DESENVOLVIMENTO DE UM JOGO MULTIPLATAFORMA PARA DISPOSITIVOS MÓVEIS

Filipe Quadros Borges
Ernesto Lindstaedt

Resumo: O mercado de jogos para dispositivos móveis oferece uma ótima oportunidade para desenvolvedores independentes que tenham um desenvolvimento rápido e contínuo de novos jogos e conteúdo. O presente trabalho tem objetivo de desenvolver um jogo multiplataforma para dispositivos móveis baseado nas características mais comuns encontradas nos jogos de maior sucesso disponíveis nas lojas de aplicativos *mobile*. Ao final do trabalho, o objetivo foi alcançado com sucesso, foi possível implementar um protótipo do jogo com as características desejadas em HTML5 com Javascript para as plataformas Windows Phone 8 e Android. A arquitetura utilizada se mostrou adequada para o desenvolvimento, apenas com algumas ressalvas quanto ao desempenho em dispositivos mais antigos.

Palavras-chave: jogos. dispositivos móveis. multiplataforma. arquitetura. HTML5. Android. Windows Phone.

1 INTRODUÇÃO

1.1 Motivação

O mercado de jogos para dispositivos móveis passa por um momento bem aquecido, oferecendo uma ótima oportunidade para desenvolvedores independentes, também conhecidos como *indie*, e para jovens e pequenas empresas que podem, de um dia para o outro, ganhar muito dinheiro se algum de seus jogos caírem no gosto do grande público. Uma confirmação desse grande movimento do mercado é que, dentro dos já existentes sites e portais especializados no mercado de jogos, estão sendo criadas áreas e seções voltadas especificamente para os jogos de telefones celulares inteligentes (*smartphones*) e *tablets* (GAMASUTRA, 2013; GAMEDEV, 2013).

Um desenvolvedor independente com uma boa ideia tem condições de ganhar uma fatia lucrativa desse mercado, porém a grande concorrência e a velocidade da mudança do gosto do público é um inimigo dos pequenos. A possibilidade de que um pequeno produtor de jogos consiga viver somente desta

atividade está ficando cada vez mais difícil, pois depende de uma produção contínua de conteúdo para os jogos que estão em seu momento de sucesso e do desenvolvimento rápido e contínuo de novos jogos, que possam por sua vez cair nas graças do público. Um segundo fator que tem influência no sucesso do desenvolvedor de jogos é a abrangência de plataformas, pois cada plataforma suportada dá acesso a um grande número de potenciais usuários.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral desse trabalho é desenvolver o protótipo de um jogo multiplataforma para dispositivos móveis. O jogo escolhido será baseado em características comuns dos jogos de maior sucesso disponíveis nas lojas de aplicativos móveis no momento do desenvolvimento desse trabalho.

1.2.2 Objetivos Específicos

Os objetivos específicos desse trabalho são:

- Escolher uma tecnologia para desenvolvimento que permita desenvolver e publicar o protótipo para mais de uma plataforma móvel.
- Analisar o desempenho do protótipo em pelo menos duas configurações de hardware de dispositivos.
- Escolher uma arquitetura para o protótipo que facilite o desenvolvimento, a manutenção, a extensão e o reuso de seus e artefatos e código para o desenvolvimento futuro de outros jogos com características semelhantes.
- Analisar a possibilidade de desenvolver um *framework* com base no protótipo desenvolvido.

1.3 Delimitação do Estudo

Por restrição na disponibilidade de recursos para desenvolvimento desse trabalho o protótipo será desenvolvido e pulicado somente para as plataformas Android (versões 2.3 e 4.1) e Windows Phone 8. A plataforma de desenvolvimento utilizada é Windows 8, o que exclui o desenvolvimento para iOS, e os únicos

dispositivo disponíveis para teste são celulares Android e o emulador para Windows Phone 8.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- O capítulo 2 trata do referencial teórico para o assunto de desenvolvimento de jogos para dispositivos móveis. A primeira parte contém a pesquisa das características mais comuns dos jogos de sucesso. A segunda parte busca uma a arquitetura a ser utilizada para atingir os objetivos desse trabalho.
- No capítulo 3 temos a análise das tecnologias para desenvolvimento multiplataforma para dispositivos móveis e a escolha da opção mais adequada.
- O capítulo 4 apresenta os trabalhos relacionados, explorando um jogo de sucesso com características semelhantes ao objetivo final desse trabalho e um segundo trabalho que serviu de inspiração para a análise arquitetural.
- O capítulo 5 descreve a metodologia do trabalho e está organizado nas subseções: 5.1, metodologia da pesquisa, e 5.2, metodologia do desenvolvimento.
- No capítulo 6 é realizada a análise e apresentados os resultados do projeto.
- No capítulo 7 temos a conclusão e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Características dos Jogos para Dispositivos Móveis

Numa visão alto nível, jogos são universos virtuais populados por entidades cujo propósito é interagir com o jogador, possivelmente interagindo também entre si, criando uma experiência agradável e divertida. Nesse aspecto os jogos para dispositivos móveis são idênticos aos outros jogos para computadores pessoais ou consoles de videogame, e essa proposta precisa levar em conta as características mais específicas de jogos *mobile*.

A forma mais comum de classificação de jogos é denominada gênero e categoriza os jogos pela forma de interação e a experiência do usuário. Outras formas de classificação também podem ser utilizadas levando em conta características técnicas, visuais e integrações. Como exemplo podemos citar: ser ambientado em 2D ou 3D; ser um jogo em rede; ou um jogo que tenha interação com redes sociais.

Analisando os jogos para dispositivos móveis de grande sucesso nas lojas de aplicativos das plataformas móveis (APPLE, 2013; GOOGLE, 2013) podemos encontrar os principais gêneros a serem explorados: *arcade* (Tiny Wings), simulação (Pou), e *puzzle* ou quebra-cabeças (Angry Birds, Cut the Rope), que são os gêneros a serem explorados no desenvolvimento deste trabalho. Outra característica importante é o que todos esses jogos fazem parte de uma categoria chamada de jogos casuais, ou seja, jogos que tem uma jogabilidade simples e sessões de jogo rápidas, de menos de um minuto até alguns poucos minutos. Finalmente temos a característica de ser ambientado e com artes em 2D que é certamente uma das mais comum nos jogos em dispositivos móveis.

2.2 Arquitetura de Software para Jogos

Diversas arquiteturas e paradigmas diferentes podem ser utilizados ao estruturar o código para o desenvolvimento de um jogo (PLUMMER, 2004). O foco neste trabalho será encontrar uma arquitetura que facilite o desenvolvimento do protótipo e a sua manutenção e extensão. Outra característica que também será levada em conta é a facilidade do reuso de código de artefatos para o desenvolvimento futuro.

O assunto arquitetura é muito rico. Por exemplo, podemos analisar arquitetura em diversos níveis de abstração: sistemas, subsistemas, módulos e componentes, e a escolha de um modelo de arquitetura para um nível de detalhe em uma aplicação não invalida o uso de outro modelo em outro nível de detalhe. Com base no trabalho de PLUMMER (2004), podemos verificar que a arquitetura para jogos chamada s*ystem of systems* (sistemas de sistemas) se apresenta adequado para atender o objetivo aqui proposto pelas suas características de extensibilidade e facilidade de reuso de componentes já prontos e será utilizada como modelo. Essa arquitetura define bem como organizar os subsistemas responsáveis por cada domínio de

conhecimento necessário para a construção do jogo, como gráficos, inteligência artificial, áudio, etc. Porém, essa mesma fonte não se faz escolha de um modelo específico para definição das diversas entidades que formam o jogo, e esse será o assunto discutido ao longo dessa seção.

2.2.1 Arquitetura da Hierarquia de Entidades do Jogo

O foco da desta seção é fazer uma análise arquitetural em um nível mais baixo, o nível da definição de como as entidades que formam o universo do jogo são especificadas, atualizadas e interagem entre si e com o jogador. Estas serão chamadas neste trabalho simplesmente de entidades, mas dependendo do autor podem ter outras denominações das quais as mais comuns são: objeto, objeto-de-jogo, item, ator e unidade. Apesar de diferentes todos esses nomes se referenciam a um mesmo conceito.

Recorrendo tanto para literatura (GREGORY, 2009) quanto para os diversos blogs, fóruns, grupos e sites disponíveis na internet sobre o assunto (BILAS, 2002; GAMASUTRA, 2013; GAMEDEV, 2013; MARTIN, 2007; WEST, 2007), a principal divisão nesse aspecto é: modelo hierárquico versus modelo baseado em componente.

2.2.2 Modelo Hierárquico

O modelo hierárquico, ou centrado em objeto, é a aplicação direta do conceito de herança da orientação a objetos para criação dos diversos tipos de entidades diferentes, onde uma classe base de entidade é estendida pelas classes de entidades mais específicas (figura 1). Esse modelo é geralmente implementado com uma arquitetura de camadas, com toda a lógica do jogo, onde residem as diversas entidades, desenvolvida na camada superior utilizando a infraestrutura – desenho, física, entrada, GUI – que reside nas camadas inferiores. É o mais utilizado para jogos e vários *frameworks* e *engines* seguem esse conceito. Como exemplo podemos citar OGRE3D (OGRE, 2012) e COCOS2D e suas diversas variações (COCOS2D, 2013; COCOS2D-X, 2010).

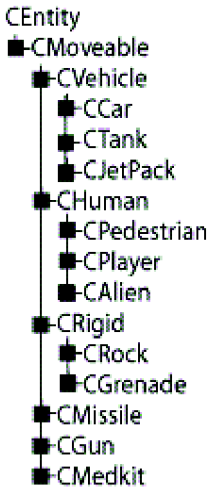

Figura 1: Modelo hierárquico de entidades.
Fonte: WEST, 2007.

A vantagem desse modelo é que ele é simples e passa despercebido pelo nível arquitetural maior: não é necessário deter-se nele, ele surge como efeito colateral da orientação a objetos empregada para desenvolver o software. É facilmente utilizado por quem está acostumado a desenvolver sistemas em orientação a objeto (GREGORY, 2009).

O seu maior problema é que normalmente ele acaba com hierarquias profundas e funcionalidades difíceis de agregar (BILAS, 2002; WEST, 2007). Por exemplo, teríamos uma classe entidade que seria a base de toda a hierarquia: entidade → desenhável → físico → inimigo. Nesse caso todo inimigo participa da simulação física e é desenhável. O impacto de especificar um inimigo que não participa da simulação de física (ex. um fantasma) ou um objeto físico não desenhável (ex. um ponto de gravidade) seria uma revisão de toda hierarquia: para compormos entidades diferentes das atuais precisamos reestruturar a hierarquia de classes.

2.2.3 Baseado em Componente

Este modelo usa composição ao invés de extensão para separar a funcionalidade e as características das entidades em pedaços modulares chamados de componentes, dessa forma uma entidade existe e é definida pela composição de suas partes (GREGORY, 2009; MARTIN, 2007; WEST, 2007). A figura 2 ilustra um exemplo de implementação desse modelo.

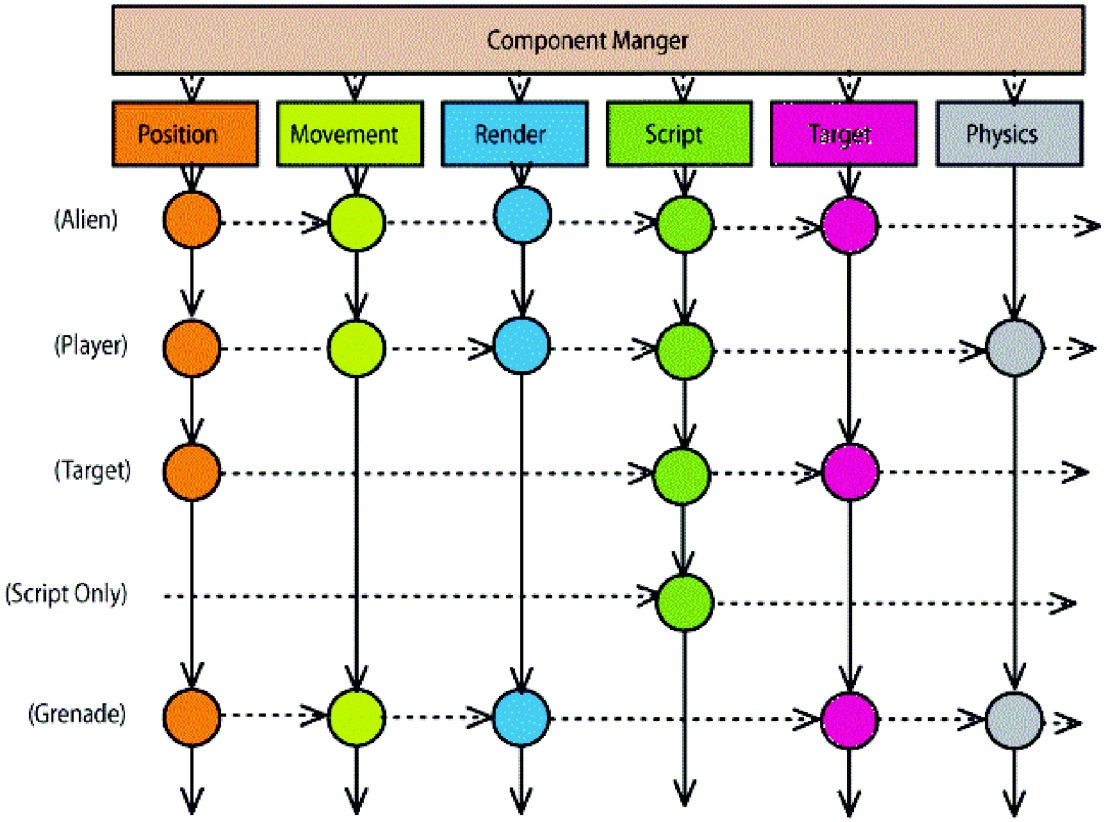

Figura 2: Modelo arquitetura de entidades baseado em componentes.
Fonte: WEST, 2007.

Ele pode ser implementado em diversos paradigmas, mas normalmente é implementado em uma linguagem orientada a objetos, porém sem uso abusivo de herança. Ele abre caminho para que a arquitetura maior da aplicação seja um sistema de sistemas centrado em dados, o que favorece muito o reuso e substituição de partes isoladas da base de código do jogo e também facilita a evolução contínua em diferentes composições das entidades para melhorar a experiência do jogador (PLUMMER, 2004). Diversos *frameworks* e *engines* fazem uso desse modelo, por exemplo: Unity3D, Artemis Framework e MelonJS.

Existem duas correntes neste modelo:

- Componentes somente dados: também chamado de *entity systems* é normalmente relacionado com *data oriented design* e restringe os componentes a armazenarem somente dados. As entidades são formadas pelos componentes, mas com sistemas implementando os comportamentos. Separa os comportamentos dos dados para ganhar em

clareza e simplicidade dos algoritmos e em desempenho. Exemplo: Artemis Entity System Framework (ARTEMIS, 2012).

- Componentes como comportamento e dados: Os componentes agregam duas responsabilidades: dados e comportamentos. As entidades são montadas com a agregação de diversos componentes como peças de lego, que definem todos os comportamentos e características da entidade que os possuem. É uma arquitetura com características menos restritas que um *entity system*. Exemplos: Unit3d, Crafty.js, MelonJS.

2.2.4 O Modelo Selecionado

O modelo de arquitetura selecionado foi o baseado em componente, com componentes somente dados. As características desse modelo se mostram adequadas para simplificar o desenvolvimento e facilitar o reuso de códigos existentes e a criação de conteúdo. A característica de separar os comportamentos dos dados para ganhar em clareza e simplicidade dos algoritmos torna a manutenção mais fácil, e o fato dos componentes serem somente dados facilita a criação de *templates* de entidades serializados, que por sua vez facilita o processo de criação.

3 DESENVOLVIMENTO MULTIPLATAFORMA PARA DISPOSITIVOS MÓVEIS

Existem diversas soluções para desenvolvimento multiplataforma de jogos para dispositivos móveis e cada uma dessas opções oferece características diferentes que devem ser consideradas: curva de aprendizado, complexidade, linguagem utilizada, ferramentas disponíveis e nível de desempenho. Este capítulo apresenta algumas das opções mais utilizadas e que serão avaliadas para desenvolvimento deste trabalho.

3.1 Opção 1: desenvolvimento nativo em linguagem C/C++

O desenvolvimento em linguagem C/C++ é há muitos anos a preferência para desenvolvimento de jogos AAA (jogos profissionais de alta qualidade) para plataformas tradicionais como consoles e computadores pessoais (GREGORY, 2009), sendo nesse caso praticamente a única linguagem utilizada. Os principais

motivos identificados para isso são o desempenho, a liberdade de acesso aos dispositivos de hardware e o suporte em diversas plataformas.

Porém no ambiente móvel essa não parece ser a mesma realidade. Discussões sobre o assunto em sites da internet apontam que apesar do desenvolvimento nativo em C/C++ ser muito popular para *mobile* multiplataforma, outras opções também disputam esse espaço com outras opções (GAMASUTRA, 2013; GAMEDEV, 2013).

Pontos positivos dessa opção:

- Riqueza de material sobre o assunto.
- Maior desempenho.
- Suporte em diversas plataformas moveis. Ex. Android, iOS, Windows Phone, BlackBerry.

Pontos negativos dessa opção:

- Complexidade da linguagem.
- Grande curva de aprendizado.
- Cada plataforma que for suportada deve ser adaptada na base de código, atividade que demanda esforço.

3.2 Opção 2: MonoGame

O projeto MonoGame foi criado inicialmente para permitir que desenvolvedores utilizando a plataforma .NET e o *framework* XNA pudessem publicar seus jogos para iOS. Atualmente ele suporta uma série de plataformas móveis incluindo Windows Phone 8, Android e iOS (MONOGAME, 2013).

O sítio *web* dessa solução apresenta uma riqueza de jogos e *engines* que são construídas utilizando-a como base, demonstrando a sua popularidade. A documentação existente apesar de incompleta é de fácil acesso e existe uma comunidade ativa de desenvolvedores (MONOGAME, 2013).

Pontos positivos dessa opção:

- Apresenta bom desempenho, apesar de inferior comparado com C/C++.
- Suporte nas plataformas móveis: Android, iOS e Windows Phone 8.

Pontos negativos dessa opção:

- Complexidade na publicação para iOS e Android.

- Publicação para plataformas iOS e Android utilizam as bibliotecas Xamarin que são proprietárias e externas ao Monogame.
- Publicação para plataformas iOS e Android precisam ser adaptadas para uso das bibliotecas Xamarin, atividade que demanda esforço extra.

3.3 Opção 3: desenvolvimento em HTML5 com javascript

A plataforma *web* sempre teve uma grande produção de produtos de *software*. O desenvolvimento de aplicações *online*, sites e serviços *web* utilizando tecnologias HTML, JavaScript e relacionadas é atualmente uma das áreas mais movimentadas da informática. Apesar disso, antes da proposta de padronização do HTML5, as tecnologias ligadas à *web* não eram populares entre os desenvolvedores de jogos.

A tecnologia HTML5 e a criação dos novos elementos *canvas* e *audio* causaram o aparecimento de diversos jogos, *frameworks* e *engines* desenvolvidos em HTML com JavaScript. Sua popularidade está em pleno crescimento e a comunidade de desenvolvedores que usa essa solução é ampla, muita documentação e exemplos do uso dessa tecnologia para jogos estão disponíveis em sites, *blogs* e fóruns na internet. Com essa tecnologia é possível desenvolver jogos que rodam em qualquer plataforma, móvel ou não, que disponibilize um navegador moderno compatível com HTML5.

A popularização dessa nova versão das tecnologias *web* em um momento em que os dispositivos móveis também se tornam populares provocou o surgimento de soluções para desenvolvimento de aplicativos móveis multiplataforma utilizando a tecnologia HTML5. São exemplos o Phonegap e Titanium que permitem desenvolver um aplicativo totalmente em HTML e empacota-lo como uma aplicação nativa para diversas plataformas móveis com um pequeno esforço. Utilizando alguma dessas soluções é possível desenvolver um jogo completamente em tecnologia HTML5 e publicá-lo para diferentes plataformas *mobile*. O Phonegap, especificamente, disponibiliza um serviço de *build online* que, em conjunto com alguns cuidados durante o desenvolvimento, publica o aplicativo para diversas plataformas móveis com esforço muito próximo de zero.

Pontos positivos dessa opção:

- Facilidade de desenvolvimento.

- Baixa curva de aprendizado.
- Riqueza de material sobre o assunto.
- Suporte em qualquer plataforma móvel que disponibilize um navegador moderno compatível com HTML5.
- Facilidade de publicação das aplicações.

Pontos negativos dessa opção:

- Resolução de problemas de compatibilidade entre diferentes navegadores e plataformas é uma atividade que demanda esforço extra.
- Desempenho é menor comparando com as outras soluções.

3.4 Comparação entre opções

Toda opções se apresentam adequadas para o desenvolvimento multiplataforma de jogos para dispositivos móveis. A tabela 1 é uma tabela comparativa compilada conforme suas vantagens e desvantagens das opções. Os sinais de mais (+) ou menos (-) representam respectivamente uma vantagem ou desvantagem. Sinais duplos reforçam uma característica frente às outra opções.

	Nativo C/C++	**MonoGame**	**HTML5 + Javascript**
Facilidade	-		+
Documentação	+		+
Plataformas suportadas	+	+	+
Esforço de adaptação	--	-	-
Desempenho	++	+	-

Tabela 1: Vantagens e desvantagens das opções.

3.5 Escolha da Opção

A opção escolhida foi o desenvolvimento utilizando HTML5 com Javascript por apresentar características mais adequadas ao objetivo do trabalho. A facilidade

de desenvolvimento e facilidade de publicação de aplicações estão alinhadas com desenvolvimento rápido e o suporte de diversas plataformas móveis é um dos pontos de interesse da proposta.

Analisando os pontos negativos percebemos que não são impeditivos de seu uso: os problemas de compatibilidade são facilmente contornáveis utilizando alguma das diversas bibliotecas disponíveis para isso, o desempenho menor é aceitável para o gênero de jogos escolhidos com a restrição de ambiente 2D.

A opção MonoGame foi excluída pela necessidade de uso de bibliotecas externas proprietárias para publicação da aplicação em plataformas Android e iOS, característica considerada impeditiva pelo custo de licenciamento desses componentes. Desconsiderando-se esse problema essa solução seria também adequada para o desenvolvimento do trabalho proposto.

4 TRABALHOS RELACIONADOS

4.1 Série de Jogos Angry Birds

Figura 3: Captura de tela do jogo Angry Birds.
Fonte: GOOGLE, 2013.

Angry Birds (figura 3) é um jogo multiplataforma da produtora Rovio (ANGRY, 2009). Originalmente desenvolvido somente para iOS ele foi portado para múltiplas plataformas móveis (Android, Windows Phone, Blackberry) e talvez seja o maior

exemplo de sucesso de jogos para dispositivos móveis, extrapolando até chegar em outros tipos de dispositivos como consoles e computadores. É um jogo casual com *gameplay* simples cujo objetivo é arremessar pássaros para destruir porcos que estão posicionados no cenário.

Ao longo do tempo diversos novos capítulos – conjunto de níveis – foram publicados, adicionando novo conteúdo ao jogo de forma a manter o público interessado e consumindo esse conteúdo. Para aproveitar ainda mais o sucesso do jogo Angry Birds original, os desenvolvedores passaram também a publicar uma série de jogos derivados e relacionados criando uma franquia de jogos. Os jogos já lançados dessa franquia são:

- Angry Birds (2009)
- Angry Birds Seasons (2010)
- Angry Birds Rio (2011)
- Angry Birds Space (2012)
- Angry Birds Star Wars (2012)
- Angry Birds Friends (2012)
- Bad Piggies (2012)
- Angry Birds Star Wars II (2013)

Em comparação com o original esses jogos são: uma variação do conteúdo do jogo (Angry Birds Seasons), variação de algumas regras do gameplay (Angry Birds Space) ou um jogo completamente diferente (Bad Piggies), mas que compartilha características básicas com o original. Essa capacidade produção contínua de conteúdo e novos jogos é ponto chave no sucesso da produtora Rovio.

Após uma breve análise, é possível afirmar que esse jogo apresenta uma série de propriedades que são relacionadas com os objetivos desse projeto. As características do jogo são aderentes ao encontrado na seção 2.1: esse é um jogo casual, 2D de gênero *puzzle* (quebra-cabeças), publicado para várias plataformas móveis (iOS, Android, Windows Phone 8, Blackbarry). Além disso o sucesso do jogo foi alavancado com a capacidade de produção contínua de conteúdo e pela produção de diversos novos jogos semelhantes.

4.2 Artemis Entity System Framework

Artemis (ARTEMIS, 2012) é um *Entity System framework* de alto desempenho para jogos, escrito em Java, que faz a gestão de entidades em um mundo virtual de jogo. Ele é inspirado na série de postagens intitulada *Entity Systems are the future of MMORPG* por Adam Martin (MARTIN, 2007) em seu *blog*. Ele é uma implementação que segue o modelo arquitetural para as entidades do jogo baseado em componentes somente dados, que foi escolhido para a implementação do protótipo desse trabalho.

O *framework* Artemis foi usado no desenvolvimento de diversos jogos e foi portado para várias linguagens. Apesar disso, o sítio *web* do *framework* contém um aviso que diz que ele ainda é considerado um trabalho experimental baseado em um conceito experimental e seu uso implica em um risco a ser assumido por conta do próprio utilizador.

Esse *framework* é baseado no conceito de que as entidades em um universo virtual de um jogo existem somente como identificadores puros, seus componentes contem somente dados e sistemas processam as entidades com base em seus aspectos, ou seja, a características de possuir ou não um conjunto de componentes. Isto promove a separação de conceitos e simplifica o *design* do jogo.

Do ponto de vista do utilizador do *framework*, as principais classes que o compõem são:

- World: Classe que cria e armazena as entidades e também tem a responsabilidade de executar os sistemas que processam essas entidades.
- Component: Classe base para todos os componentes. Os componentes devem ser definidos pelo utilizador e somente conter dados e nenhuma lógica do jogo.
- Entity: Classe que identifica uma entidade do jogo. Ela serve como um agrupador dos componentes que a formam e definem suas características.
- EntitySystem: Classe base para os sistemas que processam as entidades. Os sistemas devem ser definidos pelo utilizador e conter a lógica do jogo, e são onde realmente a implementação do jogo reside. Um sistema tipicamente seleciona as entidades de acordo com o aspecto de seu domínio, ou seja, a característica de conter ou não uma combinação de

componentes relacionados ou domínio de conhecimento do sistema e as processa de acordo com a sua funcionalidade específica. Por exemplo, um sistema de desenho poderia selecionar somente as entidades que possuem componentes Posicao e RepresentacaoVisual e processar elas desenhando essas entidades na tela. Outro sistema, SistemaDeMovimento, poderia selecionar as entidades com os componentes Posicao e Velocidade e processar elas atualizando o valor do componente Posicao com base no valor atual e no valor do componente Velocidade.

O diagrama de classe a seguir (figura 4) apresenta uma visão simplificada do relacionamento entre as classes que formam esse *framework*, do ponto de vista do utilizador.

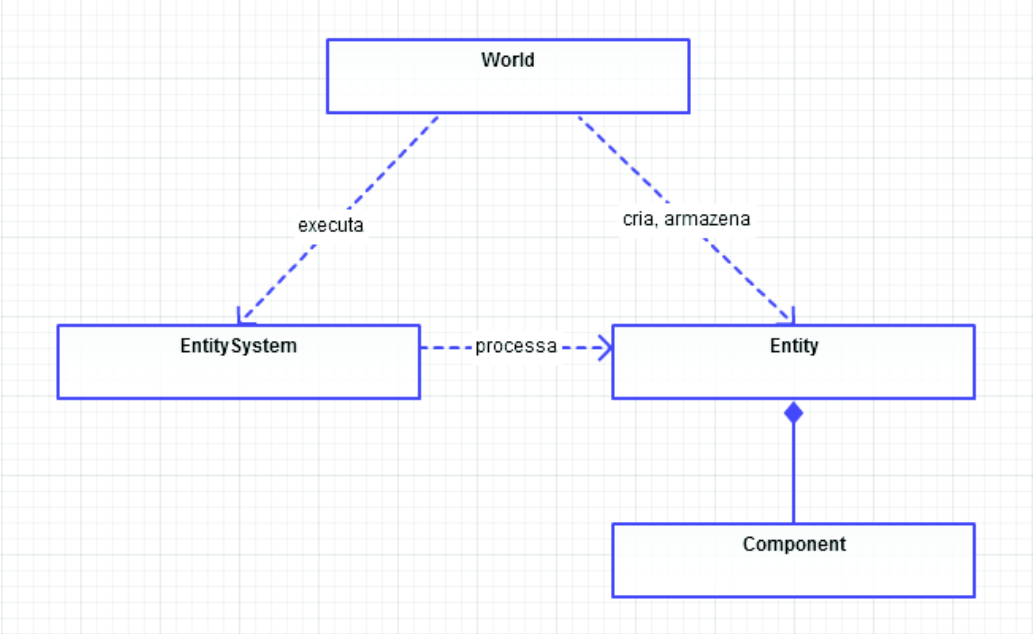

Figura 4: Diagrama de classes: visão simplificada do framework Artemis.

5 METODOLOGIA

5.1 Metodologia da Pesquisa

Foi realizada uma pesquisa bibliográfica para o levantamento das necessidades do mercado e descrição teórica sobre os assuntos de jogos para dispositivos móveis, arquitetura de *software* para jogos e desenvolvimento multiplataforma para dispositivos moveis. A lista dos requisitos foi desenvolvida com

base nos objetivos a serem atingidos por esse trabalho. Aplica-se, neste contexto, o método de estudo de caso único tendo em vista realizar a validação das teorias citadas, permitindo um aprofundamento e comprovação do referencial teórico. Como técnica, a coleta de dados foi utilizada a observação sistemática na forma de verificação dos requisitos utilizando testes de *software*.

5.2 Metodologia do Desenvolvimento

5.2.1 Escolha do Jogo

A escolha de um jogo para basear o desenvolvimento deve levar em conta as características de jogos para dispositivos móveis encontradas na seção 2 deste trabalho – jogo casual, 2D de gênero *arcade* ou quebra-cabeças. Para facilitar o foco na parte técnica também é interessante a escolha de um jogo que já tenha os artefatos de arte, como imagens e sons, disponíveis para ser utilizado.

BREAKOUTS (2013) é um sítio popular na *web* que reúne diversas implementações em código aberto de uma versão do clássico jogo Breakout, com o objetivo de comparar *engines* para jogos em Javascript. Apesar desse sítio não ser especializado para dispositivos móveis, essa característica é considerada um bônus na comparação entre as diferentes implementações e algumas delas afirmam suportar esses dispositivos. Por apresentar as características verificadas como típicas de jogos para dispositivos móveis, disponibilizar os artefatos de arte publicados em licença aberta e por ter implementações de referências que podem ser utilizadas na avaliação dos resultados e comparação o jogo Breakout foi escolhido para ser implementado neste trabalho.

O jogo Breakout (figura 5) é um jogo de gênero *arcade* cujo objetivo é rebater a bola (1) com o objetivo de quebrar todos os blocos (2). A plataforma móvel no inferior da tela – ou *paddle* – (3) deve ser controlada pelo jogador com o objetivo de rebater a bola e impedir que ela se mova para além da linha de base do cenário, caso em que o jogador perde uma jogada. Ao destruir blocos alguns itens – ou *powerups* – (4) podem cair e, ao tocarem no *paddle*, serem consumidas e resultarem em alguma vantagem como uma bola extra ou uma desvantagem como perder uma bola ou diminuir o tamanho do *paddle*.

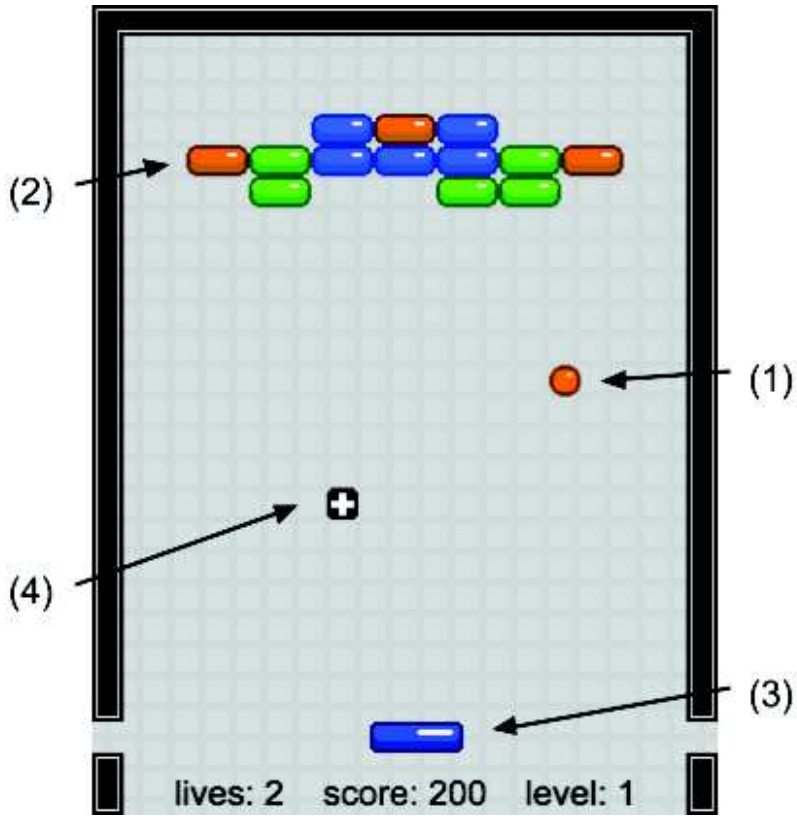

Figura 5: Captura de tela do jogo Breakouts.
Fonte: BREAKOUTS, 2013.

5.2.2 Arquitetura do Protótipo

5.2.2.1 Requisitos

Lista de requisitos funcionais:

- Apresentar tela *loading*: quando o jogador abre o aplicativo do jogo uma tela deve ser exibida enquanto os artefatos são carregados.
- Apresentar tela de menu: o jogo deve apresentar uma tela de menu a partir da qual o jogador pode iniciar o jogo.
- Mover o *paddle*: o jogador deve poder mover o *paddle* para rebater bolas e coletar *powerups*.
- Rebater a bola: quando uma bola toca no *paddle* ela deve inverter o sentido do componente vertical da sua velocidade.

- Coletar *powerups*: quando um *powerup* toca o *paddle* o *powerup* é destruído e um efeito é disparado (uma bola extra é criada para um “+” e uma bola é destruída para um “-”).
- Destruir os blocos: quando uma bola toca um bloco, esse bloco é destruído.
- Gerar *powerups*: quando um bloco é destruído, há uma chance de um *powerup* ser gerado.
- Perder o jogo: quando não houver mais nenhuma bola em jogo o jogador perde o jogo.
- Ganhar o jogo: quando não houver mais nenhum bloco em jogo, o jogador ganha o jogo.
- Animar *sprites*: o jogo deve poder exibir as entidades utilizando animações definidas por um conjunto de imagens que serão exibidas em sequência.

Liste de requisitos não-funcionais:

- Suporte às plataformas Android, versões 2.3 e 4.1, e Windows Phone 8.
- Controle totalmente através da tela sensível a toque.

5.2.2.2 Principais Pontos no Desenvolvimento do Protótipo

Antes de tratar especificamente do desenho técnico (*design*) de algum *software* é de boa prática o levantamento dos pontos mais importante e dos riscos arquiteturais a serem enfrentados. Para o protótipo desse trabalho, esses principais pontos são extraídos de assunto de desenvolvimento de jogos, mas por esse assunto ser extenso e já contar com muito conhecimento desenvolvido tanto pela indústria quanto pela academia, ele não foi abordado em detalhes nas seções anteriores desse trabalho.

Os pontos importantes a serem implementados em jogos são listados e brevemente comentados nessa seção: o *loop* do jogo, o conceito de cenas e o conceito de entidades:

- O *loop* do jogo (ou *gameloop*) é a simulação de passagem do tempo em um jogo, uma atualização periódica dos subsistemas que compões o jogo, que dá a característica de tempo real.
- Cenas (ou estados do jogo) é a subdivisão de um jogo em diferentes estados de alto nível em que a interação ou os recursos são diferentes,

por exemplo, a tela de abertura, o menu principal, a tela de jogo principal, etc.

- Entidades, já citadas anteriormente, são os objetos que populam o universo do jogo e interagem com o jogador e entre si. A interação das entidades e o jogador é um dos fatores que mais influencia na experiência de jogo que o jogador terá.

5.2.2.3 Modelo Arquitetural de Referência

A inspiração para o modelo de arquitetural desse projeto é o trabalho de PLUMMER (2004), a arquitetura para jogos chamada *system of systems* (sistema de sistemas). Essa arquitetura organiza as responsabilidades de cada domínio de conhecimento necessário para a construção do jogo em subsistemas. Um sistema central agrega os diversos subsistemas que são responsáveis cada um por um domínio de conhecimento.

Ao desenvolver um modelo arquitetural de referência usando essa inspiração, e mantendo a simplicidade, ficamos com a arquitetura ilustrada na figura 6. Esse modelo tem um sistema principal composto de subsistemas. Esses subsistemas são compatíveis com uma interface, utilizada pelo sistema principal como protocolo de comunicação com os subsistemas.

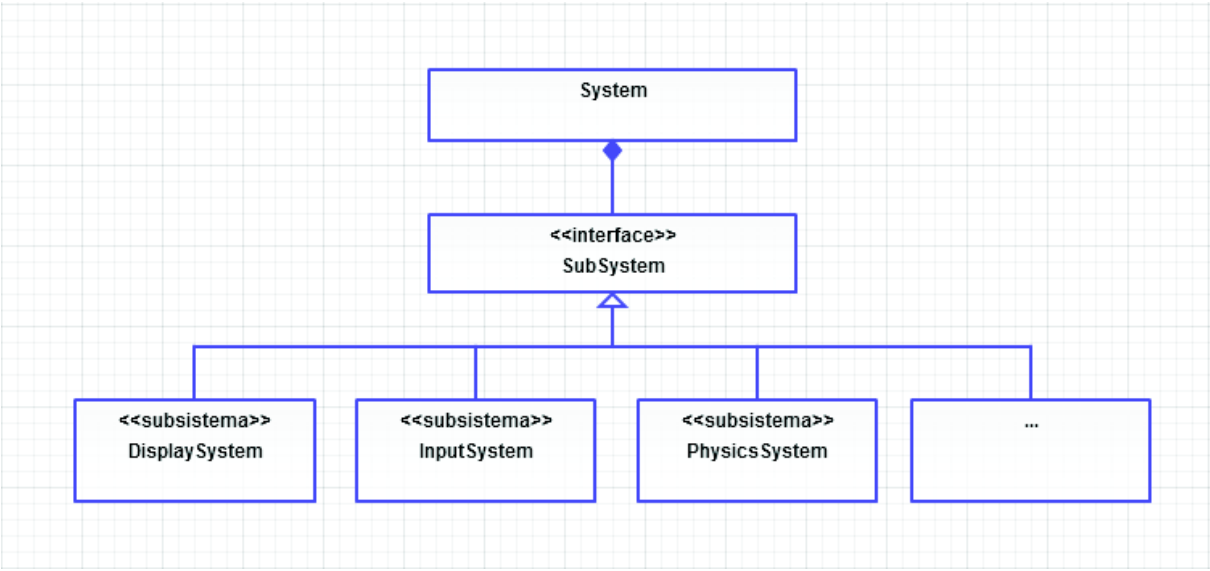

Figura 6: Modelo da Arquitetura.

5.2.2.4 Entidades do Jogo

O modelo arquitetural escolhido para a definição das entidades de jogo é o modelo conhecido como *Entity Component System*, já apresentado anteriormente. O trabalho original que serve como inspiração para o projeto e implementação desse item é o Artemis Entity System Framework (ARTEMIS, 2012).

Com base nos conceitos utilizados no *framework* Artemis e utilizando o nosso modelo arquitetural de referência temos um *Entity Component System* (ECS) simples, porém completo. A figura 7 apresenta um diagrama de classes desse ECS. Nesse diagrama estão desenhadas em claro os elementos gerais e em escuro os elementos referentes ao domínio de exibição gráfica de entidades. Os demais domínios (física, regras do jogo, animação, etc.) seguem o mesmo padrão e não foram representados para manter a clareza.

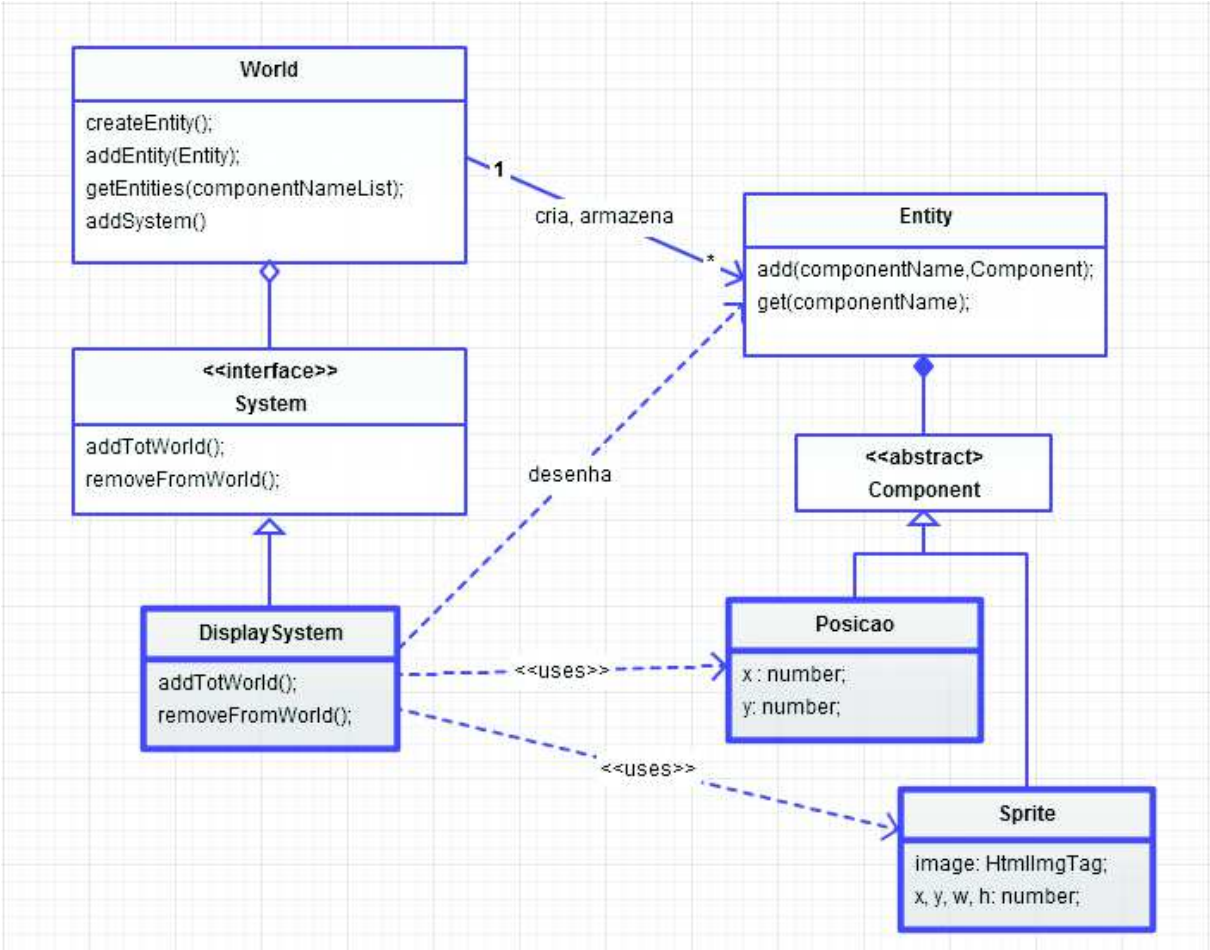

Figura 7: diagrama de classes *Entity Component System*.

Nesse modelo a classe *World* é a raiz de todo o jogo, é o mundo virtual que contém todas as entidades e agrega todos os sistemas (classe *System*). A interface *System* representa a interface para um sistema que processa entidades. Uma entidade (*Entity*) é definida pela composição de diversos componentes. Para iniciar um jogo, um objeto *World* deve ser criado, todos os objetos do tipo *System* devem ser criados e adicionados ao *World*. Em seguida as entidades são criadas, com um conjunto de componentes e adicionadas ao *World*. A partir desse ponto o jogo pode ser executado.

Uma observação quanto à linguagem Javascript. A linguagem Javascript não tem classes ou interfaces, somente objetos, e tem tipagem dinâmica. O diagrama (figura 7) apresenta interfaces, porém essas representam somente um protocolo a ser seguido, pois interfaces não existem em Javascript. De forma semelhante temos a classe abstrata *Component*. Na implementação todos os componentes são objetos Javascript puros, que devem conter as propriedades específicas do tipo do específico de componente.

Abaixo (Figura 8) um exemplo de código mostrando como as entidades são definidas pela composição de componentes (extraído do código do jogo):

```
var BLOCK = 16; // tamanho de cada bloco das sprites
// create ball
var ball = world.createEntity();
ball.add('sprite', {imgid: 'img_tiles', w: 16, h: BLOCK,
                                        x: 7 * BLOCK, y: 4 * BLOCK});
ball.add('position', {x: 2 * BLOCK, y: 10 * BLOCK});
ball.add('rigidBody', {bodyType: 'dynamic', w: 1 * BLOCK, h: 1 * BLOCK});
ball.add('animated', {animation: 'anim_ball'});
ball.add('ball', {});
world.addEntity(ball);
```

Figura 8: Exemplo de código de definição de entidades.

5.2.2.5 *Loop* do Jogo

De acordo com a seção anterior já é possível a criação e gerenciamento das entidades do jogo, porém ainda não temos a simulação da passagem de tempo real. Uma adição simples ao modelo apresentado nos permite simular a passagem do tempo. Uma classe *Game* gerencia a passagem do tempo em nível mais alto, inicializada pelo método *run*, faz as preparações necessárias, inicializa os objetos *World*, *System* e *Entity* iniciais e executa em um intervalo de tempo definido (30 ms

nesse caso) o método *step* da classe *World* informando o tempo decorrido (delta). A classe *World* por sua vez dispara o método *step* de todos os seus subsistemas, fazendo com que a passagem do tempo seja percebida por toda a cadeia de classes. A figura 9 é um diagrama de classes parcial que apresenta somente esse aspecto do modelo.

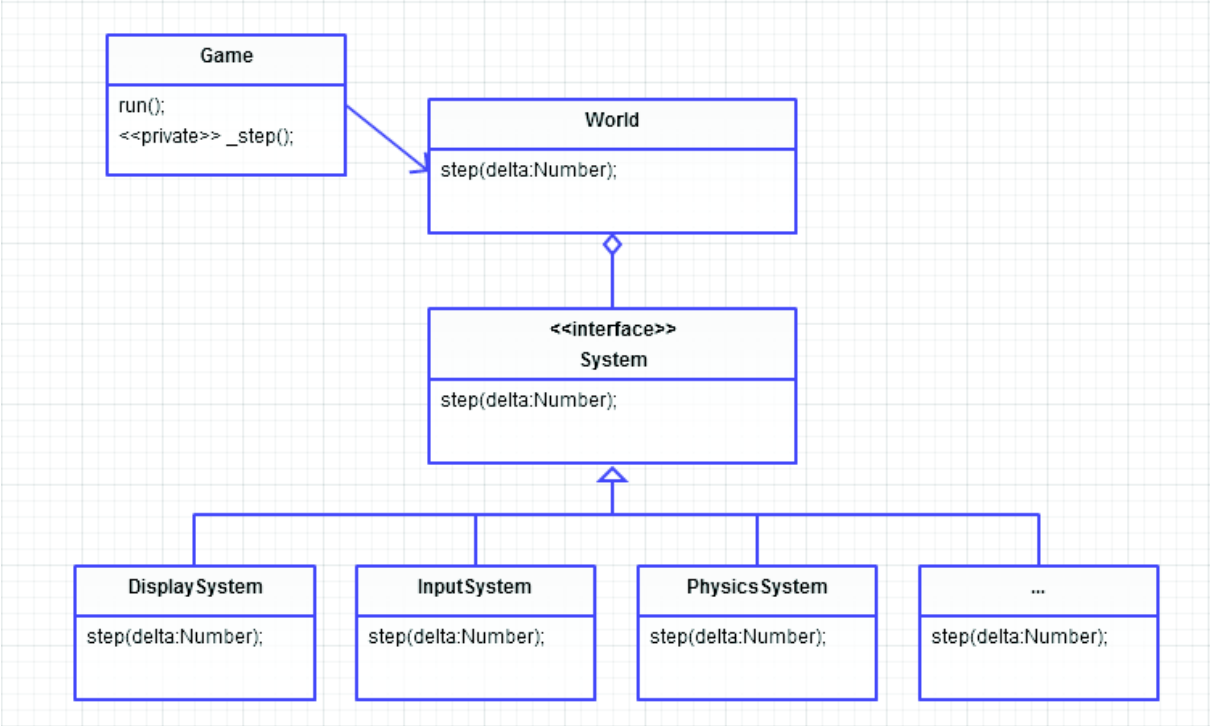

Figura 9: Classes envolvidas no loop do jogo.

5.2.2.6 Cenas

Utilizando os aspectos desenvolvidos nas seções anteriores já é possível termos um jogo, porem somente tratamos do *gameplay* do jogo, a cena jogável, e em nenhum momento solucionamos a necessidade de termos as demais cenas do jogo, por exemplo, menu principal e *loading*.

Uma solução simples adotada é de inserir um controle de cenas abaixo da classe principal *Game*. Dessa forma, essa classe deixa de tratar diretamente com o *Entity System* e passa a delegar essa responsabilidade para uma cena específica de *gameplay*. Como resultado podemos criar cenas que contenham outras responsabilidades e navegar entre elas, com a classe *Game* gerenciado a navegação. O diagrama representado na figura 10 apresenta esse modelo.

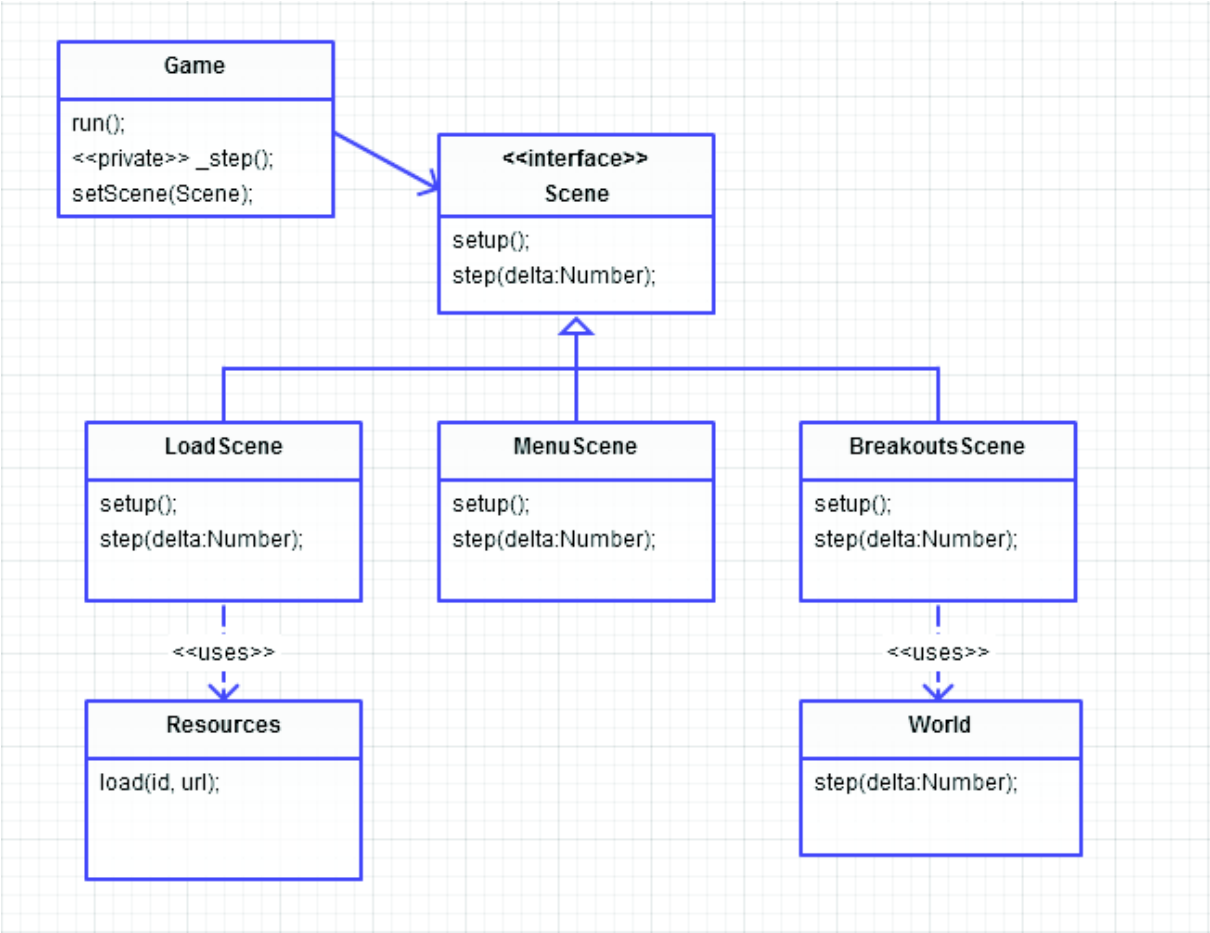

Figura 10: Modelo de Cenas do Jogo.

5.2.2.7 Lista de Sistemas

Para implementação do protótipo do jogo Breakouts atendendo aos requisitos são necessários diversos sistemas responsáveis por domínios de conhecimentos diferentes. Segue a lista dos sistemas e as suas responsabilidades:

- InputSystem: Captura os toques do usuário na tela e cria um componente ‘moveTo’ no *paddle*.
- PhisicsSystem: Gerencia a física do jogo. Move as entidades baseado em sua velocidade e posição, detecta colisões e para cada colisão cria um componente ‘collision’ na entidade. Também é responsável por mover entidades que tenham um componente ‘moveTo’.
- OutOfLevelSystem: Responsável por destruir entidades que tenham saído da área de jogo.

- GameStateSystem: Implementa a máquina de estado com as regras gerais do jogo: faz o *setup* de um level; cria ou destrói entidades com base nas regras de *gameplay*; verifica se o usuário ganhou ou perdeu o jogo.
- ExpirationSystem: Responsável por destruir entidades que tenham tempo de vida contado, por exemplo o contador inicial de espera.
- AnimationSystem: Responsável por atualizar a representação gráfica (componente 'sprite') de entidades que possuem o componente 'animated'.
- DisplaySystem: Exibe a representação gráfica de entidades que contenham os componentes 'position' e 'sprite'.

5.2.3 Implementação do Protótipo

5.2.3.1 Linguagens

Conforme a escolha da tecnologia, o protótipo foi desenvolvido em HTML5 com Javascript. Foi utilizado HTML para a exibição, utilizando um elemento *canvas* para a área de jogo, e todo código foi desenvolvido em Javascript.

5.2.3.2 Ambiente de desenvolvimento

As características do ambiente de desenvolvimento utilizado são:

- Sistema Operacional: Windows 8 Pro
- IDE: Netbeans IDE 7.4
- Navegadores para teste pré-*build* (site HTML+Javascript):
    - Firefox 24 para Windows 8
    - Navegador padrão Android 4.1
    - Navegador padrão Windows Phone 8 (emulador)
- Versionamento de Código: Git, utilizando o serviço *online* do GitHub.
- Ferramenta de Build: Phonegap Build.
- Ambiente de teste pós-*build* (aplicação Nativa):
    - Android 4.1 (emulador) - tela 480x800 pontos
    - Android 4.1 - Dispositivo Motorola D3
    - Android 2.3 - Dispositivo Samsung Galaxy i5500
    - Windows Phone 8 (emulador) - tela 480x800 pontos

5.2.3.3 Bibliotecas utilizadas:

- CreateJS: Biblioteca Javascript para simplificar o trabalho e tratar as diferenças de compatibilidade entre as implementações dos navegadores ao utilizar objetos *canvas e audio*, a função RequestAnimationFrame e demais funcionalidades para aplicações interativas do novo padrão HTML5. Foram utilizados os módulos Ticker, para implementação do intervalo de tempo do *gameloop* e EaselJS para implementação do DisplaySystem.
- Box2dWeb: Biblioteca Javascript de simulação de física. Utilizada para implementar a simulação de física no PhisicsSystem.

5.2.3.4 Publicação Utilizando Phonegap Build

A publicação utilizando Phonegap Build é bem simples. Basta ter o código em um repositório do GitHub ou arquivo compactado, criar um produto em sua conta na aplicação *web* do Phonegap Build e executar o *build*. Todo processo é *online*, dispensando instalação de qualquer produto.

Como o protótipo foi implementado em HTML5 e Javascript padrão, o processo de *build* foi fácil e transparente. Não foi necessário nenhuma alteração para empacotar o protótipo em aplicativos nativos para Android e Windows Phone 8.

5.2.4 Teste do Protótipo

O protótipo foi testado nos ambiente Android e Windows Phone 8 especificados na seção 5.2.3.2. Em todos os ambientes os testes funcionais com base nos requisitos tiveram sucesso. O protótipo atendeu plenamente aos requisitos funcionais especificados e funcionou sem diferenças perceptíveis em todos ambientes. Um detalhe negativo quanto aos requisitos não-funcionais de compatibilidade é que no dispositivo Samsung Galaxy i5500 a tela teve uma porção cortada, e não foi encontrada solução para esse problema durante o desenvolvimento desse trabalho.

Foram executados também medidas de desempenho do protótipo nos dispositivos e, pelos padrões de mercado, os resultados foram aceitáveis somente no dispositivo mais recente: entre 45 e 60 fps (*frames* por segundo) no Motorola D3

em seções de jogo de 5 minutos. No dispositivo mais antigo o resultado não foi muito adequado: entre 16 e 25 fps no Samsung Galaxy i5500 em seções de jogo de 5 minutos. Não havia dispositivo Windows Phone 8 disponível para executar os testes de performance e nesse caso foi utilizado o emulador, onde o protótipo teve performance inconsistente: entre 9 fps e 60 fps para seções de jogo de 5 minutos.

6 ANÁLISES E RESULTADOS

O objetivo geral do trabalho foi alcançado com sucesso. Foi possível implementar um protótipo de um jogo multiplataforma para dispositivos móveis que apresenta as características comuns dos jogos populares de sucesso para esse ambiente.

Através do uso das tecnologias HTML5 com Javascript e empacotamento com Phonegap Build foi possível implantar e executar com sucesso o protótipo em dois dispositivos Android diferentes e no emulador de Windows Phone 8. O protótipo apresentou desempenho aceitável em dispositivos Android mais recentes, porém não teve desempenho adequado em dispositivo com hardware mais antigo e no emulador do Windows Phone 8.

A arquitetura apresentou-se adequada e permitiu o desenvolvimento do protótipo. Graças às características dessa arquitetura foi possível adicionar aos poucos as funcionalidades no protótipo, simplificando o desenvolvimento e facilitando a extensão. Outra vantagem dessa arquitetura escolhida é que várias partes do código ficaram isoladas e independentes, facilitando a capacidade de ser reutilizadas no futuro. Com base na seção 5.2.2, vemos que seria possível extrair as partes mais genéricas dessa arquitetura para o desenvolvimento de um *framework*, mas deixaremos esse esforço para um trabalho futuro.

7 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo do trabalho foi alcançado com sucesso. Foram encontradas as características comuns dos jogos populares de sucesso para aplicativos móveis e foi possível implementar um protótipo de um jogo que atendesse essas características. Uma proposta uma arquitetura que atendesse aos objetivos específicos foi encontrada. Ela se mostrou adequada para o desenvolvimento, apesar de que

algumas melhorias quanto ao desempenho são necessárias para uma melhor compatibilidade com dispositivos mais antigos.

7.1 Trabalhos Futuros

Os seguintes itens são sugestões para possíveis trabalhos futuros:

- Usar o protótipo de jogo desenvolvido como base para o desenvolvimento de novos jogos.
- Desenvolver um framework com base na arquitetura e código do presente trabalho.

DEVELOPMENT OF A MULTIPLATFORM GAME FOR MOBILE DEVICES

Abstract: *The current market for mobile games offer a great opportunity for independent game developers who quickly and continuously develop new games and new game content. The present study aimed at developing a multiplatform game for mobile devices based on the most common characteristics found in most successful games available in mobile app stores. The goal was successfully achieved, it was possible to implement a prototype of the game with the desired characteristics using HTML5 with Javascript for Windows Phone 8 and Android. The architecture used revealed itself adequate for the development, with a few caveats regarding the performance on older devices.*

Keywords: *game. mobile devices. multiplatform. architecture. HTML5. Android. Windows Phone.*

REFERÊNCIAS

ANGRY birds: Jogo 2D mutliplataforma de gênero *puzzle*. Rovio Entertainment Ltd., 2009. Disponível em: <http://www.rovio.com/en/our-work/games/view/1/angry-birds>. Acesso em: 14 Out. 2013

APPLE App Store: Loja de aplicativos para dipositivos Apple. Apple, 2013. Disponível em <https://itunes.apple.com/>. Acesso em: 25 Set. 2013

ARTEMIS Entity System Framework: *Framework* de código aberto para gerenciamento de entidades em um universo de jogo. Gamadu.com, 2012. Disponível em: <http://gamadu.com/artemis/>. Acesso em: 01 Out. 2013

BILAS, Scott**. A Data-Driven Game Object System**. In: GAME DEVELOPERS CONFERENCE, 2002, San Francisco, California. Gas Powered Games, 2002. Disponível em: <http://scottbilas.com/files/2002/gdc_san_jose/game_objects_slides.pdf>. Acesso em: 25 Set. 2013

BREAKOUTS: Sítio *web* que reúne diversas implementações do clássico jogo Breakout em diferentes *engines* Javascript. Matt Greer, 2012-2013. Disponível em: <http://city41.github.io/breakouts/>. Acesso em: 01 Out. 2013

COCOS2D for iPhone: *Framework* de código aberto para construção de jogos 2D. Cocos2D for iPhone, 2013. Disponível em: <http://www.cocos2d-iphone.org>. Acesso em: 25 Set. 2013

COCOS2D-X: *Engine* multiplataforma de jogos 2D de código aberto. Cocos2d-x.org, 2010. Disponível em: <http://www.cocos2d-x.org/>. Acesso em: 25 Set. 2013

GAMASUTRA - The Art & Business of Making Games: Sítio *web* para designers de jogos eletrônicos. UBM Tech, 2013. Disponível em <http://www.gamasutra.com>. Acesso em: 25 Set. 2013

GAMEDEV. Gamedev.net: Comunidade *on-line* de desenvolvedores de jogos. GameDev.Net LLC, 2013. Disponível em <http://www.gamedev.net/>. Acesso em: 25 Set. 2013

GOOGLE Play Store: Loja oficial da Google para dispositivos Android. Google, 2013. Disponível em <https://play.google.com/store>. Acesso em: 25 Set.

GREGORY, Jason. **Game Engine Architecture**. Boca Raton, Florida: A K Peters/CRC Press, 2009

MARTIN, Adam. **Entity Systems are the future of MMOG development** – Partes 1, 2, 3, 4 e 5. Artigos publicados no blog T=Machine.org, 2007. Disponível em: <http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/>. Acesso em: 25 Set. 2013

MONOGAME: Implementação em código aberto do Framework Microsoft XNA 4. MonoGame Team, 2014. Disponível em: <http://www.monogame.net>. Acesso em: 25 Set. 2013

OGRE: Object-oriented Graphics Rendering Engine: *Engine* de renderização gráfica. The OGRE Team, 2012. Disponível em: <http://www.ogre3d.org/>. Acesso em: 25 Set. 2013

PLUMMER, Jeff. **A Flexible and Expandable Architecture for Computer Games**. Dissertação de mestrado. Tempe, Arizona: Arizona State University, 2004.

WEST, Mick. **Evolve Your Hierarchy**. Artigo publicado no blog Cowboy Programming, 2007. Disponível em: <http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/>. Acesso em: 25 Set. 2013