UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE SISTEMAS DE INFORMAÇÃO

UFOQX – DESENVOLVIMENTO DE UM JOGO 2D PARA PLATAFORMA ANDROID

Emerson Fernandes de Freitas

Orientador: Prof. Dr. Alberto Sampaio Lima

Quixadá

2014

EMERSON FERNANDES DE FREITAS

UFOQX – DESENVOLVIMENTO DE UM JOGO 2D PARA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: Computação

Orientador: Prof. Dr. Alberto Sampaio Lima

Quixadá

2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

F936u Freitas, Emerson Fernandes de Freitas
UFOQX: desenvolvimento de um jogo 2D para plataforma Android / Emerson Fernandes de Freitas. – 2014.
50 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2014.
Orientação: Prof. Dr. Alberto Sampaio Lima
Área de concentração: Computação

1. Android (Programa de computador) 2. Jogos por computador 3. Linguagem de programação (Computadores) I. Título.

CDD 005.3

EMERSON FERNANDES DE FREITAS

UFOQX – DESENVOLVIMENTO DE UM JOGO 2D PARA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: Computação

Aprovado em: ___ / ___ / 2014.

BANCA EXAMINADORA

_____

Prof. Dr. Alberto Sampaio Lima

Universidade Federal do Ceará – UFC

_____

Prof Dr. João Ferreira de Lavor

Universidade Federal do Ceará – UFC

_____

Prof. Ms. Francisco Erivelton Fernandes Aragão

Universidade Federal do Ceará – UFC

RESUMO

O desenvolvimento de jogos é um assunto que está atraindo o interesse de muitos desenvolvedores. Os recursos computacionais, kits de desenvolvimento, documentação e diversos dispositivos clientes estão cada vez mais acessíveis. Entretanto, muitas vezes não se tem certeza por onde começar. Este trabalho teve como intuito apresentar e documentar o desenvolvimento de um jogo, trazendo consigo problemas e possíveis soluções para os desafios encontrados durante o processo de desenvolvimento. Como resultado é apresentado o itinerário de desenvolvimento nas etapas próprias da engenharia de software adaptadas ao desenvolvimento de jogos utilizando o motor de jogo Cocos2D-X.

PALAVRAS-CHAVES: Desenvolvimento de jogos. Android. Cocos2D-X.

ABSTRACT

Game development is a matter that is attracting the interest of many developers. Computing resources, development kits, documentation and various client devices are becoming more affordable. However, sometimes you're not sure where to start. This work has the objective to document the development of a game, bringing problems and possible solutions to the challenges encountered during the development process. As a result the route of stages of development in their own software engineering tailored to game development using the game engine Cocos2D-X is presented.

KEYWORDS: Game development. Android. Cocos2D-X.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

As novas tecnologias de computação móvel em celulares e *tablets* tem permitido que sistemas móveis executem aplicações que até então só eram possíveis em *desktops.* Entre estes aplicativos temos jogos, pacotes de escritório, pintura e realidade aumentada.

Os principais sistemas operacionais para dispositivos móveis no mercado são o *Windows Phone* da Microsoft, o iOS da *Apple* e o *Android* da Google. Atualmente, o *Android* é o líder do mercado com 83,6% dos aparelhos em circulação segundo pesquisa realizada pela consultora Strategy Analytics (2014).

Somente em 2012, de acordo com a pesquisa divulgada pela Deway (2013), foram gastos no Brasil, aproximadamente U$ 300 milhões em jogos para dispositivos móveis. Devido a este aquecimento do setor de jogos eletrônicos, desenvolvedores voltaram mais sua atenção para esse mercado.

Devido à variedade de modelos e as diferentes características de *hardware* e de sistemas operacionais, o uso de um *game engine* (motor de jogo) vem a ser recomendado. De acordo com Ward (2008), um motor de jogo existe para abstrair os detalhes das tarefas comuns relacionada ao jogo. Exemplos de abstrações são a renderização, a física e a entrada de dados. O objetivo é que os envolvidos no desenvolvimento possam focar nos detalhes que fazem seus jogos únicos.

Em seu trabalho de conclusão de curso, Davila (2012) demonstra as características do motor de jogo *Corona SDK* no desenvolvimento multiplataforma, mas sem desenvolver um jogo. Há também o trabalho realizado por Korjenioski (2011), que descreve o desenvolvimento de um jogo para *Android*, focando no processo de desenvolvimento e não no motor de jogo utilizado.

Considerando o crescimento do mercado de jogos e as novas possibilidades geradas, este trabalho tem como objetivo demonstrar o desenvolvimento de um jogo 2D para a plataforma *Android* utilizando o motor de jogo Cocos2D-X, escolhido após comparativo descrito na seção 2.3.1. Foi seguido o processo de desenvolvimento de jogos descrito na seção 2.1. O jogo se enquadra na categoria *arcade* e a ambientação objetiva alcançar principalmente o público adolescente.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, são apresentados os conceitos principais necessários ao desenvolvimento do tema. Dado que será desenvolvido um jogo, a seção 2.1 aborda o processo de desenvolvimento de jogos. Sendo o jogo para plataforma *Android*, a seção 2.2 apresenta o sistema operacional e a importância do conhecimento da arquitetura para o desenvolvedor. A seção 2.3 explica o porquê da utilização de um motor de jogo e as características do motor de jogo Cocos2D-X, que será utilizado neste trabalho.

Jogos digitais são aplicações que possuem, na maioria das vezes, um processo de desenvolvimento específico. São aplicações compostas por enredo, imagens, sons e desafios. É um tipo atrativo de mídia, principalmente pela imersão e interatividade que proporciona aos usuários (Perúcia, 2005).

2.1 Processo de Desenvolvimento de Jogos

É possível criar um jogo utilizando as fases típicas da engenharia de *software*, mas jogos eletrônicos possuem processos de desenvolvimento específicos.

De acordo com Perúcia (2005), o processo de desenvolvimento de jogos abrange as seguintes etapas:

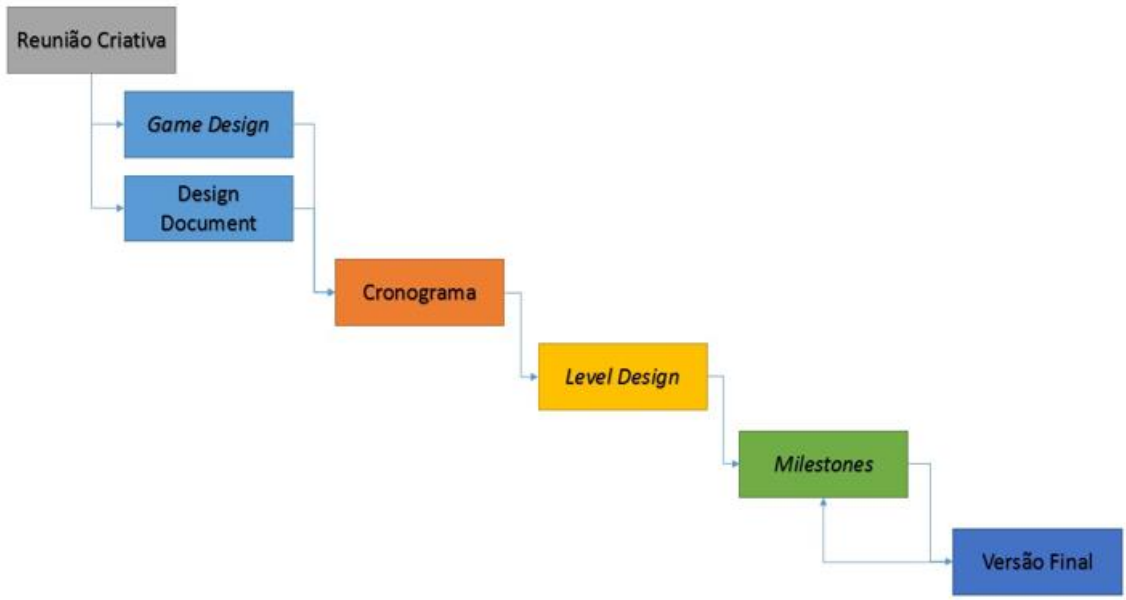
- **Reunião Criativa:** reunião onde são expostas e discutidas as ideias de jogos. Cada ideia é discutida abrangendo originalidade, inovação, público-alvo, plataforma e possibilidades de mercado.
- ***Game Design*:** processo onde são descritas as características do jogo como jogabilidade, controles, *interfaces*, personagens, armas, golpes, inimigos, fases e todos os aspectos gerais do projeto.
- ***Design Document*:** nessa fase os artistas começam a trabalhar na arte conceitual. Também nessa fase os programadores fazem a modelagem do software e implementam a estrutura básica de classes do produto.
- **Definição de Metas e Cronograma:** com o *design document* pronto, o gerente de projeto já tem noção do tamanho do produto e do tempo necessário para conclui-lo. As equipes têm suas metas definidas e o

cronograma é passado para o gerente de projeto, que organizará o cronograma geral com as metas específicas para cada equipe.

- ***Level Design*:** é um mapa geral com os desafios e missões que o jogador deve cumprir para vencer a fase. Serve como referências para que os artistas trabalhem na criação dos cenários e do visual de cada fase.
- ***Milestones*:** cada empresa define metas para versões intermediárias do produto. São geradas a cada semana ou mês e permitem que o gerente de projeto acompanhe a evolução do produto. Essas versões intermediárias possibilitam a detecção de *bugs* durante o desenvolvimento. Estes *bugs* são corrigidos para a próxima versão, aumentando a estabilidade do produto até a versão final.

A Figura 1 ilustra o ciclo do processo de desenvolvimento de jogos.

Figura 1 - Ciclo do Processo de Desenvolvimento de Jogos



Fonte 1 - Perúcia (2015)

Em projetos de pequeno porte não há necessidade de seguir todo o ciclo de desenvolvimento, pois somente o *Game Design Document* (GDD), documento gerado durante as etapas de *Game Design* e *Design Document* é o bastante para que o projeto seja executado de forma satisfatória (Perúcia, 2005).

A maioria das metodologias para desenvolvimento de jogos surgiu de adaptações de metodologias genéricas de desenvolvimento de *software*, como *Extreme Programming* e o *Rational Unified Process* (RUP).

O desenvolvimento do jogo apresentado neste trabalho seguiu a metodologia *Scrum*, na sua variante *Scrum Solo* (Seção 2.1.3). O *Scrum Solo* foi escolhido porque o trabalho será desenvolvido por uma pessoa apenas e não por uma equipe grande. Outro motivo é a priorização de que cada *sprint,* unidade básica de desenvolvimento*,* seja entregue no tempo esperado com alto valor agregado. Pequenas partes podem ser desenvolvidas e testadas, garantindo não só o funcionamento, mas que todos os responsáveis pela construção do *software*, cliente e desenvolvedor estão em sintonia de ideias.

A seção sobre metodologia de desenvolvimento (Seção 3), detalha mais o processo que foi seguido.

2.1.1 *Game Waterfall Process*

O processo cascata para jogos é uma versão adaptada do tradicional processo de desenvolvimento de sistemas denominado cascata. Ele foi batizado com este nome em alusão a uma cascata, onde o fluxo de água tem um único sentido. Assim, analogamente, no modelo cascata todas as atividades envolvidas com a produção de um sistema ocorrem de forma sequencial. Existe uma relação direta entre as etapas do modo tradicional e do modo voltado à jogos. A Figura 2 apresenta esta relação:

Figura 2 - *Game Waterfall Process*

Fonte 2 - Centro de Informática UFPE

2.1.2 *Extreme Game Development*

O *Extreme Game Development* (XGD) é uma metodologia ágil de desenvolvimento de jogos, baseado no *Extreme Programming* (XP). Basicamente, todos os princípios e a maioria das práticas do XP se aplicam também ao XGD. Mas, como o XP foi criado por programadores, algumas práticas precisam ser adaptadas para os outros papéis que compõem uma equipe de desenvolvimento de jogos (artistas, engenheiros de som, etc.). A principal motivação que levou ao surgimento do XGD foram os constantes atrasos presentes no desenvolvimento de jogos, em conjunto com as altas penalidades impostas pelas publicadoras sobre os atrasos. Isto faz com que a equipe de desenvolvimento trabalhe sobre grande pressão e aumenta as chances da entrega de *milestones* instáveis (Carvalho, 2006). Assim como o XP, o XGD se baseia em 5 princípios ou valores descritos na Tabela 1:

Tabela 1 - Valores do XGD

Valor	Descrição
Comunicação	A comunicação entre os integrantes da equipe é de suma importância. A Comunicação auxilia na solução de problemas e aumenta o sentimento de cooperação entre a equipe. Além disso, a comunicação direta deve ser feita em detrimento aos documentos formais.
Simplicidade	Simplicidade é talvez o principal valor do XGD. A lei do XP e do XGD é: “Faça o item que funcione, da forma mais simples possível.”
Feedback	É o ato de o cliente comunicar ao desenvolvedor algo de novo que ele aprendeu sobre o problema. É também o desenvolvedor comunicar ao cliente estimativas, riscos e melhorias do projeto. Quanto mais cedo se sabe, antes se pode adaptar (Beck, 2004).
Coragem	Coragem é ter ações efetivas para superar dificuldades. Quando combinada com outros valores, torna-se uma arma poderosa: coragem para comunicar, para manter simples e para ouvir o feedback (Beck, 2004).
Respeito	Toda a equipe de desenvolvimento precisa ter respeito com os demais. É preciso que cada um se importe com o projeto, caso

	contrário, o fracasso será certo. Assim como no XP, no *Extreme Game Development* todos são responsáveis pelo projeto (Beck, 2004).

Fonte – Autor, baseado no texto de Beck (2004).

Para executar o trabalho visando a utilização desses valores, o XP possui um conjunto de práticas. O XGD manteve algumas dessas práticas, com poucas ou nenhumas alterações em relação ao XP. A Tabela 2 apresenta algumas das práticas.

Tabela 2 - Práticas do XGD

Prática	Descrição
Time Inteiro	A equipe deve permanecer coesa em comunicação. A equipe é um todo, e não a soma de forças individuais.
Design Incremental	As tarefas (*assets*) de um jogo devem ser feitas da forma mais simples possível, que funcionem corretamente.
Estórias de Usuário	São pequenas descrições das funcionalidades do jogo, descritas normalmente pelo cliente.
Ciclo Semanal	O projeto é organizado e planejado para ser executado em ciclos de curta duração.
Integração Contínua	No projeto não existem “pedaços” de código-fonte. O projeto está continuamente integrado e funcionando.
Código Compartilhado	Todos da equipe compartilham todo o código-fonte do projeto.
Reuniões *Stand-up*	Reuniões rápidas de projeto, com objetivo de que toda a equipe esteja ciente do trabalho que está sendo feito num dado momento.

Fonte 3 - Autor, baseado no texto de Beck(2004)

As principais críticas ao XGD dizem respeito a um suporte ineficiente à documentação necessária ao desenvolvimento de jogos. Ou seja, o modelo em questão não produz naturalmente a documentação necessária, no nível de detalhes comumente exigido.

2.1.3 *Scrum*

O *Scrum* é um processo ágil, iterativo e incremental que foca na aceitação das mudanças que podem ocorrer, principalmente em contextos pouco definidos. Ele foi primeiramente descrito por Takeuchi e Nonaka em seu trabalho chamado *The New Product Development Game* (1986) e embora não seja exatamente uma metodologia de desenvolvimento específica para jogos, vem sendo usada por algumas empresas ao redor do mundo. Essa premissa de aceitação das mudanças que podem ocorrer e adaptação às mesmas vai de encontro a metodologias preditivas, como o RUP. Estas metodologias, ditas preditivas consideram que é possível prever todas as necessidades de um processo, antes do mesmo ser executado. De acordo com Schwaber (2004), na execução do *Scrum*, há um facilitador chamado de *Scrum Master*. Ele é responsável basicamente por monitorar a execução da metodologia e servir como mediador entre a equipe de desenvolvimento e o *Product Owner*. Este último é o responsável, dentro da equipe do projeto, por representar o patrocinador do projeto. Assim, ele é o responsável por priorizar e acompanhar as funcionalidades a serem desenvolvidas no projeto. A unidade básica de desenvolvimento no *Scrum* é chamado de *sprint*. *Sprints* tendem a durar entre uma semana e um mês e são restritos a uma duração específica. Além de definir papéis, o *Scrum* prevê também a confecção de alguns artefatos que auxiliam o gerenciamento do projeto, como mostrado na Tabela 3.

Tabela 3 - Artefatos do *Scrum*

Artefato	Descrição
Product Backlog	É uma lista de funcionalidades que o produto deve ter. Estas funcionalidades são priorizadas pelo *Product Owner*.
Sprint Backlog	Lista de tarefas a serem realizadas em cada iteração (*sprint*) do projeto. Essa lista é normalmente definida em negociação pelo *Product Owner* e a equipe de desenvolvimento.
Impediment List	Esta lista contém todos os empecilhos encontrados pela equipe de desenvolvimento que a impede de alcançar seus objetivos. Estes problemas devem

	ser resolvidos pelo *Scrum Master*.
Product Bakclog Burn Down	Relatório contendo as funcionalidades que já foram completadas. Permite a visualização do status do projeto como um todo.
Sprint Backlog Burn Down	Relatório contendo as funcionalidades que já foram completadas em uma *sprint* específica.

Fonte – Autor, baseado no *The New Product Development Game* (1986)

No *Scrum* um projeto é dividido nas seguintes fases, mostradas na Tabela 4.

Tabela 4 - Fases de projeto do *Scrum*

Fase	Descrição
Preparação	Nesta fase são feitas as definições iniciais do projeto como, por exemplo, o *Product Backlog.*
Sprint Planning Meeting	Primeiro encontro de uma *sprint*. O *Product Owner* prioriza as funcionalidades presentes no *Product Backlog* e define o que deve ser entregue na *sprint.*
Daily Scrum Meeting	Encontros diários que servem para que toda a equipe possa apresentar aos outros integrantes, o que ainda falta ser feito e quais os problemas encontrados enquanto a atividade estava sendo realizada.
Product Increment	O conjunto de funcionalidades que devem ser entregues em uma *sprint*.
Sprint Review	Momento em que o *Product Owner* valida o *Product Increment* entregue pela equipe.
Sprint Retrospective	Encontro que busca levantar, entre todos da equipe, os pontos positivos e negativos observados durante a ultima *sprint* realizada.
Update Product Backlog	O *Product Owner* é responsável por dar uma nova classificação quanto a prioridade, para as funcionalidades a serem entregues na próxima *sprint*.
Encerramento	Realizada após todos as *sprints* do projeto. É nesta fase que é feita a entrega do produto do projeto realizado.

Fonte - Autor, baseado no The New Product Development Game (1986)

Entretanto, o *Scrum* nessa forma demonstrada é de difícil aplicação quando existe apenas uma pessoa no processo de desenvolvimento. Nesse caso, o *Scrum Solo* é a solução encontrada. Nesta modalidade, os diversos papéis são descartados, mas os princípios e os artefatos do *Scrum* são respeitados, seguindo-se, então, as boas práticas do *Scrum*.

2.2 Android

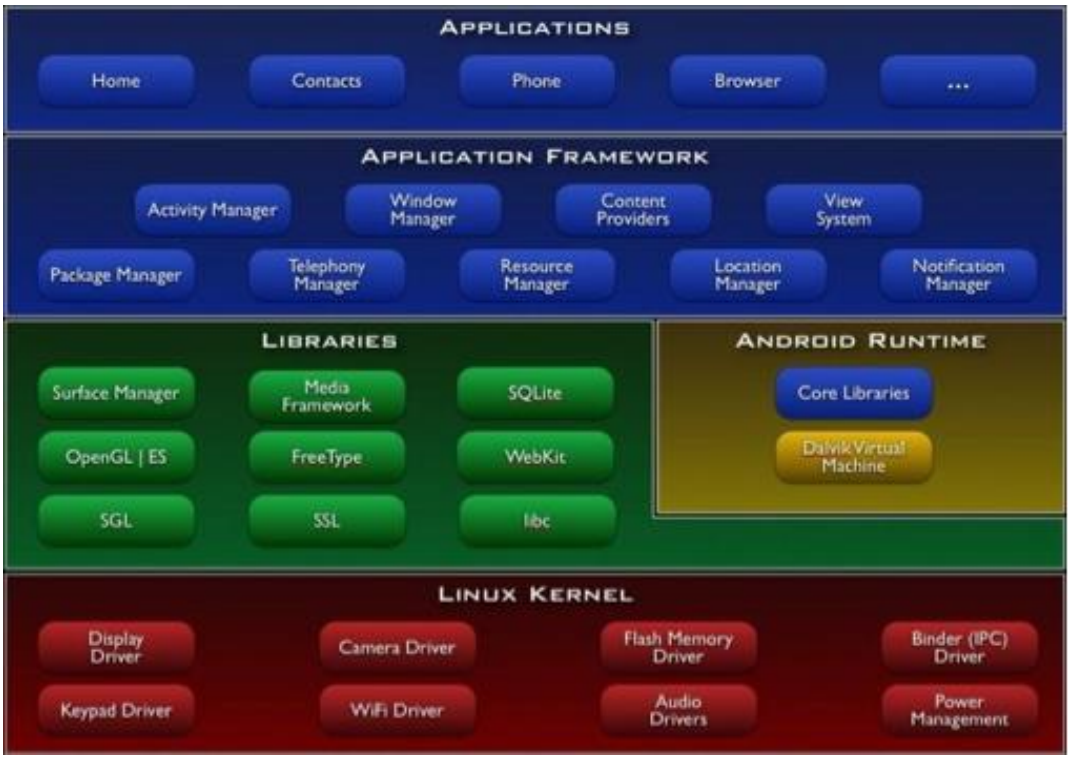
Android é um sistema operacional *open source* voltado para dispositivos móveis, foi desenvolvido pela *Open Handset Alliance* (OHA) e mantido pela *Google*. O sistema é baseado numa versão modificada do Linux.

De acordo com Lee (2011), a principal vantagem da adoção do *Android* é que ele oferece uma abordagem unificada para o desenvolvimento de aplicativos. Os desenvolvedores só precisam desenvolver para o *Android*, e seus aplicativos devem ser capazes de rodar em numerosos dispositivos, desde que estes executem o sistema *Android*.

Segundo a pesquisa realizada pela empresa de análise de mercado Gartner e publicada pela INFO Online (2013), as vendas de *smartphones* com *Android* cresceram 79,5% durante o primeiro quadrimestre de 2013 em relação ao mesmo período no ano passado. Essa percentagem refere-se a uma quantidade de 162,1 milhões de aparelhos. Estima-se que esse número continue a crescer.

Lee (2011) também fala que para entender como o *Android* funciona, é preciso conhecer sua arquitetura (Figura 3).

Figura 3 - Arquitetura do *Android*



Fonte 4- Google Inc (2011)

- ***Linux Kernel:*** núcleo em que o *Android* é baseado. Esta camada contém todos os drives de baixo nível de dispositivos para os vários componentes de hardware;
- ***Libraries:*** contém todo o código que provê as principais funcionalidades do sistema operacional. Por exemplo, a biblioteca *WebKit* provê funcionalidades para navegação *web*;
- ***Android Runtime:*** provê uma série de bibliotecas centrais que permitem aos desenvolvedores escrevem aplicativos usando a linguagem Java. Também inclui a máquina virtual *Dalvik* que permite que cada aplicativo rode em seu próprio processo;
- ***Application Framework:*** expõe as capacidades do sistema operacional *Android* para os desenvolvedores, para que esses possam fazer uso delas em seus aplicativos;
- **Aplicativos:** essa camada inclui os aplicativos nativos do dispositivo, assim como os aplicativos que foram instalados.

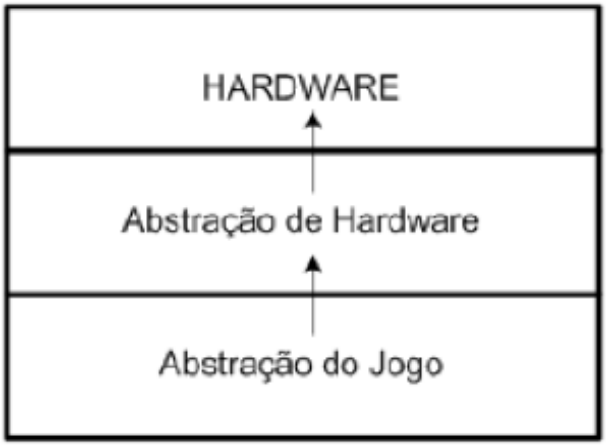
A compreensão da modularização do *Android* ajuda no processo de decisão das características do jogo. Tendo conhecimento da arquitetura, o desenvolvedor

saberá que recursos o sistema já lhe provê e poderá utilizá-los quando necessário. Dessa forma, o desenvolvedor poderá focar nos detalhes que fazem do seu jogo único.

2.3 *Game Engine*

Um *game engine* (motor de jogo) para desenvolvimento de jogos pode ser tanto uma IDE (*Integrated Development Environment*) quanto uma biblioteca de funcionalidades que são disponibilizadas para que o desenvolvedor não tenha que criar um jogo do zero. Um motor de jogo atua na camada de abstração da arquitetura do jogo, conforme mostrado na Figura 4.

Figura 4 - Arquitetura de um jogo em camadas



Fonte 5 - Centro de Informática UFPE

O propósito da camada de abstração de *hardware* consiste em facilitar o uso de recursos gráficos, de entrada/saída, de som, de rede, dentre outros.

O motor de jogo é responsável por oferecer essas abstrações. Dentro de um motor de jogo existem vários subsistemas, sendo os principais: motor gráfico, motor de física, motor de som e motor de inteligência artificial (Perúcia, 2005).

- **Motor Gráfico:** responsável pela renderização das imagens e resposta visual interativa;
- **Motor de Física:** simula a física real como gravidade, elasticidade, fricção, bem como calcula movimento de colisão entre corpos;
- **Motor de Som:** encarregado da produção dos sons e efeitos sonoros;
- **Motor de Inteligência Artificial:** responsável por definir o comportamento dos personagens não jogáveis.

2.3.1 *Game Engines* Disponíveis

Existem diversos motores de jogos para desenvolvimento de jogos para *Android* no mercado. A Tabela 5 mostra um comparativo que considera as características como linguagem suportada, IDE própria, documentação e licença entre as mais utilizadas.

Tabela 4 - Comparativo entre *Game Engines* para *Android*

Game Engine	Linguagem Suportada	IDE	Licença
AndEngine	Java	Não possui	Livre
Corona SDK	Lua	Não possui	Gratuito para teste
cocos2d-android	Java	Não possui	Livre
Unity 3D	C#, JS e Boo	Possui	Gratuito com limitações
Moai SDK	Lua e C++	Possui	Livre
FlatRedBall	C#	Possui	Livre
Cocos2D-X	C++, Lua e JavaScript	Não possui	Livre

Fonte – Autor.

Considerando a possibilidade de livre uso e linguagem suportadas, foram avaliadas para utilização na presente pesquisa os motores de jogo Unity3D, Cocos2D-X e AndEngine. O motor escolhido para desenvolver esse trabalhos foi o Cocos2D-X por sua capacidade para desenvolvimento multiplataforma, licença livre e comunidade bastante ativa, o que viabilizará trabalhos futuros. As principais características do Cocos2D-X são descritas na seção 2.3.2.

2.3.2 Cocos2D-X

O Cocos2D-x é um motor de jogo de código aberto, para desenvolvimento de jogos 2D. Ele pode ser usado para construir jogos, aplicativos e outros programas. Permite também, que os desenvolvedores explorem seu conhecimento em C++, Lua e Javascript no desenvolvimento de jogos multiplataforma.

Entre as plataformas suportadas pelo Cocos2D-x temos: *Android*, *iOS*, *Windows Phone 8*, *MacOS X* e *Windows 7*.

As principais classes do Cocos2D-X utilizadas no desenvolvimento do jogo nesta pesquisa, estão relacionadas na tabela 6.

Tabela 5 - Principais Classes do Coco2D-X

Classe	Descrição
CCScene	A classe CCScene é usada para criar telas como as de Menu, de Jogo e de Créditos. A Classe CCScene pode ser usada como contêiner de CCLayer. É uma entidade abstrata e você pode aplicar transições nela para ir de uma tela para outra.
CCLayer	Em uma *scene* (cena), você pode ter diferentes *layers* (camadas) para ajudar a organizar melhor sua cena como acontece em programas de edição de imagem. Em jogos, você poderá ter diferentes camadas como a HUD (*Heads-up Display*) e *background*. Também, CCLayers, diferentemente de CCScenes, possuem a habilidade de receber eventos de toque. Dentro de CCLayers, você pode ter CCSprites, CCMenus, e assim por diante.
CCSprite	É a classe usada para adicionar imagens na tela. A imagem pode estar no formato .png ou .jpg. Ela tem varias propriedades como *width* (largura) e *height* (altura) da imagem, setPosition, setScale, e setRotation, que podem ser usadas para manipular a posição, escala e rotação da imagem.
CCLabelTTF	É a classe usada sempre que seja necessário mostrar qualquer texto na tela. É usada principalmente para mostrar tutoriais, número de níveis no início do jogo. O texto pode ser mudado dinamicamente, por exemplo, atualizando a pontuação ou tempo durante a partida.
CCMenu	É usada para colocar elemento de interface de usuário (UI) como botões. CCMenuItems são usadas para anexar imagens, posição, e então adicioná-las em CCMenu para que sejam mostradas na tela.

Fonte – Autor.

Mais detalhes sobre a utilização dessas classes serão mostrados na seção sobre o desenvolvimento do jogo.

3 METODOLOGIA DE DESENVOLVIMENTO

Para o desenvolvimento do jogo apresentado nesta pesquisa, foi utilizado o *Scrum Solo* como metodologia de desenvolvimento. Como dito anteriormente, o projeto de desenvolvimento de um jogo possui várias etapas. As etapas utilizadas nesta pesquisa foram a de *Concepção, de Produção e de Avaliação*. Tendo em vista que não se trata de um trabalho realizado por várias pessoas e com um tempo limitado, as etapas foram resumidas nas seguintes partes, cada uma correspondendo inicialmente a uma *sprint* de 15 dias, podendo haver a necessidade de mais uma *sprint* em cada etapa.

Concepção:

- Game Design Document;
- Sprites;

Produção:

- Tela de jogo;
- Tela de game over;
- Menu principal;
- Menu de opções;
- Efeitos sonoros;

Avaliação:

- Confecção do questionário;
- Interação de alguns usuários com o jogo;
- Aplicação do questionário;
- Consolidação dos dados coletados;

A primeira etapa foi a de *Concepção*. Nessa etapa, foram produzidos o GDD, que contém todas as informações do jogo como título, gênero de jogo, enredo, mecânica de jogo, telas e áudio. O GDD foi criado tendo como base o modelo disponível em *Designing Games* (2013). Nessa etapa também, foi criada toda a parte artística do jogo (cenário, personagens e etc.). Para a criação e animação dos *Sprites* foram utilizados os softwares Photoshop (2014) e TextureParcker (2014).

A segunda etapa consistiu na Produção. Essa etapa consiste na produção do jogo em si. Nessa etapa, foram criados o código-fonte do jogo, animações dos elementos na tela e efeitos de som. Para a codificação, foram utilizados a IDE (*Integrated Development Environment*) Eclipse (2013) juntamente com *Android* SDK (*Software Development* Kit) (2013) e o motor de jogo Cocos2D-X (2013). Para os efeitos de som, foram utilizados efeitos prontos de um repositório gratuito, o *The Sound Resource* (2014).

Na terceira etapa ocorreu a avaliação do resultado do trabalho. Foi elaborado um questionário de avaliação, onde a avaliação foi realizada com a interação de alguns usuários que representem o público-alvo com o jogo. Por fim, foi aplicado o questionário aos usuário e os dados coletados foram consolidados para que pudesse ser verificada a qualidade e a satisfação do usuário em relação ao jogo.

4 RESULTADOS OBTIDOS

Nesta seção serão listados os resultados obtidos na realização desse trabalho.

4.1 GDD

Tabela 6 - GDD

Título	**UFOQX**
Descrição Simples	O jogo consiste em impedir que os inimigos cheguem até o chão do cenário.
Tipo/Gênero de Jogo	Arcade
Cenário	O jogo se passa na cidade de Quixadá, localizada no interior do estado de Ceará. A cidade é conhecida pelas famosas formações rochosas, os monólitos e é local de supostas aparições de UFOs (*Unidentified Flying Objects* ou Objetos Voadores não identificados).
Descrição Longa	O jogo é sobre uma possível invasão alienígena na cidade de Quixadá. Inúmeros UFOs estão atacando a cidade. Ao jogador cabe utilizar de uma arma especialmente

	desenvolvida, destruindo os UFOs antes que estes cheguem ao solo e causem danos a cidade. Para disparar um tiro o jogador deve tocar no local onde deseja mirar e tocar novamente para disparar. O disparo só será realizado caso a arma tenha carregado um pouco de energia, caso contrário nada acontecerá. Além dos UFOs, caixas de primeiros socorros cairão aleatoriamente. Diferentemente dos UFOs essas caixas devem tocar no solo para que seja restaurado um pouco de vida. A câmera permanece sem movimento durante toda a partida.
Sistema de jogo	O sistema de jogo comporta os seguintes elementos: • UFOs: surgem de qualquer ponto de cima da tela e avançam em direção ao solo, tendo seu movimento modificado pelos seguintes eventos; o Toque no solo: ativa a animação de explosão e diminui em 15% a vida do jogador; o Toque na onda de choque: ativa a animação de explosão e elimina o UFO da tela; • Caixas de primeiros socorros: surgem de qualquer ponto de cima da tela e avançam em direção ao solo, tendo seu movimento interrompido pelos seguintes eventos; o Toque no solo: ativa o efeito de aumento de vida e aumenta em 15% a vida do jogador; o Toque na onda de choque: ativa o efeito de explosão e elimina a caixa de primeiros socorros da tela; • Mira: surge quando o jogador toca em qualquer área da tela do jogo, sofrendo mudança pelo seguinte evento; o Toque na tela: ativa o efeito de onda de choque caso o tempo mínimo de carregamento esteja concluído, caso contrário ativa o efeito de falha;
Jogabilidade	A jogabilidade consiste apenas no toque na tela. A mecânica é

	simples, o jogador pode tocar em qualquer parte da tela e, a partir desse toque, um evento ocorre. Este evento pode disparar uma mudança os pontos de vida do jogador. O jogador não poderá aplicar efeitos de zoom e nem arrastar a tela para visualizar outras partes do cenário.
Telas	O jogo deve conter as seguintes telas: • Menu principal: tela que contém o título do jogo, a opção de iniciar o jogo, a opção de acessar o menu de opções e uma música de fundo; • Menu de opções: tela que contém a opção de desativar a música de fundo e de retornar ao menu principal; • Tela de jogo: tela onde o jogo acontece; • Tela de *game over*: tela mostrada quando o jogador perde todos os pontos de vida. Deve conter uma mensagem de encorajamento ou de parabéns caso a pontuação obtida seja maior que a anterior, a pontuação obtida, e opção para jogar novamente;
Requisitos de Áudio	Da parte sonora espera-se: • Possibilidade de desativar a música de fundo; • Música de fundo: Tema musical que permaneça ativo durante toda a partida; • Sons de alerta: devem existir sons de alerta, vinculados a certos eventos do jogo. o Pressionar botões do menu; o Disparo bem sucedido; o Disparo mal sucedido; o Colisão de um UFO com o solo; o Colisão de uma caixa de primeiros socorros com o solo; o Colisão de um UFO ou caixa de primeiros socorros com a onda de choque; o *Game over*;

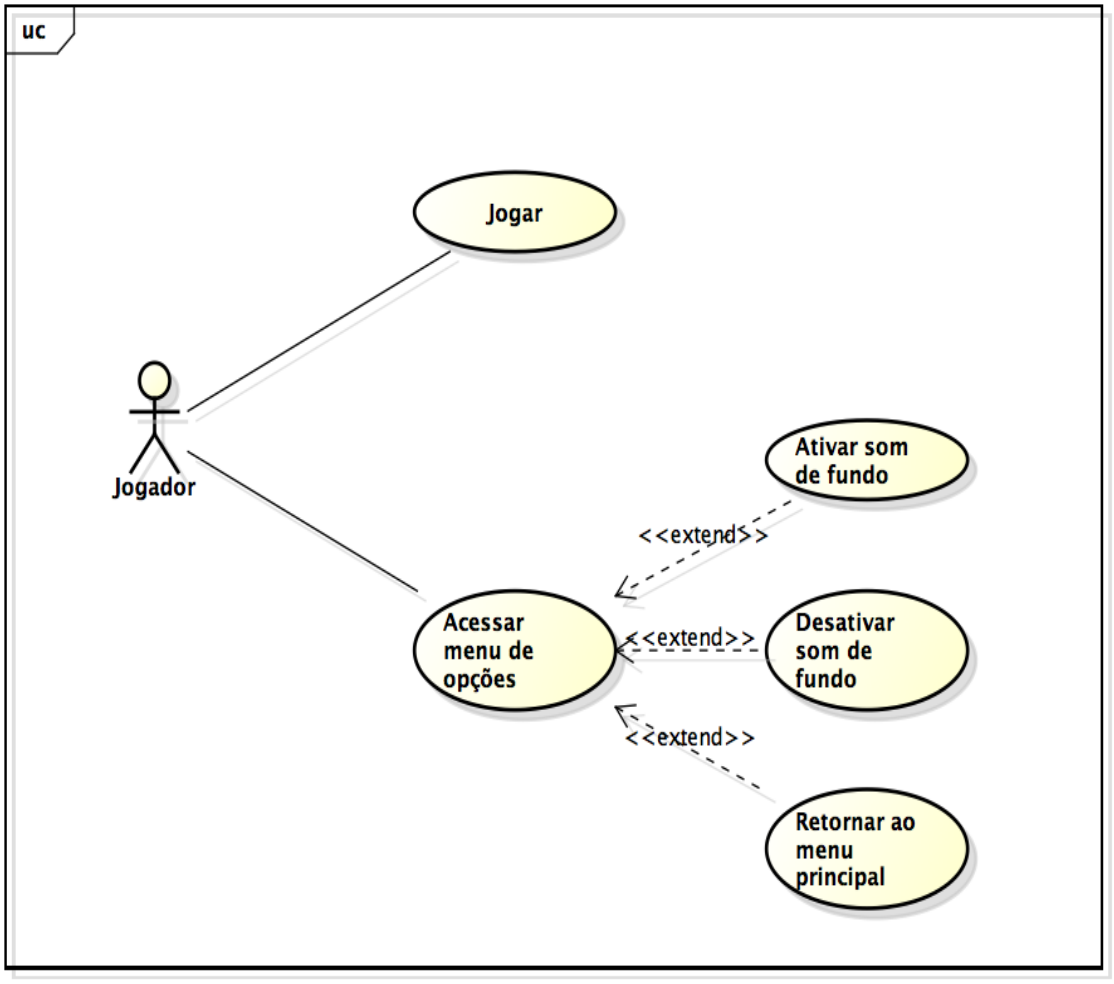
Fonte – Autor.

4.2 Modelagem Funcional

Estão presentes nessa seção os Casos de Uso (para modelagem do sistema de menus e usabilidade geral do jogo), mapa de transição (demonstrando a relação entre as telas do jogo) e *backlogs* das *sprints*.

4.2.1 Casos de uso

Figura 5 - Casos de uso do Proteja Quixadá

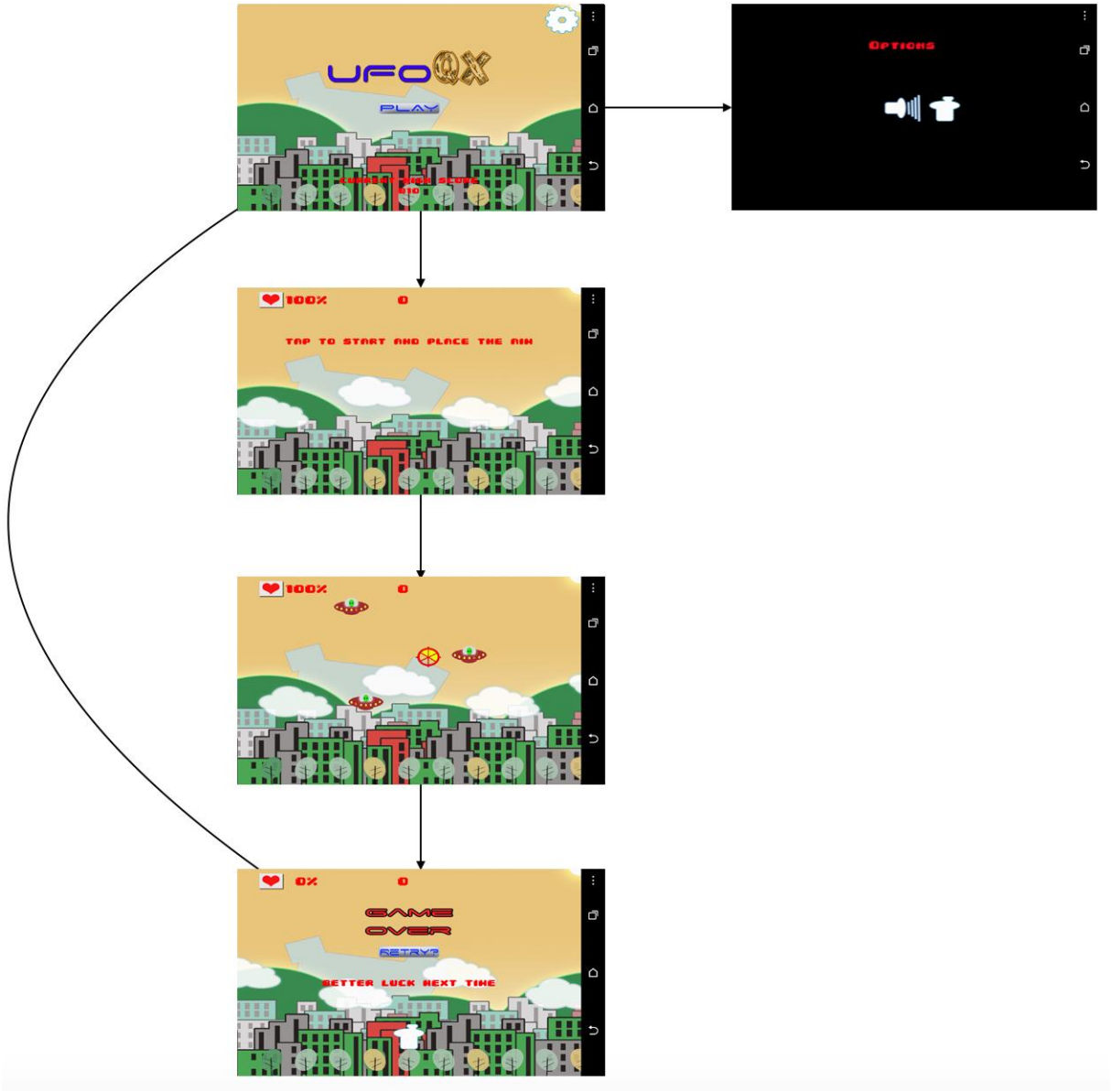


Fonte 6– Autor

4.2.2 Mapa de Transição

O mapa de transição demonstra a transição de uma tela para outra no jogo. A Figura 6 ilustra o mapa de transição deste projeto.

Figura 6 - Mapa de Transição do UFOQX



Fonte – Autor

4.2.3 *Backlogs*

Nesta seção são apresentados os *backlogs* das *sprints* realizadas ao longo do projeto.

- Sprint 01
  - Criação do GDD
    - Título;
    - Descrição simples;
    - Tipo/Gênero;

        - Cenário;
        - Descrição jogo;
        - Sistema de jogo;
        - Telas;
        - Requisitos de áudio;
    - Criação dos sprites
        - Desenho;
        - Pintura;
        - Criação do atlas de textura;
- Sprint 02
    - Criação da tela de jogo;
    - Animação dos sprites;
    - Captura e tratamento do evento de toque na tela;
    - Tratamento de colisão;
- Sprint 03
    - Criação do menu principal;
    - Criação do menu de opções;
    - Criação da tela de *game over*;
    - Adição de efeitos sonoros;
- Sprint 04
    - Criação do questionário;
    - Aplicação;
    - Consolidação dos resultados;

4.2.4 Modelagem de dados

Inicialmente foi pensado que haveria a necessidade armazenamento de dados utilizando o SQLite, padrão do *Android*. Entretanto uma outra solução foi encontrada com a classe CCUserDefault conforme a seção 6.6.

5 FUNCIONAMENTO DO SISTEMA

Com o encerramento das atividades, o funcionamento do jogo é aquele previsto no GDD. O jogo inicializa com uma tela de menu principal, onde o nome do jogo está presente. O som de fundo pode ser ativado, ou desativado, acessando o menu de opções existente no menu principal.

A tela de menu principal contém os ícones para acessar o menu de opções e para começar a jogar. Para escolher uma opção, basta tocar sobre o ícone desejado.

Durante o jogo, para pausar, basta pressionar o botão *home* do dispositivo cliente. Para retornar basta clicar no ícone do jogo novamente. Ainda não há uma opção para pausar ou sair no próprio jogo. Esse detalhe foi um dos pontos observados durante validação (seção 7) e está na lista das modificações necessárias. Existem alguns ícones na tela de *game over.* Esses ícones permitem jogar novamente ou retornar ao menu principal.

Maiores detalhes do funcionamento do jogo podem ser verificados no GDD deste mesmo documento. O arquivo executável pode ser encontrado no link a seguir:
https://drive.google.com/file/d/0B1Y3u5wz0NEZOVMzb080ZnB1N1k/view?usp=sharing

6 DESAFIOS E SOLUÇÕES

Nesta seção serão descritos os principais desafios, junto com suas soluções, encontrados durante o desenvolvimento do jogo.

6.1 Configurando o ambiente de desenvolvimento

Para começar a desenvolver com Cococs2D-X para *Android* são necessários alguns passos. Inicialmente foi necessário se instalar as seguintes ferramentas.

- Eclipse ADT (http://developer.android.com/tools/sdk/eclipse-adt.html);

- *Android* NDK (https://developer.android.com/tools/sdk/ndk/index.html);
- *Android* SDK (incluído na instalação do Eclipse ADT);
- Cocos2d-X (http://www.cocos2d-x.org/download);
- Python

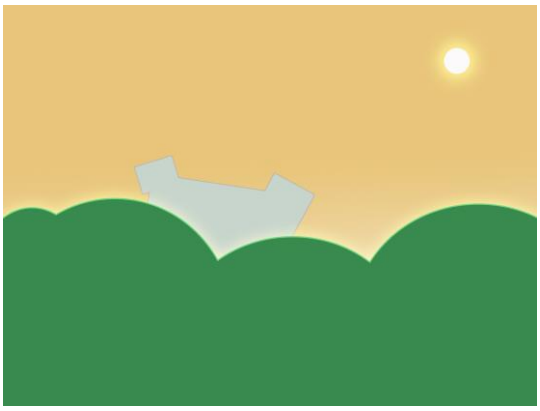
Após a instalação das ferramentas acima, é preciso realizar algumas configurações antes de iniciar o desenvolvimento. A forma de configuração muda de acordo com o sistema operacional que o desenvolvedor utiliza. Essa é uma das desvantagens do Cocos2D-X.

O trabalho foi desenvolvido em um *notebook* com o sistema operacional (SO) MacOS X e foi seguido o seguinte tutorial para instalação http://www.raywenderlich.com/33750/cocos2d-x-tutorial-for-ios-and-android-getting-started. Para desenvolvimento com SO Windows, recomenda-se o seguinte tutorial: https://software.intel.com/pt-br/articles/creating-multi-platform-games-with-cocos2d-x. Existem tutoriais para configuração em outros SOs, e nos citados acima, na página oficial do Cocos2d-X http://www.cocos2d-x.org/wiki/Getting_Started_with_Cocos2d-x.

6.2 Sprites

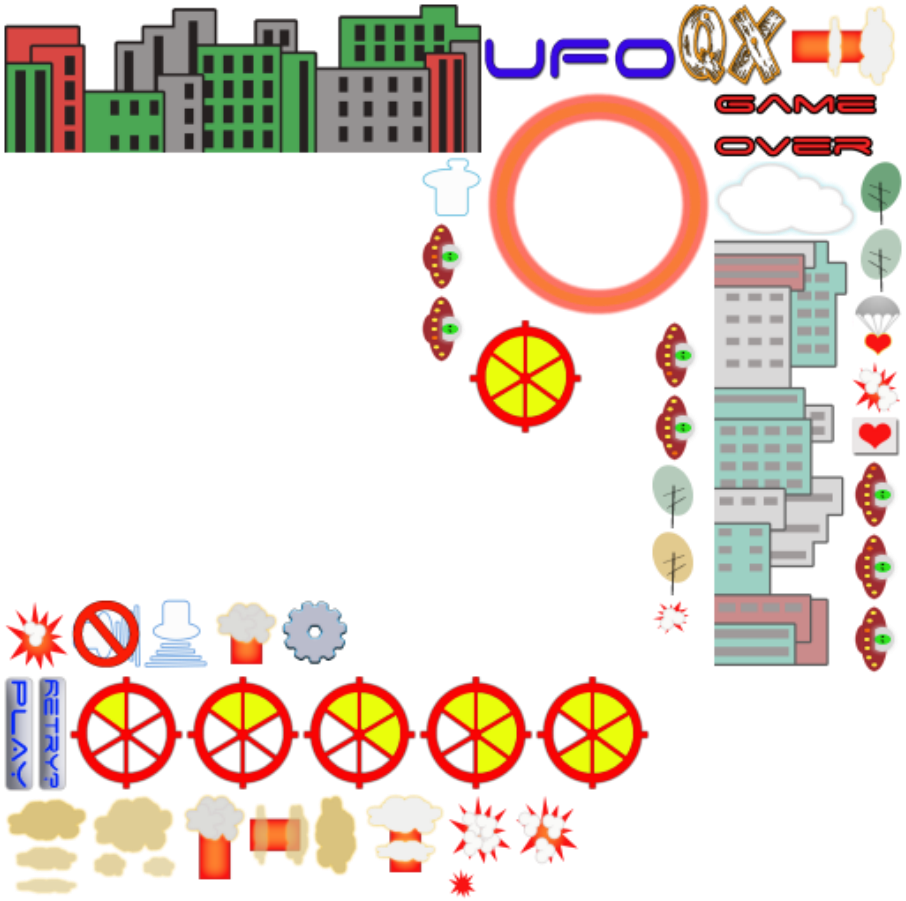
Sprites são imagens bidimensionais ou tridimensionais que se movem sem deixar traços na tela. Todos os *sprites* do jogo foram desenvolvidos com a utilização do *software* para edição de imagens Photoshop. As Figuras 7 e 8 referem-se a todas as imagens utilizadas no jogo.

Figura 7 - Imagem de fundo do Proteja Quixadá



Fonte 7 - Autor

Figura 8 - Atlas de textura do Proteja Quixadá



Fonte 8 - Autor

A Figura 8 também representa o atlas de textura do jogo. Um atlas de textura funciona como uma imagem que contém diversas outras sub-imagens menores de forma mapeada. Para criação do atlas de textura foi utilizado o *software* TexturePacker. Ao criar um mapa de textura, são gerados um arquivo de mapeamento com o formato plist e uma imagem com o formato png. Para utilizar ambos, basta copiá-los para a pasta *Resources* do projeto.

Para carregar um *sprite* e exibi-lo na tela, o Cocos2d-X utiliza a classe CCSprite. A classe CCSprite contém diversas funções para manipulação da imagem carregada. Abaixo podemos conferir a utilização da classe CCSprite para exibição da imagem de fundo do jogo.

```
CCSprite * background = CCSprite::create("background.png");
background->setPosition(ccp(tamTela.width * 0.5f, tamTela.height * 0.5f));
this->addChild(backgrond);
```

No trecho de código acima, é criada uma instância da classe CCSprite contendo a imagem de fundo, posicionada no centro da tela e adicionada a classe em questão.

Para utilizar os *sprites* contidos no atlas de textura é preciso carregar o arquivo de mapeamento e a imagem e armazenar as informações em *cache*, como mostrado no trecho de código abaixo.

```
CCSpriteFrameCache::sharedSpriteFrameCache()->addSpriteFramesWithFile(
                        "sprite_sheet.plist");
batchNode = CCSpriteBatchNode::create("sprite_sheet.png");
this->addChild(batchNode);
```

6.3 Animando *sprites*

Jogos não são feitos apenas de várias imagens estáticas sendo exibidas na tela. Para se tornar um jogo mais atraente ou divertido é preciso fazer com os *sprites* se movimentem. A animação do carregamento da arma é uma das animações criadas para o jogo. Na Figura 8, pode-se verificar 6 *sprites* que representam essa animação. Cada imagem possui um detalhe diferente, no caso uma parte mais pintada.

Para criar uma animação no Cocos2D-X, foram utilizadas as classes CCAnimation e CCAnimate. Abaixo pode-se conferir a utilização das duas classes citadas para a criação da animação da mira.

```
animacao = CCAnimation::create();
for (i = 1; I < 7; i++) {
        char string[100] = { 0 };
        sprintf(string, "aim_%i.png", i);
        frame = CCSpriteFrameCache::sharedSpriteFrameCache()->spriteFrameByName(string);
        animacao->addSpriteFrame(frame);
}

animation->setDelayPerUnit(6.0 / 6.0f);
animation->setRestoreOriginalFrame(false);
animacaoMira = CCAnimate::create(animacao);
animacaoMira->retain();
```

No trecho de código acima, é criada, com utilização de um laço, uma animação contendo as 6 imagens referentes a mira “aim_%i.png” contidas no atlas de textura. A classe CCAnimation é apenas para criação da animação. Para rodar a animação é necessário atribuir a animação criada a uma variável da classe CCAnimate. Por fim, basta definir o momento que deseja rodar sua animação e usar a função runAction da classe CCSprite.

6.4 Criando menus

Para criação de menus o Cocos2D-X possui as classes CCMenu e CCMenuItemImage. A classe CCMenuItemImage funciona apenas para definir uma imagem para um item do menu, assim como a posição onde deseja que a imagem seja colocada. Após criar todos os CCMenuItemImages que precise, basta adicioná-los a uma instância da classe CCMenu. O trecho de código abaixo exemplifica esse processo.

```
CCMenuItemImage *itemOpcoes = CCMenuItemImage::create("menu.png",
                    "menu.png", this, menu_selector(MainMenu::optionsScene));
itemOpcoes->setPosition( ccp(tamTela.width - itemOpcoes->getContentSize().width/2,
tamTela.height -  itemOpcoes->getContentSize().height/2));

CCMenuItemImage *itemPlay = CCMenuItemImage::create("play.png", "play.png",
                    this, menu_selector(MainMenu::playGame));
itemPlay->setPosition(ccp(tamTela.width/2, tamTela.height * 0.5));

CCMenu* Menu = CCMenu::create(itemOpcoes, itemOpcoes, NULL);
Menu->setPosition(CCPointZero);
this->addChild(Menu, 10);
```

6.5 Colisão

O sistema de colisão do jogo foi determinado para definir quando o jogador perde pontos de vida e quando ganha pontos. O trecho de código abaixo exemplifica o método para teste de colisão entre dois corpos utilizado no jogo.

```
bool HelloWorldScene::checarColisao(CCSprite* sprite1, CCSprite* sprite2) {

CCRect rect1 = CCRectMake (sprite1->getPosition().x - (sprite1->getContentSize().width/2),
```

```
sprite1->getPosition().y - (sprite1->getContentSize().height/2), sprite1->getContentSize().width,
sprite1->getContentSize().height);
CCRect rect2 = sprite2->boundingBox();
if (rect1.intersectsRect(rect2)) {
        return true;
} else {        return false;
}
}
```

O método recebe dois sprites como parâmetro, cria dois retângulos que representem a área utilizada pelos sprites recebidos e então testa se há intersecção entre os dois retângulos utilizando o método “IntersectsRect“.

6.6 Persistindo dados

Devido a simplicidade do jogo, só foi necessário o armazenamento da pontuação do jogador. Para salvar a pontuação foi utilizado a classe CCUserDefault, esta age como um pequeno banco de dados. Você pode salvar e recuperar um tipo base conforme o trecho de código abaixo.

```
…
int highScore = CCUserDefault::sharedUserDefault()->getIntegerForKey(
                        "HighScoreKey");
if (score > highScore) {
        CCUserDefault::sharedUserDefault()->setIntegerForKey("HighScoreKey",
                                score);
        CCUserDefault::sharedUserDefault()->flush();
...
```

No código acima a variável highScore recebe um valor inteiro que estiver salvo na chave de nome HighScoreKey. Caso a pontuação atual (*score*) seja maior que a que estava salva, a chave HighScoreKey recebe o seu valor.

6.7 Adicionando efeitos de som

Cocos2D-X utiliza o *framework* CocosDenshion para manipulação de sons. Para utilizar o *framework* basta importar a biblioteca “SimpleAudioEngine.h” e adicionar o *namespace* CocosDenshion na classe onde irá utilizar os sons. Após isso basta pré-carregar o efeito de som e utilizá-lo quando necessário. O trecho de código abaixa demonstra como pré-carregar um efeito e usá-lo em determinado momento.

```
//pré-carregando
SimpleAudioEngine::sharedEngine()->preloadEffect(
                CCFileUtils::sharedFileUtils()->fullPathForFilename("explosion.wav").c_str());
//utilizando
                SimpleAudioEngine::sharedEngine()->playEffect("explosion.wav");
```

7 VALIDAÇÃO

O sistema foi validado a partir da interação de alguns usuários representando o público em geral. O teste consistiu em deixar o usuário interagir livremente com o jogo. Era esperado que o jogador pudesse, facilmente, interagir com o jogo.

Para a verificação de que os objetivos propostos foram alcançados, foi confeccionado um questionário, cujo modelo encontra-se disponível no apêndice A deste documento. A amostra populacional foi composta de 20 elementos, de diferentes faixas etárias, mas que não ultrapassou 30 anos, o que possibilitou a análise das diferenças entre as percepções dos usuários por faixa etária.

Os objetivos puderam ser validados pelos usuários após um período de 20 minutos jogando.

Tabela 7 - Qualidade em Uso

Característica	Atributo
Satisfação	O usuário se diverte ao jogar

Fonte – Autor

Tabela 8 - Qualidade Interna/Externa

Característica	Subcategoria	Atributo/Objetivo

Confiabilidade	Maturidade	O jogo apresenta erros
Usabilidade	Apreensibilidade	É fácil aprender o sistema do jogo
Funcionalidade	Adequação	O sistema sonoro possui efeitos de som
		O sistema de animação funciona
		O usuário possui controle sobre onde está tocando

Fonte – Autor

A relação entre o atributo de qualidade e a pergunta no questionário é apresentada na tabela seguinte.

Tabela 9 - Atributo x Questão

Atributo/Objetivo	Questão
O usuário se diverte ao jogar	O jogo é divertido?
O jogo apresenta erros	O jogo apresentou algum erro?
O sistema sonoro possui efeitos de som	Sons tocam corretamente?
O sistema de animação funciona	As imagens estão animadas de forma adequada?
É fácil aprender o sistema do jogo	Foi fácil aprender a jogar?
O usuário possui controle sobre onde está tocando	O controle é fácil? / O toque responde bem?

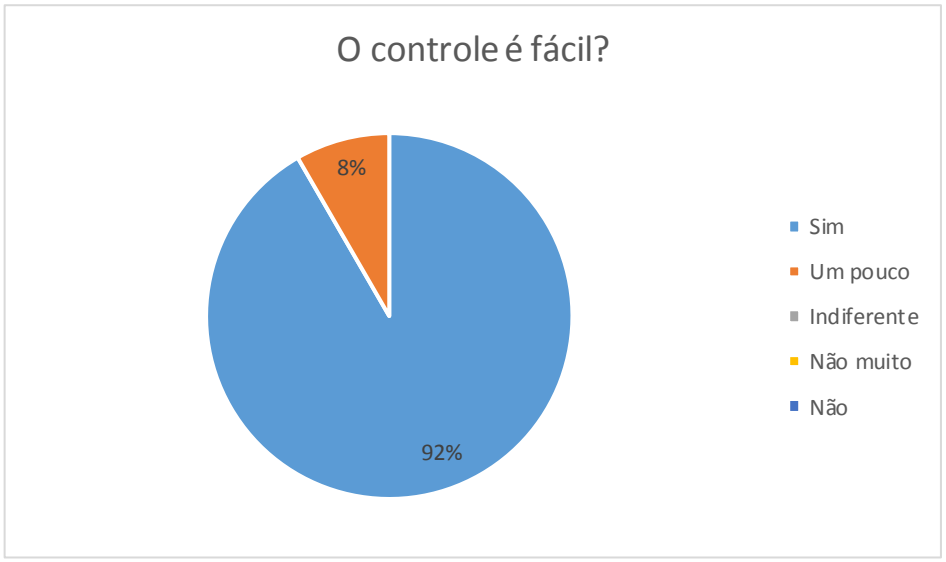
Fonte – Autor

Duas das questões foram levantadas para avaliar o retorno sobre o estado atual dos gráficos e sons. A ideia é validar se realmente existe a necessidade de retrabalhar essas duas questões, possibilitando uma maior satisfação do jogador.

Outro ponto importante foi a adição de perguntas que permitam identificar o perfil do usuário em relação a: gênero, idade, experiência com jogos e quais seus gêneros favoritos. Essa informação possibilita um melhor planejamento de ações de divulgação do mesmo, caso exista interesse, além da própria análise de dados.

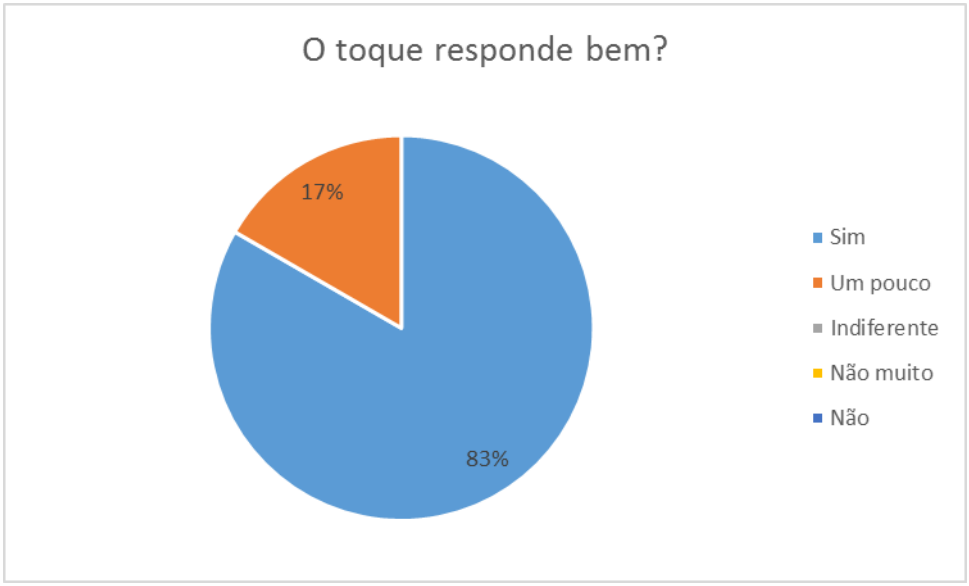
Com os dados coletados, podemos seguir com algumas análises. Começamos com uma visão de como ficou a avaliação de cada atributo.

Gráfico 1- O controle é fácil?



No Gráfico 1, pode-se observar que o controle é fácil. Entretanto, durante o jogo, diversas tentativas de utilização de comandos não suportados pelo jogo foram observadas, entre elas: arrastar objetos e inclinar o dispositivo.

Gráfico 2 - O toque responde bem?

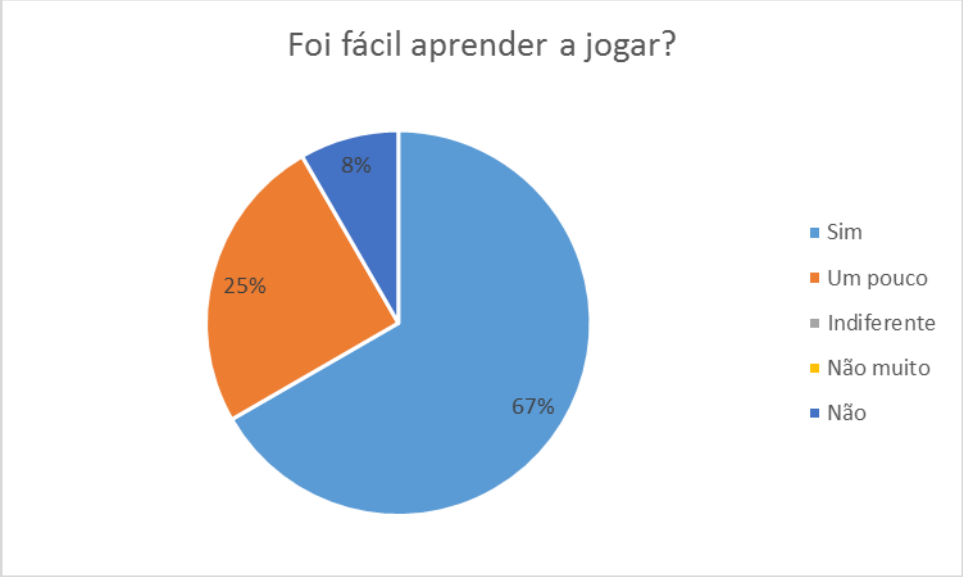


No gráfico 2 pode-se observar que o toque funciona bem. Não houve nenhum contratempo identificado nessa questão.

Gráfico 3 - Os gráficos são agradáveis?

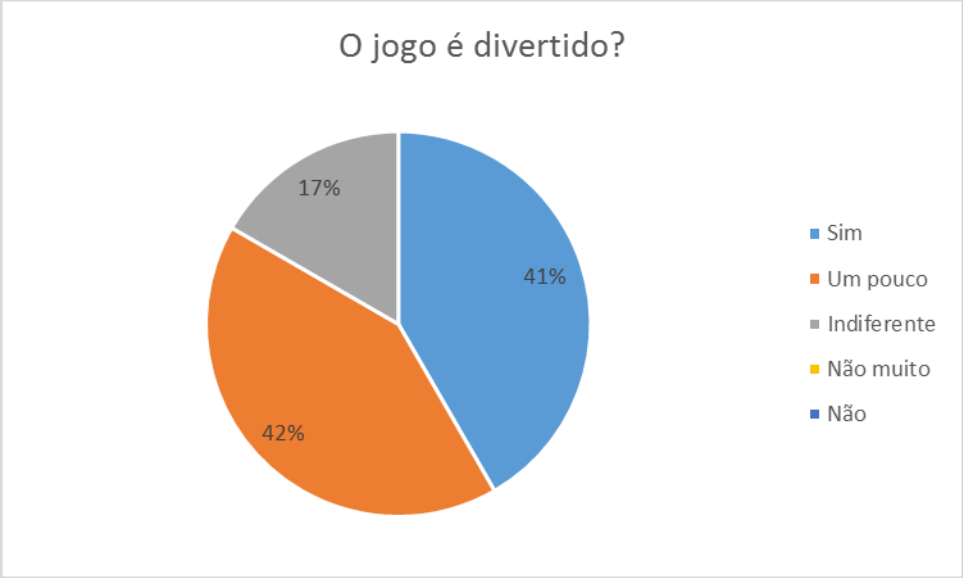
No Gráfico 3 aparece o que já era esperado: uma aceitação razoável dos gráficos atuais, mesmo que eles não sejam de um nível alto de elaboração.

Gráfico 4 - Foi fácil aprender a jogar?



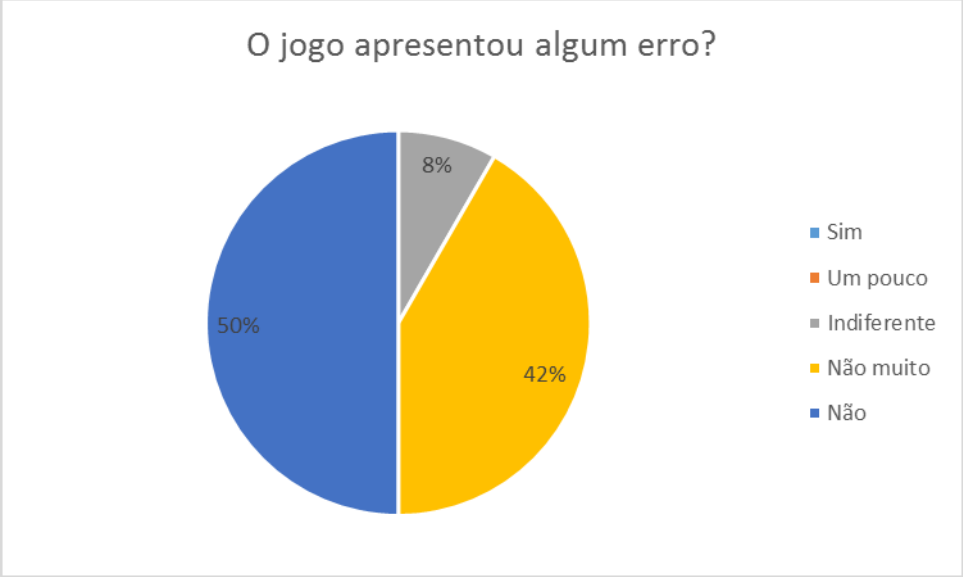
O Gráfico 4 mostra que nem sempre é necessário um tutorial. Em poucas tentativas os jogadores conseguiram entender a mecânica do jogo.

Gráfico 5 - O jogo é divertido?



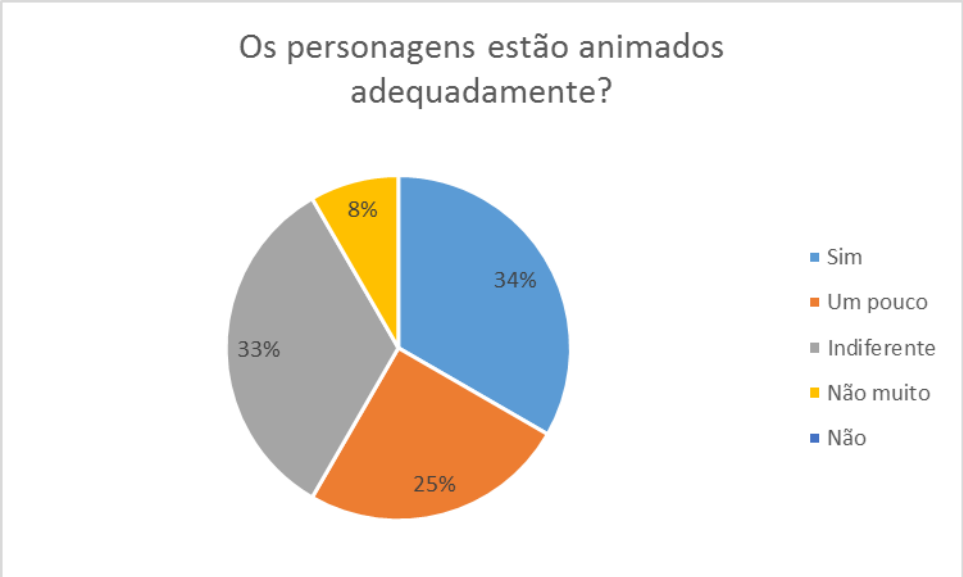
O Gráfico 5 demonstra que o jogo conseguiu agradar a grande maioria, com a apenas umas duas exceções. A maior queixa entre esses jogadores foi que o jogo se torna enjoativo muito facilmente.

Gráfico 6 - O jogo apresentou algum erro?



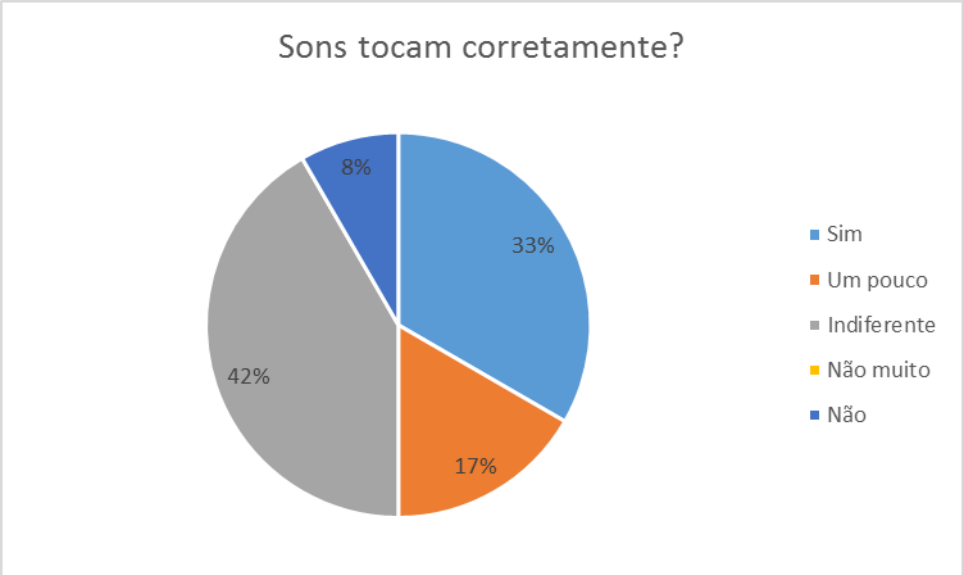
O Gráfico 6 demonstra que o jogo não apresentou erros aos jogadores, ou pelo menos nenhum foi citado durante a validação.

Gráfico 7 - Os personagens estão animados adequadamente?



As animações funcionaram como o esperado. Entretanto, alguns jogadores perceberam que o cenário permanece sem mudanças. Essa é um dos pontos a melhorar em futuras atualizações.

Gráfico 8 - Sons tocam corretamente?



Embora os efeitos sonoros tenham funcionado da forma esperada, muito dos jogadores queixaram-se a respeito da qualidade dos mesmos. Mais um ponto a melhorar em futuras atualizações.

Outra questão que vale ressaltar é a de que o jogo acabou sendo jogado em dispositivos diferentes. No dia da validação existiam 10 dispositivos *Android* disponíveis, sendo que apenas três modelos com configurações diferentes. Na tabela 6 abaixo, temos um resumo das configurações.

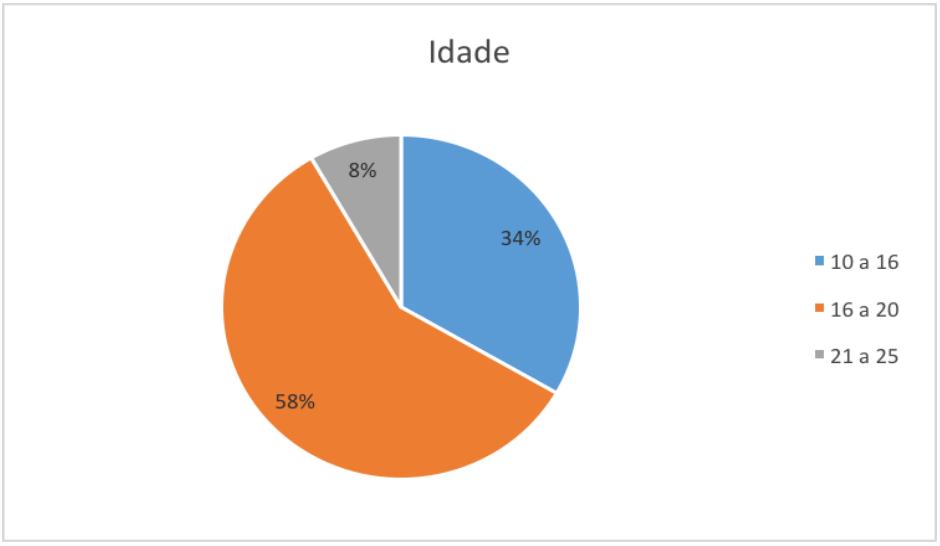
Tabela 10 - Configurações dos aparelhos utilizados na validação

Aparelho	HTC One M8 (Deselvolvimento)	Motorola Moto G	LG Optimus L5
Resolução de Tela	1920x1080	1280x720	800x480
Processador	Quad-core 2.3GHz	Quad-core 1.2GHz	Dual-core 1GHz
Versão do Android	4.4	4.4	4.1
Memória RAM	2 Gb	1 Gb	512 mb

O jogo manteve um excelente desempenho em todos os aparelhos. O único porém foi com os aparelhos de menor resolução. Nestes aparelhos, os gráficos ficaram levemente distorcidos. Esse é um ponto que deve ser trabalhado para um lançamento em mercado.

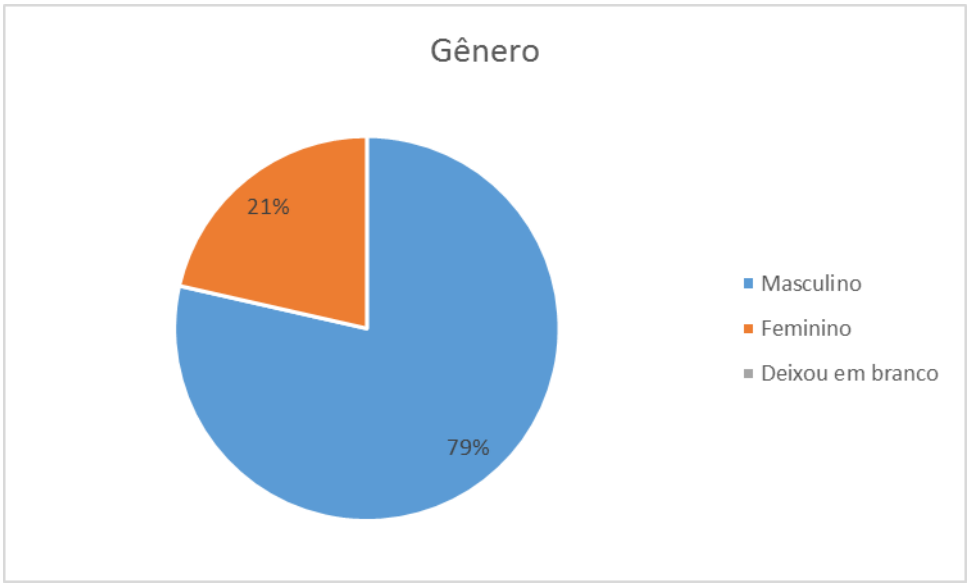
O total de participantes na validação foram de 16 jogadores. Alguns com maior convívio com jogos, outros com menos. Abaixo tem-se mais informações sobre o grupo.

Gráfico 9 - Idade



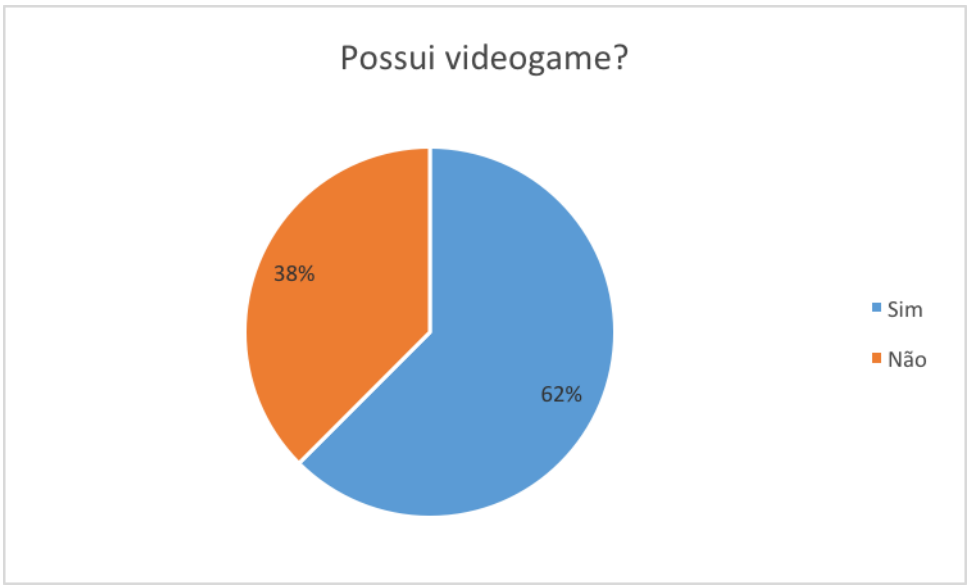
O Gráfico 11 mostra informações a respeito das idades dos jogadores do grupo que validou o trabalho. É um público jovem, em especial por conta do grupo escolhido ser de uma turma de um curso para essa faixa etária.

Gráfico 10 - Gênero



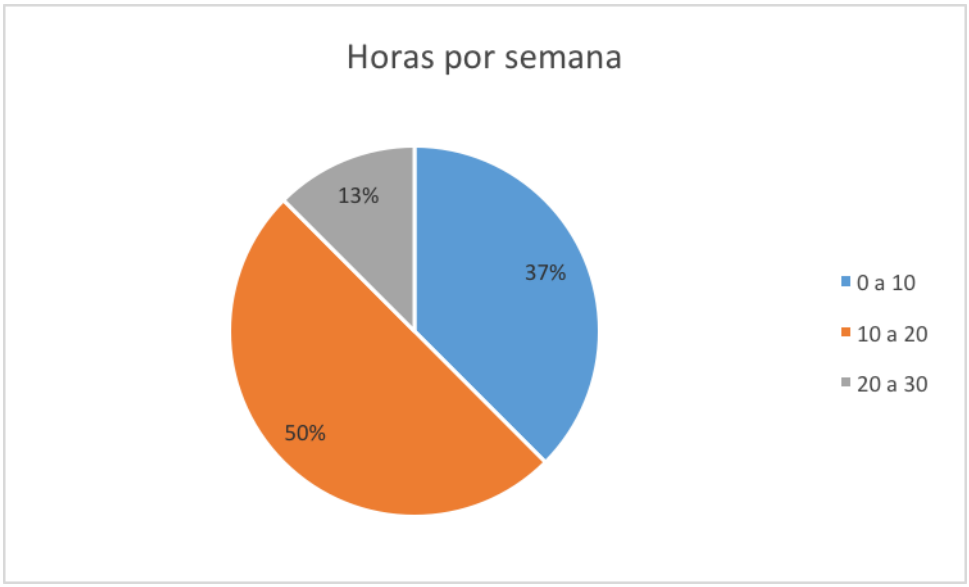
A maior parte do grupo é do gênero masculino, conforme verificado no gráfico 10. Uma das razões pela baixa quantidade de mulheres é a baixa quantidade de mulheres que participam do curso onde a validação foi aplicada.

Gráfico 11 - Possui videogame?



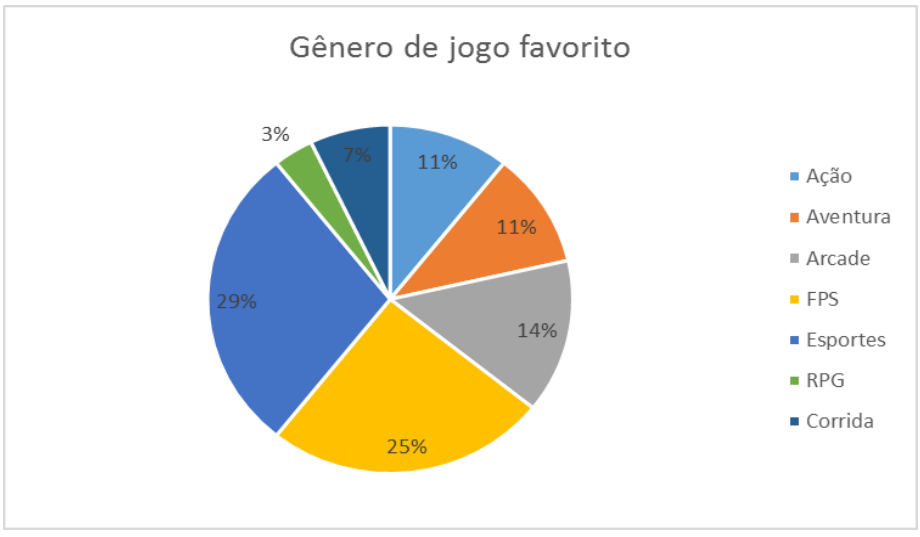
Apenas dois jogadores do sexo masculino e todos do sexo feminino não possuem um videogame, pois computador não foi considerado. O gráfico 11 reflete este ponto.

Gráfico 12 - Horas por semana



No Gráfico 12 é refletido a questão da posse de um videogame, com uma grande quantidade de jogadores jogando entre vinte ou menos horas por semana. Desse grande grupo, apenas 4 declararam não gastar nem uma hora por semana, apontando com um zero a quantidade de horas.

Gráfico 13 - Gênero Favorito



No Gráfico 14 tem-se o gênero favorito dos jogadores. Mesmo aqueles que não possuem *videogame*, ou declararam não jogar, escolheram um gênero sem perguntar se poderia deixar em branco. Como curiosidade, apesar de a opção ser esportes os jogadores citaram especificamente futebol como gênero favorito. Vale ressaltar que foi permitido escolher mais de um gênero.

Apesar do jogo ser bem avaliado, é interessante ver que Arcade ficou em terceiro lugar entre os gêneros favoritos do grupo, sendo os outros Ação e FPS.

A maioria dos participantes da validação possuíam um dispositivo móvel e, apesar destes dispositivos serem usados para jogos, não consideravam um aparelho para jogo. Para a maior parte deste grupo o conceito videogame se refere a um aparelho específico e exclusivo para jogos.

O tamanho da amostra foi pequeno, e por isso, pouco significativo. Os números citados fazem mais sentido se acompanhados do perfil de cada jogador. Infelizmente, cruzar e listar as informações dos jogadores pode agregar pouco a esta validação.

8 CONCLUSÃO

Este trabalho foi centrado no desenvolvimento de um jogo com o *game engine* Cocos2D-X. Tema escolhido em razão do crescimento do mercado de entretenimento, em especial do mercado de jogos. Tal crescimento está muito relacionado a grande expansão do mercado de dispositivos móveis.

Ao longo deste trabalho, diversos pontos do desenvolvimento de um jogo com Cocos2d-x foram abordados: manipulação de *sprites*, animação, criação de menus, colisão e sistema de som. Apesar do tema desta pesquisa ser muito relevante, foi apresentado uma breve introdução ao tema desenvolvimento de jogos para plataforma *Android*. Mesmo dentro do *game engine* escolhido, existe muito mais a ser explorado em termos de desenvolvimento de jogos.

Neste trabalho, também, foi possível validar que os diversos pontos desenvolvidos foram realizados de forma satisfatória. Os jogadores que participaram

da validação gostaram do jogo. Havia um receio sobre a aceitação das imagens utilizadas, mas os jogadores gostaram das mesmas.

Como trabalhos futuros, pode-se citar o desenvolvimento de um jogo mais complexo, com finalidade comercial.

REFERÊNCIAS

STRATEGY ANALYTICS, **Domínio esmagador: *Android* detém 84% do mercado mundial.** Disponível em: < http://m.dinheirovivo.pt/m/article?contentID=4217052> Acesso em 03 de Nov. de 2014.

DEWAY, **Desenvolvimento do Mercado de Aplicativo de Games.** Disponível em: <http://www.deway.com.br/crescimento-do-mercado-de-aplicativos-de-games> Acesso em: 16 de Out. de 2013.

WARD, Jeff, ***What is Game Engine?*** Disponível em: <http://www.gamecareerguide.com/features/529/what_is_a_game_.php> Acesso em 16 de Out. de 2013.

DAVILA, Gabriela, **Corona SDK: Uma Introdução ao Desenvolvimento Móvel de Jogos Multi - Plataforma. Faculdade de Tecnologia de Americana.** Americana - SP, 2012.

PERUCIA, Alexandre Souza et al. **Desenvolvimento de Jogos Eletrônicos.** 2. ed. São Paulo: Novatec, 2005. 320 p.

CARVALHO, G. H. P.**Um Modelo Preditivo para Desenvolvimento de Jogos de Computador.** 2006. 102p. Trabalho de Graduação (Graduação em Ciências da Computação). Universidade Federal de Pernambuco.

BECK, K., ANDRES, C. **Extreme Programming Explained: EmbranceChange.**Second Edition. 2004.

THE NEW PRODUCT DEVELOPMENT GAME***, The New Product Development Game.*** Disponível em: <http://mis.postech.ac.kr/class/MEIE780_AdvMIS/paper/part3/32_The%20new%20product%20development%20game.pdf> Acesso em 12 de Mai. de 2014.

SCHWABER, K. **Agile Project Management with Scrum**. Microsoft Press. Edição: 1ª. Ano: 2004.

INFO, **Crescimento de Vendas do Android.** Disponível em: < http://info.abril.com.br/noticias/mercado/android-tem-crescimento-de-79-5-nas-vendas-16052013-42.shl> Acesso em 17 de Out. de 2013.

LEE, Wei-Meng. **Introdução ao Desenvolvimento de Aplicativos para Android.** Rio de Janeiro: Ciência Moderna, 2011.

TEXTUREPACKER, **TexturePacker.** Disponível em: <http://www.codeandweb.com/texturepacker> Acesso em 12 de Mai. de 2014.

PHOTOSHOP, **Photoshop.** Disponível em: <http://www.adobe.com/br/products/photoshop.html> Acesso em 12 Mai. de 2014.

ECLIPSE, **Eclipse IDE.** Disponível em: <http://www.eclipse.org/downloads/>Acesso em 28 de Out. de 2013.

ANDROID SDK, **Android SDK.** Disponível em: <http://developer.android.com/sdk/index.html/> Acesso em 28 de Out. de 2013.

COCOS2D-X, **Cocos2D-X.** Disponível em: < http://www.cocos2d-x.org/> Acesso em 25 de Out. de 2014.

THE SOUND RESOURCE, ***The Sound Resource*.** Disponível em:<http://www.sounds-resource.com/> Acesso em 15 de Out. de 2014.

ABNT NBR ISO/IEC 25000:2008. Disponível em: <http://www.abntcatalogo.com.br/norma.aspx?ID=1552 >. Acesso em 02 de Jun. de 2012.

APÊNDICE A – Questionário de Validação

Identificação

Idade: ____

Gênero: ☐ Masculino ☐ Feminino

Possui algum video-game? ☐ Sim ☐ Não

Quantas horas você costuma jogar por semana? _____

Quantos videogames já possuiu? _____

Qual seu gênero favorito?

☐ Ação

☐ Aventura

☐ Corrida

☐ FPS

☐ RPG

☐ Arcade

☐ Outro: _____

Como você avaliaria os seguintes itens a respeito do jogo?

	Sim	Um pouco	Indiferente	Não muito	Não
O controle é fácil?					
O toque responde bem?					
Os gráficos são agradáveis?					
Foi fácil aprender a jogar?					
O jogo é divertido?					
O jogo apresentou algum erro?					
O personagem está animado adequadamente?					
Sons tocam corretamente?					