jogos eletrônicos.

O principal na produção de um jogo é saber que, como qualquer outro software, porém com mais intensidade, é necessária a adoção de processos de desenvolvimentos para desenvolvê-los.

2.2.1 Design Bible

A *Design Bible* é um documento onde estão relacionadas todas as especificações do jogo, como se fosse um manual de instruções para os desenvolvedores. “De fato, tão importante é este documento, que o processo de desenvolvimento não pode começar sem que

esse esteja pronto.” (CLU e BITTENCOURT, 2005. p. 1327). Neste manual de instruções devem constar os seguintes elementos:

a) Roteiro

Com a crescente evolução dos jogos, cada vez mais este elemento se assemelha aos filmes. Segundo Clua e Bittencourt (2005), este é um item fundamental para o processo de criação e será elemento crucial para convencer os investidores da potencialidade do produto.

Este roteiro é bastante parecido com o roteiro de um filme, porém com o diferencial de que deve deixar espaço para a interferência do jogador no desencadeamento da história, levando em consideração os diversos rumos que a história pode tomar de acordo com as escolhas e ações do jogador. Assim como nos filmes, ao elaborar o roteiro, deve-se também levar em consideração o público-alvo para quem o jogo se destina.

b) Game Design

É a conceituação artística do jogo. Dada à complexidade dos cenários elaborados nos jogos atuais, é necessário que esta parte do *Design Bible* seja escrita por um artista. Nesta parte do documento são expostas as principais características dos cenários, esboços de personagens, descrição das texturas fundamentais, mapas e descrições das fases.

c) Game Play

Neste tópico são descritos aspectos sobre a jogabilidade, subentende-se que jogabilidade são as regras do jogo. Nesta parte é essencial descrever que o jogo é algo divertido e que possui desafios interessantes. Este item é de suma importância para guiar os programadores, principalmente na etapa de codificação.

d) Interface Gráfica

Pode ser dividida em *ingame* e *outgame*. Na primeira, são descritos todos os aspectos da entrada de dados e da instrumentação disponível durante o jogo. A segunda é responsável por descrever como serão os menus de entrada, configuração, instrumentação, jogos salvos, carregamento de jogos salvos, ou seja, todas as operações de suporte que o jogo possa vir a ter.

2.2.2 Planejamento

Segundo Perucia et al. (2005), antes de começar a desenvolver o jogo em si é importante planejar. Qualquer modificação “brusca” no rumo do jogo durante o desenvolvimento pode ser muito prejudicial ao resultado. Quanto mais tempo é gasto com o planejamento do jogo, mais tempo se economiza posteriormente, durante o processo de produção do mesmo; além do mais, é possível fazer uma validação do projeto antes mesmo de o jogo ser produzido.

Além disso, é preciso ter em mente que um jogo não é um aplicativo comum, ao planejar um jogo, deve-se levar em conta que ele está sendo desenvolvido não apenas para atender determinados requisitos, mais sim, está sendo planejado e desenvolvido para o entretenimento das pessoas. O jogador precisa sentir-se parte do jogo, a chamada imersividade, mesmo em jogos casuais, onde os jogos não são elaborados com tecnologias de ponta.

2.3 CLASSIFICAÇÃO DOS JOGOS

Segundo Lopes (2006), os jogos são classificados para que possam ser diferenciados, de forma a facilitar o estudo dos jogos e suas peculiaridades. Os jogos são classificados pelos seguintes critérios:

- **Dimensionalidade:** Esse critério considera o número de dimensões em que o jogo se expressa, ou seja, são classificados levando em consideração se os jogos são em 2D (duas dimensões – Figura 5) ou em 3D (três dimensões – Figura 6).

Figura 5 – Jogo em 2D. Street Fighter Alpha 2 (Capcom., 1998).

Fonte: UOL Jogos, 2012.

Figura *6* – Jogo em 3D. The Legendo of Zelda: Ocarina of Time (Nintendo, 1998).

Fonte: LOPES, 2006.

- **Ponto de Vista:** Considera a posição da "câmera" em relação ao personagem que o jogador controla, essa classificação é subdividida em:
    - **Primeira Pessoa:** O jogador observa o jogo pelo ponto de vista do próprio personagem que ele controla. Essa característica contribui muito para a identificação do jogador com o personagem e para o sentido de imersão (Figura 7).

Figura *7* – Jogo em primeira pessoa. No One Lives Forever 2 (Monolith, 2002).

Fonte: LOPES, 2006.

    - **Terceira Pessoa:** O jogador observa o personagem com uma perspectiva diferente, encara o personagem que manipula como sendo "ele", daí o

termo “terceira pessoa”, ou seja, acompanha a história e controla o personagem com uma perspectiva “de fora” (Figura 8).

Figura *8* – Jogo em terceira pessoa. Spider-Man: The Movie (Gray Matter, 2002).

Fonte: LOPES, 2006.

- **Gênero:** Essa classificação subdivide-se em subgêneros devido à grande variedade de características e jogabilidade semelhantes. Os gêneros são: ação, aventura, estratégia, RPG, esporte, simulação, tabuleiro e quebra-cabeças. E ainda dentro de cada gênero, os jogos também podem ser subdivididos dentro das categorias citadas.
- **Por número de jogadores:** Considera o número máximo de jogadores simultâneos na mesma partida de um jogo, essa classificação subdivide-se em:
  - ***Single player*:** Um jogo *single player* é aquele em que apenas um jogador participa da partida;
  - ***Multiplayer:*** Muitos jogadores participam da mesma partida, podendo fazê-lo de modo cooperativo ou competitivo.
  - ***Massive player:*** São jogos em que centenas de jogadores podem participar ao mesmo tempo, esse tipo de jogo geralmente possui servidores distribuídos e contam com uma cobrança mensal. Um exemplo de *massive player* é o jogo Tíbia, um RPG gratuito *online*.

2.4 SPRITES

De acordo com Schoeder e Broyles (2013, p. 62), este pode ser considerado o aspecto mais necessário para o desenvolvimento de qualquer jogo 2D. *Sprites* podem ser usados tanto para mostrar imagem de botões, personagens, ambientes, *background* e qualquer outra entidade do jogo que necessite de uma representação na forma de imagens.

A criação de *sprites* é uma técnica amplamente usada em jogos em 2D e em alguns jogos em 3D. Segundo Perucia et al. (2005), essa técnica consiste em fazer um documento com todos os movimentos e estados possíveis de um objeto em cena, ou seja, desenhar cada movimento ou alteração, um após o outro, e depois apresentá-los quadro a quadro na cena. Geralmente *sprites* são criados para cada personagem em cena, dependendo da necessidade da aplicação.

Os *sprites* possuem a chamada cor de transparência, essa cor poderá ser qualquer cor, desde que não seja a usada nas telas de fundo, sendo substituída pelos pixels que estiverem no *background* da cena. Sem essa camada, os quadros seriam visualizados com um retângulo em sua volta.

Esse conceito de animação com *sprites* vem dos antigos desenhos animados, onde todos os movimentos dos personagens e objetos em cena eram feitos utilizando esta técnica.

As boas técnicas de animação com a utilização de *sprites* dizem que quando o *Sprite* é apresentado em *loop* infinito, por exemplo, para fazer o personagem parecer estar correndo, o primeiro e o último quadro do *Sprite* devem ser idênticos para que aconteça uma harmonia e encaixe na cena. Já em cenas que somente acontecerão uma vez, por exemplo, uma explosão, não é necessária a utilização deste conceito.

2.5 PARALLAX SCROLLING

É uma técnica de deslizamento que usa camadas que se movem em velocidades diferentes para formar o cenário do jogo. Com esta técnica, o cenário dá a impressão de profundidade, criando um pseudo-3D por ter mais de um ponto de referência. A *Parallax Scrolling* tem como princípio posicionar as camadas e exibi-las em diferentes velocidades.

De acordo com Schroeder e Broyles (2013, p. 120), aplicar essa técnica no *background* de um jogo 2D pode resultar em um agradável efeito de perspectiva. Podem-se criar *backgrounds* que darão a ilusão de profundidade através do uso dessa técnica que define a velocidade do movimento dos *sprites* baseado no movimento da câmera.

2.6 DETECÇÃO DE COLISÕES

É uma técnica utilizada por todos os jogos, essencialmente a detecção de colisões é o que faz com que aconteça uma “interatividade” entre os objetos em cena. Existem inúmeras formas de detectar uma colisão em um jogo eletrônico, dentre todas estas formas de detecção existem, segundo Zanardo (s.d), quatro delas que são mais comumente utilizadas.

- **Colisão por *Bounding-box:*** Essa técnica consiste basicamente em fazer retângulos imaginários ao redor dos objetos e depois verificar se esses retângulos colidiram, essa é uma técnica boa e leve, mas pode não apresentar um bom resultado quando os objetos têm uma forma complexa ou circular.
- **Colisão entre dois círculos:** Para detecção de colisão entre dois círculos, a técnica de *Bounding-Box* não apresentaria o resultado esperado, para isso existe a técnica de detecção de colisões entre dois círculos, essa técnica consiste em verificar a distância entre os centros dos círculos que se deseja verificar se a colisão ocorreu ou não, se essa distância for maior que a soma dos raios dos círculos então não ocorreu a colisão.
- **Colisão por *Pixel Perfect:*** É uma das técnicas de menor desempenho, porém, é a técnica de melhor precisão de todas, em todos os casos a colisão é detectada perfeitamente. Consiste em verificar cada *pixel* da imagem para saber se a parte do desenho da imagem colidiu ou não com outro desenho.
- **Colisão entre polígonos:** Em casos em que as formas não são complexas ao ponto de usar a técnica de *Pixel Perfect* e que o efeito da colisão por *Bounding-Box* não ficaria bom, existe a técnica de detecção de colisão entre polígonos. Essa técnica é parecida com a técnica de *Bounding-Box*, mas, ao invés de usar quadrados, são usados polígonos, esses polígonos envolvem o objeto ou os objetos nos quais se quer testar a colisão.

2.7 ENGINES

Depois de ver estes conceitos e técnicas para tratar efeitos como gravidade, física, detecção de colisões e outros, se pensa que não é viável desenvolver jogos eletrônicos, pois

são muitas questões a serem tratadas antes mesmo de pensar em começar a desenvolver um jogo. Para resolver este problema existem as *engines*.

Engines são bibliotecas com um pacote de funcionalidades que auxiliam no desenvolvimento dos jogos, evitando assim que toda a criação tenha que ser feita do zero. Essa biblioteca é responsável normalmente pela modelagem em 2D ou 3D, itens de jogabilidade, como o sistema de detecção de colisões, inteligência artificial, gravidade, entre outros elementos. As *engines* auxiliam e muito no desenvolvimento de jogos eletrônicos, pois, não se torna necessário reinventar a roda toda a vez que se quer desenvolver um novo jogo. Além disso, com o auxílio de *engines* fica muito fácil de migrar um jogo de uma plataforma para outra. (KLEINA, 2011).

De acordo com Kleina (2011), os primeiros jogos a utilizarem uma *engine* de outro companhia foram *Driller* (1987) e *Dark Side*, do ano seguinte, a *engine* utilizada por estes jogos foi a Freescape, da Incentive Software. Mas o termo "*engine*" surgiu somente em meados da década de 1990.

Atualmente existem muitos *frameworks* de desenvolvimento de jogos, a maioria deles são principal ou exclusivamente para *desktops*. Os mostrados na tabela 1 são reconhecidos como utilizáveis no Android.

Tabela 1 – *Frameworks* de jogos para Android

Nome	Código-Aberto	Preço	URL
AndEngine	Sim	Gratuito	http://www.andengine.org
Box2D	Sim	Gratuito	http://code.google.com/p/box2d/
Cocos2D	Sim	Gratuito	http://www.cocos2d-x.org/download/
Corona SDK	?	US$ 199+/ano	http://www.anscamobile.com/corona/
Flixel	Sim	Gratuito	http://flixel.org/
Libgdx	Sim	Gratuito	http://code.google.com/p/libgdx/
PlayN	Sim	Gratuito	http://code.google.com/p/playn/
Rokon	Sim	Gratuito	http://code.google.com/p/rokon/
ShiVa 3D	Não	£ 169+ por cada editor e servidor	http://www.stonetrip.com/
Unity	Não	$ 400+	http://unity3d.com/unity/publishing/android.hmtl

Fonte: DARWIN, 2012.

2.8 JAVA

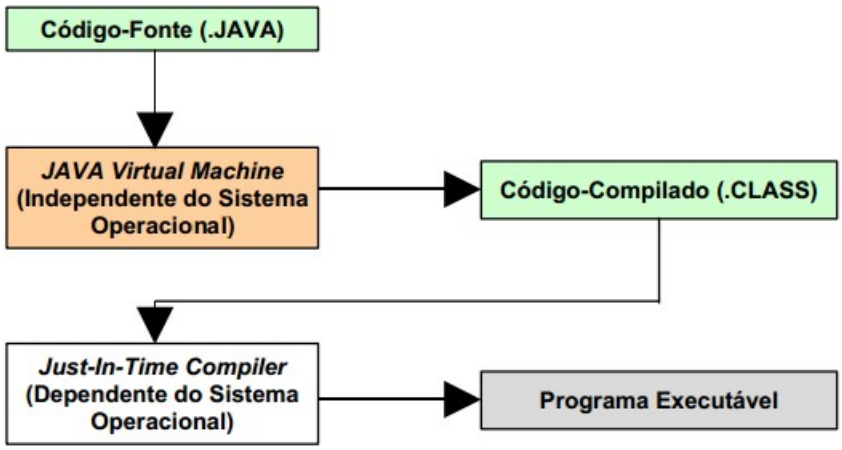
Lançada em 1995 pela Sun Microsystems e hoje mantida pela Oracle, é uma linguagem de programação multiplataforma que, de acordo com a Oracle, é executada em mais de 850

milhões de computadores pessoais e em bilhões de dispositivos em todo o mundo, inclusive telefones celulares e dispositivos de televisão.

“JAVA é uma tecnologia que permite a construção otimizada de aplicações distribuídas, a partir das quais múltiplos computadores podem ser acessados através de uma rede por múltiplos usuários simultaneamente.” (SANTOS JÚNIOR, 2006, p. 3).

O fato de a linguagem JAVA ser multiplataforma faz com que um programa escrito nesta linguagem possa ser executado em diferentes sistemas operacionais sem a necessidade de alterações no código-fonte do programa. (Figura 9). (SANTOS JÚNIOR, 2006).

Figura *9* – Esquema genérico de compilação de programas JAVA.



Fonte: SANTOS JÚNIOR, 2006.

2.9 ANDROID

O Android é uma plataforma desenvolvida pela Google, em conjunto com outras grandes empresas do ramo de telefonia e tecnologia, consiste em uma plataforma de desenvolvimento para aplicativos móveis baseada no sistema operacional Linux, com código aberto (*open-source*) e com um ambiente de desenvolvimento poderoso e flexível (LECHETA, 2013). A plataforma foi desenvolvida utilizando a linguagem de programação Java, é uma linguagem orientada a objetos, independente de plataforma, mantida atualmente pela empresa Oracle.

O fato de ser de código aberto faz com que o Android tenha uma enorme vantagem para quem desenvolve aplicações para essa plataforma e até mesmo para os usuários finais, pois profissionais do mundo inteiro podem fazer alterações, mudando funções, características,

implementações ou simplesmente corrigindo erros (*bugs*), o que significa que o Android está em constante melhoria. Isso também é excelente para os fabricantes de dispositivos móveis, uma vez que os mesmos podem utilizar o sistema operacional Android em seus aparelhos sem ter que pagar nada por isso, a licença Apache Software Foundation (ASF) permite que alterações possam ser feitas no código fonte do Android para criar produtos customizados sem a necessidade de compartilhas as alterações.

As aplicações para Android não têm um único ponto de entrada como o método *main*, utilizado em muitos outros sistemas, essas aplicações possuem quatro tipos de componentes essenciais que o SO pode instanciar (ANDROID, 2013). São eles:

- ***Activity:*** Responsável pela criação da tela do aplicativo Android, podendo um aplicativo ter várias *Activities*, por exemplo, uma aplicação de *email* pode ter uma *Activity* para mostrar a lista de *email*, outra para escrever um *email*, entre outras;
- ***Service:*** É um componente que executa em *background* para executar operações de longa duração ou para realizar tarefas de processos remotos, também é utilizado para realizar tarefas em paralelo a uma *Acitivity* e não possui uma interface visual, por exemplo, enquanto o usuário escuta uma música ele pode usar outros aplicativos, ou pode ser utilizado para realizar algum processo sem bloquear as interações do usuário com a *Activity*.
- ***Content providers:*** Este é o componente responsável por gerenciar o compartilhamento de dados entre as aplicações. Os dados podem ser persistidos em um arquivo de sistema, em um banco de dados SQLite, na Internet, ou em qualquer unidade de persistência que o aplicativo possa acessar. Através deste componente, outras aplicações podem fazer consultas ou modificar dados, desde que o *Content provider* permita.
- ***Broadcast receivers:*** Este componente funciona como um gatilho e fica aguardando algum evento ocorrer como, por exemplo, a chegada de uma mensagem de texto. Ao ser notificado da ocorrência do evento, o *broadcast receiver* é ativado para tratá-lo, como, por exemplo, através da execução de uma *Activity*, para iniciar uma aplicação que trate a mensagem.

Outro ponto muito interessante do Android é que o sistema foi projetado para que qualquer aplicação possa se utilizar da câmera simplesmente para tirar uma foto, por exemplo se a aplicação necessita utilizar a câmera para tirar a foto, quando a foto for tirada, ela é

retornada à aplicação que pode manipulá-la. Para o usuário final parece que a câmera está incorporada na aplicação, quando na verdade ela usou outro aplicativo para fazê-lo.

2.10 GOOGLE PLAY GAMES SERVICES

As APIs são interfaces de programação criadas com a missão de facilitar o trabalho dos desenvolvedores, recentemente na abertura do evento Google I/O 2013 (2013), foram apresentadas novas APIs que tornam mais fácil para os desenvolvedores incluir nos jogos recursos como *saves* armazenados na nuvem e sincronizados entre múltiplos dispositivos, conquistar, rankings globais e uma boa dose de integração com o Google+. O Google+ é o cerne dos recursos sociais das novas APIs.

2.11 GOOGLE PLAY

O antigo Android Market hoje chamado de Google Play é uma loja virtual de aplicativos para o SO Android. Nele são disponibilizados jogos, papéis de parede e todas as demais aplicações, gratuitas ou pagas. Para ter acesso ao Google Play é necessário possuir uma conta do Google, desta forma é possível acessar o Google Play de qualquer dispositivo e ter acesso aos aplicativos, até mesmo os que já foram comprados com outro aparelho. (CÂMARA, 2012).

O Android Play não está disponível somente nos dispositivos móveis com Android, também pode ser acessado via *web browser* pelo site da Google.

Para ter acesso ao Google Play a fim de disponibilizar novos aplicativos desenvolvidos, é necessário ter uma conta Google, aceitar os termos do contrato de distribuição do desenvolvedor do Google Play, pagar uma taxa de US$ 25.00 e desenvolver os aplicativos seguindo as normas da Google. Feito isso, o desenvolvedor terá o controle completo de suas aplicações disponibilizadas para os usuários e informações como comentários, avaliações, entre outros.

2.12 AND ENGINE

A AndEngine é uma *engine* de jogos eletrônicos em 2D desenvolvida exclusivamente para a plataforma Android, ela foi criada e desenvolvida por Nicolas Gramlich. Ela auxilia, facilita e abstrai muitas implementações necessárias e essenciais para o desenvolvimento de jogos, como uma boa *engine* ela conta com suporte à detecção de colisões, implementa algumas propriedades físicas como: gravidade, velocidade, elasticidade, entre outros. Esta *engine* possui em especial o auxílio no desenvolvimento de modos *multiplayer*, partículas e como é voltada para dispositivos móveis, também conta com o suporte a *multi-touch*.

É totalmente gratuita, de código aberto e pode ser incorporada a qualquer jogo mesmo que esse seja vendido ou simplesmente disponibilizado para *download* ou outros.

Um diferencial da AndEngine são suas extensões que são criadas pelo próprio Nicolas Gramlich ou pela comunidade e po*stas par*a download para que os desenvolvedores possam fazer uso dessas novas extensões. Isso acaba possibilitando que novas tecnologias e métodos novos sejam implementados e incorporad*os a en*gine sem a neces*sidade d*e releases de novas versões, tornando assim o processo de ev*olução da en*gine muito mais fácil e natural.

2.12.1 Resolution Policy

Devido à grande variedade de tamanhos de telas e proporções de telas a AndEngine possui classes para manipular essas características. Essas classes partem de uma resolução específica e então através de escalas vão manipulando as imagens da aplicação para qualquer dispositivo.

Com essas classes não há necessidade de se preocupar se a resolução da tela do dispositivo condiz com a resolução que foi implementada, pois a *engine* pode alargar ou diminuir os *sprites* e imagens da cena para caber na tela do dispositivo que a aplicação está rodando. Simplesmente basta colocar os *sprites* e imagens nas coordenadas com base na resolução escolhida inicialmente e deixar que a *engine* manipule-as de acordo com a política de resolução escolhida.

Por padrão, a AndEngine traz quatro políticas, mas como seu código é *open-source* pode-se encontrar mais classes para este fim, ou pode-se implementar uma classe com as soluções desejadas. Segundo Schroeder e Broyles (2013, p. 18), as quatro políticas de resolução da AndEngine são:

- **FixedResolutionPolicy:** Permite aplicar um tamanho fixo para a tela da aplicação, independentemente do tamanho da tela do dispositivo a aplicação var iniciar uma tela com o tamanho de largura e altura passadas no construtor da classe;
- **FillResolutionPolicy:** Essa política de resolução é usada quando se quer criar a tela da aplicação que seja do mesmo tamanho da tela do dispositivo, ou seja, a mesma altura e largura da tela do dispositivo. Essa política pode distorcer (alargar) os *sprites* e imagens da aplicação tendo em vista que ela vai ajustar todos os objetos da cena para caberem na tela do dispositivo;
- **RatioResolutionPolicy:** É a melhor alternativa quando se quer obter o tamanho máximo da tela sem perder a qualidade dos *sprites* e imagens da aplicação. Porém, devido à grande variedade de tamanhos e proporções de telas é possível que em alguns dispositivos possam a vir aparecer barras pretas no topo e na borda ou nos lados da aplicação, devido ao fato de não poder "esticar" os *sprites* e imagens da aplicação;
- **RelativeResolutionPolicy:** Com essa política é possível aplicar escalas, ou mais larga ou mais estreita para as telas da aplicação. O fator de escalonamento começa com 1f e é recomendado que não seja maior que 1.8f, pois, caso o fator de escalonamento seja muito grande a aplicação pode fechar sozinha sem alertas de erros ou qualquer coisa do tipo.

2.12.2 Box2D

O Box2D é uma engine *open source* em C++ para simulações de física em 2D. Foi desenvolvida por Erin Catto e está sob a licença zlib.

Para a AngEngine existe uma extensão chamada Physics Box2D Extension que utiliza-se da *engine* Box2D, essa extensão permite que seja implementado conceitos de física nos jogos criados com a AndEngine, ou seja, permite-nos estabelecer e aplicar regras em objetos como: gravidade, inércia, torque, velocidade, testes de colisão, entre outros.

A extensão Box2D é muito importante para o desenvolvimento de jogos mais realistas e com maior aceitação do público, além de facilitar a programação do jogo, pois não é necessário desenvolver algoritmos para simular características da física nos objetos, características como gravidade, aceleração, força, colisões, etc.

2.13 CLEAN THE SEA

O protótipo foi desenvolvido com o intuito de oferecer uma visão mais completa e aprofundada de como são feitos os jogos eletrônicos. O protótipo foi desenvolvido com a AndEngine, mas servirá de como referência para comparação com outras *engines*.

Como todo software, os jogos também precisam de um planejamento inicial e esse planejamento pode ser feito com o auxílio de alguma metodologia, no caso específico dos jogos essa metodologia pode ser a *Design Bible*.

2.13.1 Design Bible

- Roteiro:

  Um peixe que vive no mar, porém no seu dia a dia precisa da ajuda de pessoas que estão dispostas a salvar as vidas marinhas e precisam limpar os oceanos sujos de lixo de outras pessoas.

  O peixe facilmente confunde o lixo que as pessoas jogam no mar com comida, então é de suma importância recolher os lixos do mar antes que o peixe os coma.

- Game Design:

Figura 10 – Sprite do botão jogar

Fonte: Do Autor, 2014.

Figura 11 – Sprite do botão sair

Fonte: Do Autor, 2014.

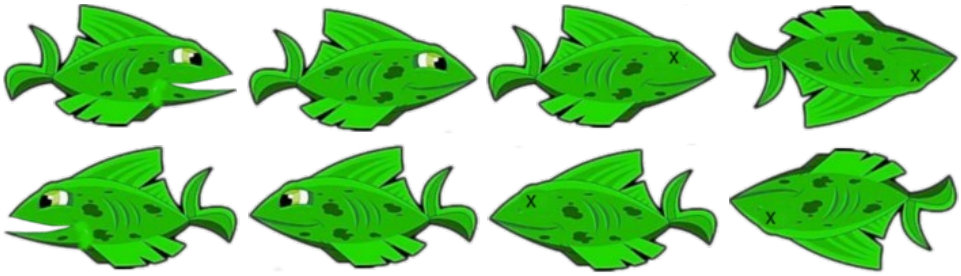
Figura 12 – Sprite dos botões de controle de áudio

Fonte: Do Autor, 2014.

Figura 13 – Imagem que representa o lixo

Fonte: Do Autor, 2014.

Figura 14 – Sprite para as animações do peixe



Fonte: Do Autor, 2014.

- Game Play:

  Enquanto o peixe nada pelo cenário o jogador deve remover os lixos que estão caindo pelo mar, para isso deve-se tocar em cima dos objetos.

  Caso o jogador toque no peixe, o mesmo irá morrer e a partida vai encerrar.

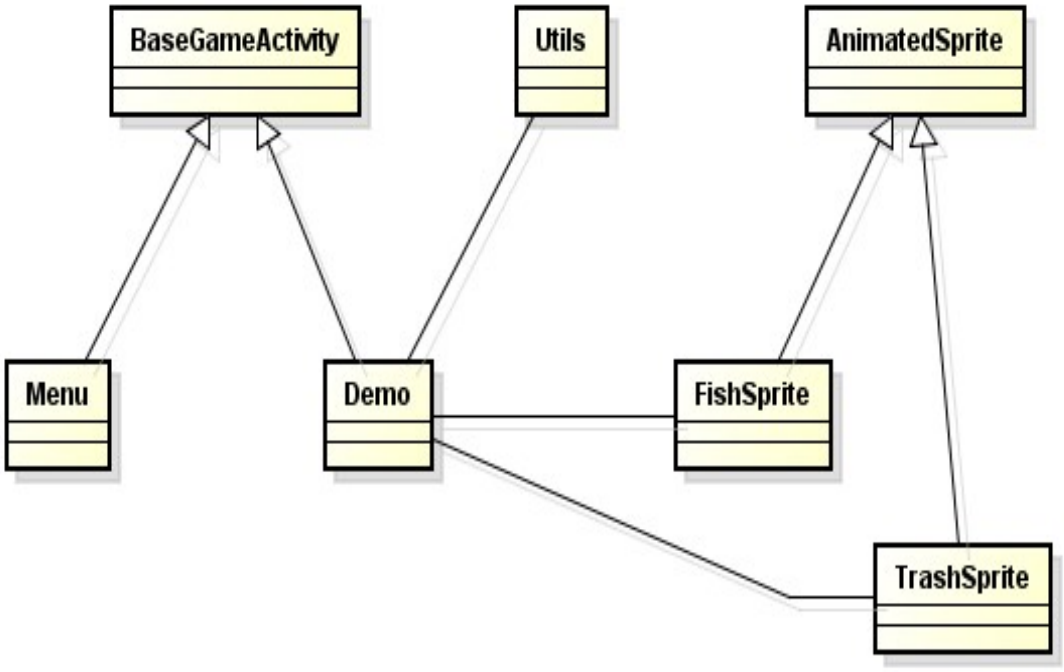
- Interface Gráfica
  - In Game:

    Durante a partida nenhum controle aparece na tela, toda a funcionalidade do jogo é baseada no toque do usuário na tela.
  - Out Game:

    Será um simples menu que mostrará as opções de iniciar uma nova partida ou sair do jogo.

2.13.2 Diagrama De Classes

Figura 15 – Diagrama de classes do trabalho a ser desenvolvido



Fonte: Do Autor, 2014.

2.13.3 Programação

Nesta seção, busca-se aprofundar mais sobre a programação do jogo, mostrando exemplos práticos de como são usadas algumas funções consideradas básicas e explicando o passo a passo de cada uma destas funções.

- **Classes Principais**

  Na versão GLES2 da AndEngine há duas classes principais da AndEngine, a BaseGameActivity e a SimpleBaseGameActivity, essas classes são o ponto inicial de cada tela do jogo. As duas classes têm a mesma finalidade, porém diferenciam-se na maneira como são implementadas.

  A BaseGameActivity, caixa de texto 1, foi totalmente reformulada na versão GLES2 da AndEngine, com essa reformulação o desenvolvedor tem um maior controle sobre os métodos da classe, nenhum dos métodos é chamado automaticamente, todos necessitam que seja acionado o *call-back* do método, esses

call-backs informam à classe que o método terminou sua execução, fazendo então com que o próximo método seja executado.

Caixa de Texto 1 – Implementação da BaseGameActivity

```
public class BaseGameExample extends BaseGameActivity {
	@Override
	public EngineOptions onCreateEngineOptions() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public void onCreateResources(
			OnCreateResourcesCallback pOnCreateResourcesCallback)
			throws Exception {
		// TODO Auto-generated method stub
	}

	@Override
	public void onCreateScene(OnCreateSceneCallback pOnCre-
ateSceneCallback)
			throws Exception {
		// TODO Auto-generated method stub
	}

	@Override
	public void onPopulateScene(Scene pScene,
			OnPopulateSceneCallback pOnPopulateSceneCallback)
throws Exception {
		// TODO Auto-generated method stub
	}

`
```

Fonte: Do Autor, 2014.

Essa mudança ocasionou inúmeras críticas de desenvolvedores que haviam desenvolvido suas aplicações com a versão antiga (GLES1) e tinham de fazer diversas alterações no código-fonte das aplicações para fazer o *upgrade* para a versão GLES2. A solução então foi criar uma classe idêntica a classe da BaseGameActivity da versão GLES1 para a versão GLES2 da *engine*, essa classe ficou conhecida como SimpleBaseGameActivity (Caixa de Texto 2).

Caixa de Texto 2 – Implementação da SimpleBaseGameActivity

```
public class SimpleBaseGameExample extends SimpleBaseGameActivity {
	@Override
	public EngineOptions onCreateEngineOptions() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	protected void onCreateResources() {
		// TODO Auto-generated method stub
	}

	@Override
	protected Scene onCreateScene() {
		// TODO Auto-generated method stub
		return null;
	}

}
```

Fonte: Do Autor, 2014.

- **Física do Jogo**

Um ponto sempre importante na maioria dos jogos são as leis da física simuladas no jogo, na AndEngine tem-se a disposição a extensão Box2D (Caixa de Texto 3) que é responsável por todos os conceitos de física aplicados no jogo. Essa extensão é relativamente fácil de ser usada, porém é necessário ter conhecimento prévio sobre física simulada nos jogos, isso para que os métodos disponíveis na extensão sejam aproveitados da melhor forma possível, beneficiando assim a jogabilidade e o desempenho da aplicação.

Para utilizar a extensão é simples, basta baixá-la e adicioná-la no projeto, depois é necessário criar um atributo na classe na qual se deseja utilizar a extensão, posteriormente inicializá-la dentro do método *onPopulateScene* e por fim passá-la como parâmetro no método *registerUpdateHandler* do objeto *Scene* conforme apresentado na caixa de texto 3:

Caixa de Texto 3 – Implementação da extensão Box2D

```
public class BaseGameExample extends BaseGameActivity {
	private int WIDTH = 800;
	private int HEIGHT = 480;

	private Camera mCamera;
	private Scene mScene;

	private FixedStepPhysicsWorld mPhysicsWorld;

	@Override
	public EngineOptions onCreateEngineOptions() {
		mCamera = new Camera(0, 0, WIDTH, HEIGHT);
		EngineOptions engineOptions = new EngineOptions(true,
ScreenOrientation.LANDSCAPE_FIXED, new RatioResolutionPolicy(1.5f),
mCamera);
		engineOptions.setWakeLockOptions(WakeLockOp-
tions.SCREEN_ON);
		return engineOptions;
	}

	@Override
	public void onCreateResources(
			OnCreateResourcesCallback pOnCreateResourcesCallback)
			throws Exception {
		// TODO Auto-generated method stub
		pOnCreateResourcesCallback.onCreateResourcesFinished();
	}

	@Override
	public void onCreateScene(OnCreateSceneCallback pOnCre-
ateSceneCallback)
			throws Exception {
		mScene = new Scene();
		mScene.setBackground(new Background(Color.GREEN));
		pOnCreateSceneCallback.onCreateSceneFinished(mScene);
	}

	@Override
	public void onPopulateScene(Scene pScene,
			OnPopulateSceneCallback pOnPopulateSceneCallback)
throws Exception {
		mPhysicsWorld = new FixedStepPhysicsWorld(60, new
Vector2(0f, SensorManager.GRAVITY_EARTH / 4), false, 8, 3);
		mScene.registerUpdateHandler(mPhysicsWorld);
		pOnPopulateSceneCallback.onPopulateSceneFinished();
	}
```

Fonte: Do Autor, 2014.

A extensão usa os valores gravitacionais da classe *SensorManager* do Android, essa classe traz uma série de valores de gravidades distintas, mas, como o contexto do protótipo do jogo CleanTheSea se passa embaixo d'agua é preciso simular a micro gravidade da água.

Para simular a micro gravidade da água é necessário dividir o valor da gravidade da Terra (10) por 4 (quatro), causando assim um resultado satisfatoriamente parecido com a micro gravidade da água, mas ainda é necessário modificar as forças de impacto dos objetos para que os mesmos não fique quicando na tela. Essa é uma adaptação da micro gravidade subaquática que gerou resultados satisfatórios para ser implementada como simulação de gravidade final.

- **Imagens e *Sprites***

Os *sprites* e imagens são pontos fundamentais na criação de jogos 2D, eles são quem fazem com que o jogo ganhe forma e "vida". Para o CleanTheSea os *sprites* criados são simples e com poucas variações de movimentos, porém, já dão ao jogo um ar mais "refinado" e realista. A maior dificuldade de quem trabalha com *sprites* utilizando a AndEngine é a grande quantidade de código que é preciso para que um único *sprite* seja mostrado na cena, outro fator que atrapalha é a grande variedade de classes possíveis para implementação dos *sprites*, essa grande quantidade de classes causa uma grande confusão na hora da implementação.

Para adicionar um *sprite* na cena é necessário obedecer a algumas etapas para posteriormente poder manipulá-lo em cena. Primeiro é necessário criar dois atributos, um da classe BuildableBitmapTextureAtlas e o outro da classe TiledTextureRegion.

Caixa de Texto 4 – Atributos do *Sprite*

```
private BuildableBitmapTextureAtlas mFishTextureAtlas;
private TiledTextureRegion mFishTextureRegion;
```

Fonte: Do Autor, 2014.

Posteriormente no método onLoadResources é preciso instanciar os objetos, o objeto BuildableBitmapTextureAtlas é basicamente usado para estipular o tamanho total da imagem que será utilizada e o objeto TiledTextureRegion é onde será estipulado qual será a imagem e quantas colunas e linhas o *sprite* possui, conforme mostra a caixa de texto 5.

Caixa de Texto 5 – Instancia dos objetos do *Sprite*

```
mFishTextureAtlas = new BuildableBitmapTextureAtlas(
                                        getTextureManager(), 900, 250);
mFishTextureRegion = BitmapTextureAtlasTextureRegionFactory
                        .createTiledFromAsset(mFishTextureAtlas,
                                this, "sprite_green_fish.png", 4, 2);
mFishTextureAtlas.build(new
                        BlackPawnTextureAtlasBuilder<IBitmapTextureAtlas-
Source,
                                                        BitmapTextureAtlas>(1, 2, 3));
mFishTextureAtlas.load();
```

Fonte: Do Autor, 2014.

Por fim, basta criar um objeto AnimatedSprite e instanciá-lo informando qual a sua posição na cena e qual o TiledTextureRegion para esse objeto. Para criar a animação do *sprite* basta chamar o método *animate* do objeto AnimatedSprite e passar como parâmetros uma *array* com o tempo entre um *sprite* e outro, qual o *sprite* de início e o de fim e se é para ficar repetindo essa ação. Depois finalmente é só adicioná-lo à cena, como mostrado na caixa de texto 6.

Caixa de Texto 6 – Instancia da classe AnimatedSprite

```
AnimatedSprite sprite = new AnimatedSprite(100, 100, mFishTextureRegion,
                                        getVertexBufferObjectManager());
sprite.animate(new long[]{500, 500}, 0, 1, true);
pScene.attachChild(sprite);
```

Fonte: Do Autor, 2014.

- **ParallaxScrolling**

O ParallaxScrolling é uma técnica muito eficiente e com ela é possível criar um "pseudo 3D" em jogos 2D, ajudando assim a criar uma ilusão de movimento do cenário na cena, no caso do protótipo essa técnica foi usada de forma simples, sendo que apenas dois *layers* foram usados, o *layer* do fundo fica estático e o *layer* frontal (vegetação) se move da direita para a esquerda em uma velocidade constante. Essa técnica é bem simples de ser implementada com a AndEngine apesar de ser necessário escrever bastante código (Caixa de Texto 7).

Caixa de Texto 7 – Implementação da técnica ParralaxScrolling

```
public BitmapTextureAtlas mBackgroundTextureAtlas;
public ITextureRegion mBackgroundTextureRegion;
public BitmapTextureAtlas mVegetationTextureAtlas;
public ITextureRegion mVegetationTextureRegion;


mBackgroundTextureAtlas = new BitmapTextureAtlas(this.getTextureMan-
ager(), 1024, 1024);
mBackgroundTextureRegion = BitmapTextureAtlasTextureRegionFactory.cre-
ateFromAsset(mBackgroundTextureAtlas, this, "seabed.png", 0, 0);
mBackgroundTextureAtlas.load();

mVegetationTextureAtlas = new BitmapTextureAtlas(this.getTextureManager(),
1024, 512);
mVegetationTextureRegion = BitmapTextureAtlasTextureRegionFactory.create-
FromAsset(mVegetationTextureAtlas, this, "vegetation.png", 0, 0);
mVegetationTextureAtlas.load();


final AutoParallaxBackground autoParallax = new AutoParallaxBackground(0,
0, 0, 10);
                autoParallax.attachParallaxEntity(new ParallaxEntity(0.0f, new
Sprite(0, HEIGHT - mBackgroundTextureRegion.getHeight(), mBackgroundTex-
tureRegion, getVertexBufferObjectManager())));
                autoParallax.attachParallaxEntity(new ParallaxEntity(-3f, new
Sprite(0, HEIGHT - mVegetationTextureRegion.getHeight(),
mVegetationTextureRegion, getVertexBufferObjectManager())));
```

Fonte: Do Autor, 2014.

- **Sons e Música**

Os sons nos jogos servem para aumentar a imersão do jogador no jogo, ou seja, com os sons é possível aumentar a sensação de realidade no jogo, como sons de explosão, ruídos, tiros, etc. Ao unir os sons com efeitos visuais o jogo ganha e muito em qualidade.

A música serve para criar o “clima” do jogo, portanto, músicas podem aumentar o suspense em jogos de terror ou aumentar a adrenalina em jogos de ação por exemplo.

Dependendo dos efeitos sonoros e da trilha musical o jogo pode ser um sucesso ou um fracasso. É claro que em jogos de menos impacto muitas vezes os produtores não se preocupam tanto com trilha musical, mas os efeitos sonoros são sempre um item a não ser deixado de lado.

Para adicionar um efeito sonoro no jogo com a AndEngine basta criar um objeto Sound e no onCreateResources instanciar o objeto, depois basta usar os métodos do objeto Sound para reproduzir o som, pará-lo, etc.

Para adicionar uma música basta seguir os mesmos passos feitos para o efeito sonoro, porém, é necessário criar um objeto Music e não um objeto Sound.

2.13.4 Resultado Final

O resultado final do protótipo foi considerado razoável, por ser um protótipo itens como pontuação e outros níveis de dificuldades não foram desenvolvidos, o protótipo foi desenvolvido focando o potencial da *engine* em coisas como simulação de física, detecção de colisões e efeitos sonoros.

Devido à falta de documentação e tempo não foi possível adicionar novos recursos ao protótipo (Figura 16, 17 e 18).

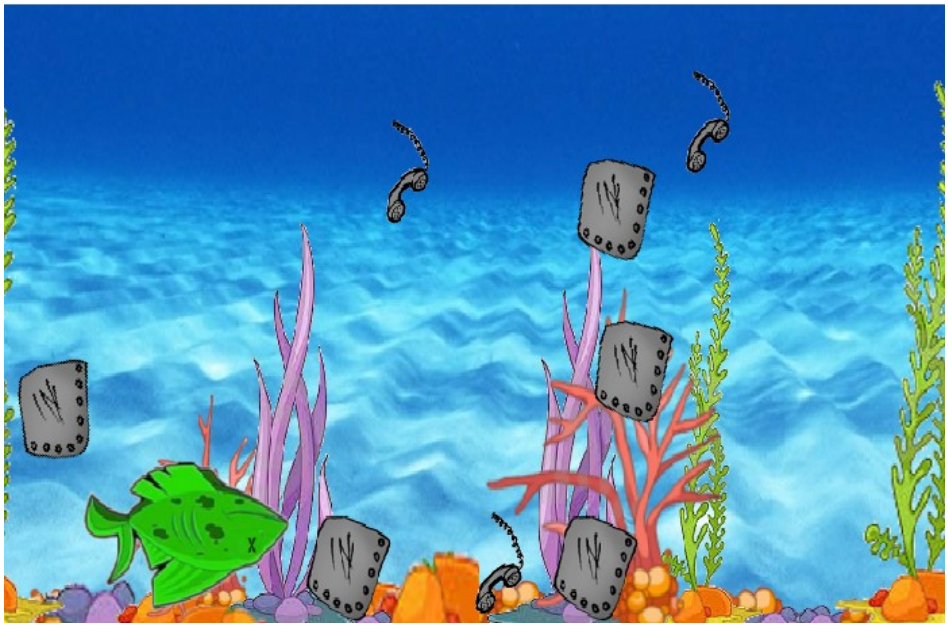
Figura 16 – Cena inicial do protótipo

Fonte: Do Autor, 2014.

Figura 17 – Cena de demonstração do protótipo

Fonte: Do Autor, 2014.

Figura 18 – Cena de demonstração do protótipo 2



Fonte: Do Autor, 2014.

3 CONSIDERAÇÕES FINAIS

No desenvolvimento de jogos eletrônicos as *engines* têm um papel central, a escolha da *engine* correta para o projeto é de fundamental importância, essa escolha tem relação direta com o resultado final do jogo e com os gastos no decorrer do desenvolvimento. Hoje em dia é praticamente impossível desenvolver um jogo de qualidade em tempo hábil sem a utilização de uma *engine* e de seus recursos.

Como o projeto tinha por finalidade utilizar-se de uma *engine* grátis e *open-source* a escolha feita foi a AndEngine, essa *engine* por sua vez demostrou ter um grande e variado número de funções e extensões para o desenvolvimento de jogos para o sistema operacional Android, porém ela não foi desenvolvida por uma equipe, foi desenvolvida por uma pessoa e disponibilizada para que a comunidade pudesse desenvolve-la juntamente com seu criador, isso fez com que a *engine* tenha pouca documentação, além disso, a pouca documentação que existe não está centralizada, tornando assim, difícil o acesso a determinadas informações.

Durante o processo de desenvolvimento do protótipo ocorreram diversos problemas causados pela falta de documentação o que acabou resultando em um atraso no desenvolvimento. Por esta razão é pouco recomendável que a AndEngine venha a ser utilizada como *engine* de jogos profissionais, porém, é uma boa escolha para estudos.

Por fim, devido aos esforços de empresas e de desenvolvedores autônomos que trabalham no desenvolvimento de novas *engines* e tecnologias de desenvolvimento de jogos hoje é possível desenvolver um jogo de qualidade e competitivo sem ter uma grande equipe, até mesmo uma única pessoa é capaz de desenvolver jogos competitivos e de qualidade. Além disso o desenvolvimento de jogos eletrônicos é uma boa fonte de estudo pois engloba vários desafios no quesito de programação e conhecimento.

4 REFERÊNCIAS

ANDROID. *Application Fundamentals*. Disponível em: <http://developer.android.com/guide/components/fundamentals.html> Acesso em: 08 jun. 2013.

ANDROID já domina 75% do mercado de smartphones. Revista Veja. Disponível em: <http://veja.abril.com.br/noticia/vida-digital/android-ja-domina-75-do-mercado-de-smartphones> Acesso em: 12 mai. 2013.

CÂMARA, Marlon. O que é Android Market e como comprar aplicativos nele. 2012. Disponível em: <http://www.techtudo.com.br/dicas-e-tutoriais/noticia/2012/02/o-que-e-android-market-e-como-comprar-aplicativos-nele.html> Acesso em: 09 jun. 2013.

CLUA, E.W.G.; BITTENCOURT J. R. Capítulo 3: Desenvolvimento de Jogos 3D: Concepção, Design e Programação. p. 1313 – 1357. **XXV Congresso da Sociedade Brasileira de Computação**, 2055. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/jai/2005/003.pdf> Acesso em: 22 mai. 2013.

GOOGLE. ***Google I/O 2013***. Disponível em: <https://developers.google.com/events/io/> Acesso em: 09 jun. 2013.

JAVA. O que é a tecnologia Java e por que é necessária? Disponível em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml> Acesso em: 09 jun. 2013.

KLEINA, Nilton. O que é engine ou motor gráfico? 2011. Disponível em: <http://www.tecmundo.com.br/video-game/9263-o-que-e-engine-ou-motor-grafico-.htm> Acesso em: 12 mai. 2013.

LACHETA, Ricardo R.. **Google Android – Aprenda a Criar Aplicações Para Dispositivos Móveis Com o Android Sdk**. 3. ed., 2013.

LOPES, Gilliard. **Jogos eletrônicos**: conceitos gerais. 2006. Disponível em: <http://www-usr.inf.ufsm.br/~pozzer/disciplinas/cga_8_classificacao_jogos.pdf> Acesso em: 21 mai. 2013.

NEWZOO. **Infographic 2012 – Brazil**. Disponível em: <http://www.newzoo.com/infographics/infographic-2012-brazil/>. Acesso em: 12 mai. 2013.

PERUCIA, Alexandre S.; BERTHÊM, Antônio Córdova de B.; BERTSHINGER, Guilherme Lage B.; MENEZES, Roberto Ribeiro C.. (2005). **Desenvolvimento de Jogos Eletrônicos**. São Paulo: NOVATEC.

SANTOS JÚNIOR, João B. dos. Linguagem de Programação Java. 2006. Disponível em: <http://www.tvdi.inf.br/upload/artigos/apostilalinguagemjava.pdf> Acesso em: 09 jun. 2013.

SCHROEDER, Jayme e BROYLES Brian. **AndEngine for Android Game Development Cookbook**. Packt Publishing Ltd, 2013.

TERRA. **Retrospace: A História dos Videogames em 40 Capítulos**. Provedor Terra, Outerspace, 2005. Disponível em:

<http://outerspace.terra.com.br/retrospace/materias/consoles/historiadosconsoles1.htm> Acesso em: 05 jun. 2014.

ZANARDO, Gustavo R. Técnicas de Detecção de Colisão para Jogos. Disponível em: <http://www.vsoftgames.com/site/files/tecnicas_deteccao_colisao_para_jogos.pdf> Acesso em 18 mai. 2013.

DARWIN, Ian F.. **Android Cookbook – Primeira Edição**. São Paulo : NOVATEC, 2012.